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*— Nowadays, Simultaneous Localization and Mapping (SLAM) is considered by the Robotics community to be a mature field. Currently, there are many open-source systems that are able to deliver fast and accurate estimation in typical real-world scenarios. Still, all these systems often provide an ad-hoc implementation that entailed to predefined sensor configurations. In this work, we tackle this issue, proposing a novel SLAM architecture specifically designed to address heterogeneous sensor arrangement and to standardize SLAM architecture. Thanks to its modularity and to specific design patterns, the presented framework is easy to extend, enhancing code reuse and efficiency. Finally, adopting our solution, we conducted comparative experiments for a variety of sensor configurations, showing competitive results that confirms state-of-the-art performance.**

I. Introduction

SLAM has become a mature research field with many applications areas, ranging from autonomous vehicles to augmented reality. While there are robust solutions for well posed use-cases - *e.g.* laser-based localization of wheeled robots in planar environments [1], [2] - there are scenarios in which either the robot, the environment or the requirements are so challenging that a large amount of further fundamental research is needed, as pointed out by Cadena *et al.* [3].

In this context, multi-modal SLAM can help to enhance the robustness of the system, providing redundant information about the environment. This could improve the system performances in challenging scenarios or when a sensor is not suitable to provide a specific feature - *e.g.* extracting colors from LiDAR data. Multi-cues SLAM has been explored over time by the research community and many state-of-the-art systems support two or more sensors at the same time - *e.g.* Visual-LiDAR Odometry (VLO) or Visual-Inertial Odometry (VIO). Still, the majority of these systems are meant to be used with a *predefined* combination of sensors. In this sense, they result difficult to extend or to combine with other systems.

In this paper, we propose a custom SLAM architecture that natively supports heterogeneous sensors and aims at standardizing multi-modal SLAM. The architecture allows to mix-up different cues in a plug-and-play fashion thanks to the isolation of the core SLAM modules and, hence, enhances code reuse and efficiency. In addition, exploiting specific SLAM-driven design patterns, our approach allows to embed new cues even by simply editing a configuration file. The entire architecture is oper-source and coded in modern C++[1].

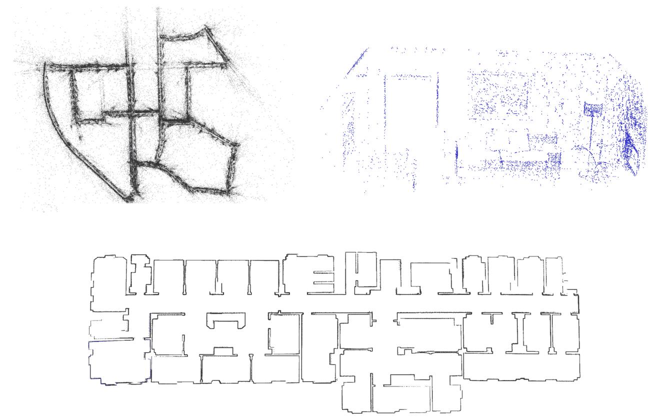


Fig. 1: Result of 3 different SLAM pipelines, all embedded in our architecture. Top left represents `kitti-00`, top right instead `icl-lr-0`. Bottom shows the map produced by the 2d-lidar pipeline on simulated data.

We validated our architecture using multiple 2D-LiDARs (in combination also with wheel odometry), stereo and RGB-D cameras, resulting in outcomes similar to ad-hoc state-of-the-art systems in all scenarios - as illustrated in Fig. 1.

The remainder of this paper is organized as follows: Sec. II presents a brief overview synopsis of multi-modal SLAM systems; in Sec. III, instead, we propose an overview of the generic building block of a SLAM system; in Sec. IV, we show the multi-sensor our architecture; finally, in Sec. V we report the results obtained with such an architecture with different sensors configuration.

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-modal 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

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: `http://srrg.gitlab.io/srrg2.html`

multiple cues, is to have a main sensor and a supplementary one. The latter is supports the system initialization or provides specific cues such as the scale. In the context of Visual-SLAM this scenario is very common nowadays. Many state-of-the-art system combine the use of a monocular camera and an Inertial Measurement Unit (IMU) to perform SLAM [4]–[7]. In this sense, the IMU data is integrated over time [8] to produce a coarse estimate of the relative motion between two frames and to infer the scale of the state. Similarly, in the work of Pire *et al.* [9], 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.* [10] developed Kimera, a SLAM framework which combines camera images (either from a monocular or stereo setup) together with IMU data to construct 3D metric-semantic maps. In the context of LiDAR-based SLAM, Zhang *et al.* [11] 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 fidelity) and then refines it exploiting scan-matching based LiDAR Odometry (LO) (low frequency, high fidelity). Newman *et al.* [12], instead, used the additional cue coming from RGB camera to compute loop-closure trough feature-based Visual Place Recognition (VPR).

In recent years, given the maturity of the SLAM problem, the research community started exploring the standardization and modularization of SLAM systems. In this sense, closed-box architecture that can deal with specific sensors in a pre-determined way leave room to dynamic multi-cues systems. Our work investigates along this research direction. In this context, Schneider *et al.* [13] 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-missions sessions. 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 it's own configuration. Still, this framework is not intended to deal with multiple sensors other than a camera and IMU.

More recently, Blanco-Claraco proposed *MOLA* [14], 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 algorithm 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 gives the freedom to the user to completely define the front-end module, who must implement some virtual functions for 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.

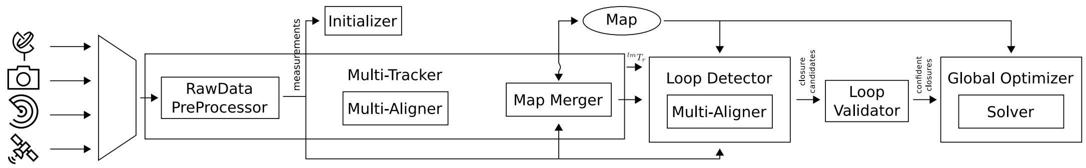
Similarly, Labbé *et al.* [15] proposed a multi-sensor graph SLAM system called RTAB-Map. The modularity is intrinsically granted by the use of Robot Operating System (ROS), by which every processing module runs over a ROS node. RTAB-Map was originally designed to be an appearance-based loop closure detection approach [16], 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 in a "passive" map manager, that takes as input odometry measurement - 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 digested at the same time in a single framework. Still, to extend the system, one has to completely develop a processing modules that given raw sensor reading provides ego-motion estimation.

Most of the concept we adopt and extend have been previously explored in the work of Colosi *et al.* [17]. Here the authors defined a taxonomy of a generic graph-based SLAM system. In this definition, each presented component is responsible for a single task, clearly defined by its input, outputs and mission. Though, the authors focused single sensor scenarios.

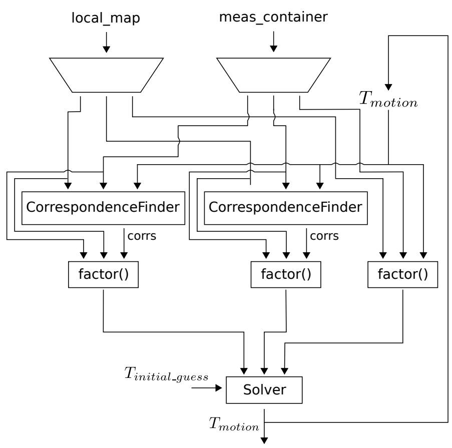
The partition of a SLAM system in components is also investigated in the survey on Younes *et al.* [18]. In this work, the authors design a generic Keyframe-based SLAM (K-SLAM) flowchart made by several building blocks. Furthermore, they explain for each of them the expected functionalities and the current state-of-the-art implementations available. Even though, this work is only restricted to monocular camera systems. Still, the idea behind the architecture is reasonably general and might be extended to more generic graph-based SLAM system, as we do in our work.

III. Taxonomy of a Graph-Based SLAM System

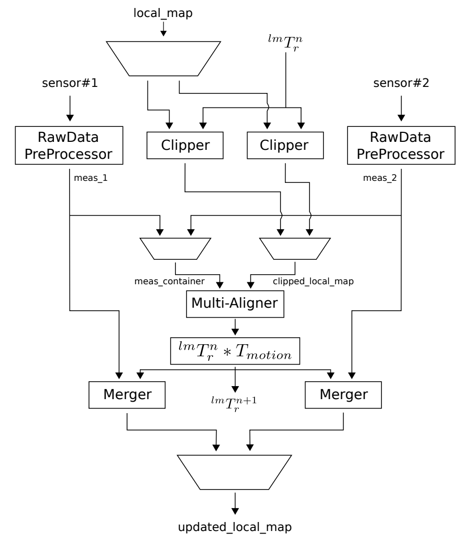
In this section, we introduce the notions we endorse in this work. A modern SLAM system is generally composed by a group of modules which process a set of shared data structures. Each processing module is in charge of performing a relatively isolated task that takes the input



(a) Multi-cues architecture.



(b) Multi-aligner.



(c) Multi-tracker.

Fig. 2: Top image: blueprint of our multi-cues 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.

data, processes them and produces some output quantities. Generally, the outcome of a SLAM system can be represented trough a factor graph [19]. In this sense, 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 two poses. To avoid unbounded growth of the factor graph, nodes are generally spawned according some kind of heuristic - *e.g.* when the robot distance between the last variable's pose is higher than a threshold. Variables in the graph, then, correspond to the pose of the robot in these *key-frames*. Furthermore, one could "attach" to each key-frame, information about the structure of the environment, represented by the *landmarks*. Therefore, each variable in the graph represents a rigid body, that we indicate as *local map*.

Colosi *et al.* [17] analyzed how a generic single-sensor SLAM system is composed. In the remaining of this section, we review these concepts, while, in the next section, we will extend them to multi-cues SLAM systems.

A. Core Modules

The workflow of a generic SLAM system should i) process raw sensor's reading and generate data in a canonical format for the rest of system, ii) estimate the relative motion between two 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. In this context the core modules involved can be summarized as follows:

RAW DATA PRE-PROCESSOR: as the name suggests, this module takes as input a raw sensor measurement and extracts suitable data-structures that can be used in the other modules. For example, given a RGB-D image, its output would consist in 3D visual-landmarks. We indicate with the term *measurement* the output of such module.

ALIGNER: this module would compute the relative motion between two sensor readings - or between a measurement and a local map. It is agnostic to the current system state since its only inputs are two entities (a fixed and a moving one) and possibly an initial guess of their offset. A possible implementation might exploit ICP to register two point-clouds - *e.g.* the one extracted from the last measurement and the previous one or the current local map.

TRACKER: it is in charge of managing and updating the current local map and generate a pose estimate of the traversing robot. Methods like VO or scan-matching are typical instances of this module.

GRAPH-SLAM: its task is to arrange local maps in a factor graph, detecting loop-closures and eventually trigger global optimization.

B. Support Modules

Each core module owns and uses several other smaller sub-modules that could be used to isolate specific tasks, enanching code reusabilty and modularity. The sub-modules combination in each core unit gives it a different "flavor", but does not affect its input-output or its role in the total workflow. In the remaining, we illustrate the principal support modules of a generic SLAM system.

CORRESPONDENCE FINDER: given two compatible entities, computes the data association between them. Many implementation of this are possible, either based on their appearance [20], [21] or geometry [22], [23].

MERGER: its task is to incorporate new entities extracted from the current sensor reading in a local map. Different mapping approaches can be exploited also in this case.

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, 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.

Many other smaller components can be isolated in a SLAM system. Examples of them are the Map Clipper (generates a local view of the input map), Local Map Splitter (triggers the creation of a new local map) or specific components to bootstrap the SLAM system - *e.g.* in monocular configuration to estimate the 3D structure of the environment from a sequence of images.

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. 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 to estimate a single quantity: the current robot's pose. Therefore, referring to Sec. III-A, the only core modules involved in this paradigm shift are Aligner and Tracker. Still, to allow heterogeneous sensors to coexists in a unique architecture, one should also define an appropriate data structure to represent multi-modal measurements and local-maps. Note that, all the support modules are relatively agnostic to the fact that the system works in a single or multi sensor configuration. Hence, in the remaining of this section, we will first provide our design to store and manage heterogeneous measurements and local-maps and then we will address the changes in the aforementioned core modules.

A. Multi-cues 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 data-type, value and by its name within a containing structure the Dynamic Property Container (DPC). Properties can represent basically anything, from Plain Old Data (POD) structures - *i.e.* a number or a point cloud - or to entire modules - named *Configurables*. Thanks to the introspection, accessing at run time a specific Property in a DPC, requires just to know its name within. Given this, we can store different types of measurement in a 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 multi-modal 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-cues Core Modules

Once addressed how to store dynamic and multi-modal 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, we can add to the Aligner and Tracker the capability of having multiple *slices* designed addresses a specific sensor reading type.

More in detail, a Multi-Cue Aligner, called Multi-Aligner for brevity, is composed by a single Iterative Least-Squares (ILS) solver and a set of *aligner-slices*. The former is in charge of optimizing the registration graph to compute the robot motion, while the latter produces factors for the different sensors to be fed into the registration graph. As in the single-sensor case, the variable to be estimated remains only the robot relative motion, still, all factors concerning different sensors will concurrently affect the estimate.

The same reasoning is applied to the Multi-Cue Tracker - shortened as Multi-Tracker. It is composed by a single local map - stored through a DPC - and multiple *tracker-slices*. Given a *measurement property container* and a *scene property container* as local map, each tracker-slice will work to process and manage a single cue from them. Fig. 2 provides a schematic illustration of the entire multi-cues workflow.

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 rear-facing. The system will have two separate Data Pre-processors. Accordingly, the *measurement property container* is composed by two Properties - relative to the point cloud extracted from the two sensors - and the local map is a DPC with two point-cloud too. The Multi-Aligner is composed by 2 *slices*, one for the front rangefinder and the other for the rear one. Both of them will compute the data association between its current measurement and the local-map. Each of the two *slices* will

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.

expose a set of constraints coming from the association, 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 relative point cloud of the local map.

Note that, thanks to this design, every other module in the architecture remains agnostic to the number of sensor 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

Beside 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 are able to automatically store the graph and the local map on disk. This is carried on by our custom-built library, that 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 disk. Note that, since also Configurables can be stored in Properties, we are able to *instanciate* an entire pipeline reading from the configuration file. Even if such 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 file on-the-go. We exploit the same Property features to provide the user with a shell to load and run configurations. Finally, Configurable entities allow to expose module *actions* that can be triggered at runtime. For example, one can pause the pipeline and save the factor graph on disk 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 these two modalities does not require to change the code. Currently, the multi-process viewing system relies on the ROS communication infrastructure to share data.

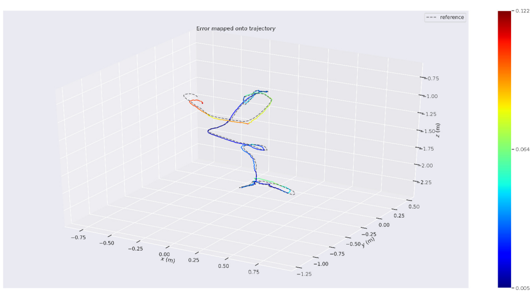
V. Experiments

In this section we provide both qualitative and quantitative results obtained though the *instanciation* of different SLAM pipelines embedded in our architecture. In this sense, the purpose of this section is to show that completely different pipelines are able not only to coexist together, but they also achieve competitive results, and, thus, there is no apparent negative impact on the systems' performances using our architecture. Thanks to the modular design of the proposed approach, the system natively allows to mix heterogeneous sensor - *e.g.* LiDAR-2D and RGBD - in a unique multi-cues pipeline. In the remaining of this section, we will show the results obtained with LiDAR-2D, Stereo and RGB-D SLAM pipeline instantiations respectively. All the experiments have been performed on laptop with Ubuntu 18.04 and GCC 7, equipped with an Intel Core i7-7700HQ @ 2.80 GHz and 16GB of RAM. Note that, all the processing in our architecture is *single threaded*.

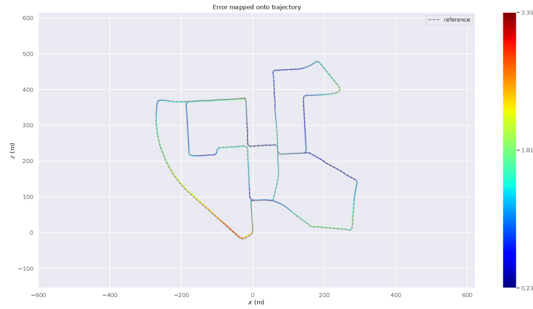
A. LiDAR-2D

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.

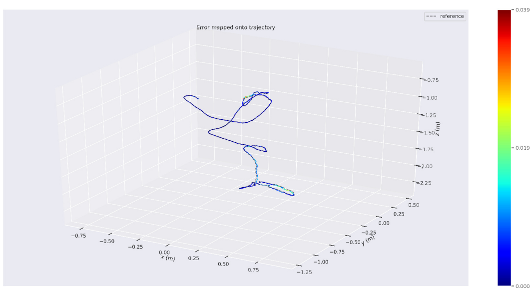
To check the performance of our pipeline, we used simulated data, gathered using ROS Stage [26]. We recorded multiple sessions with different path lengths. The simulated



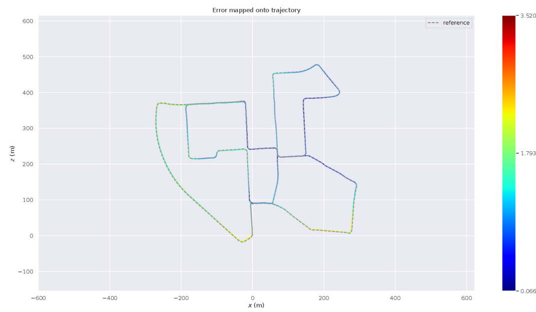
(a) ICL `lr-0`: ProSLAM.



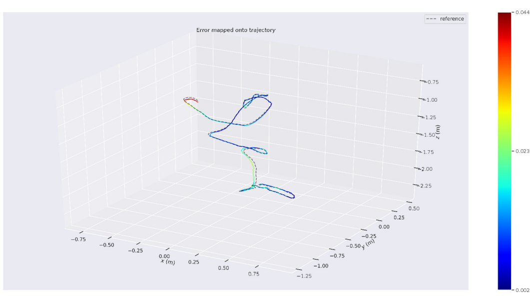
(b) KITTI `00`: ProSLAM.



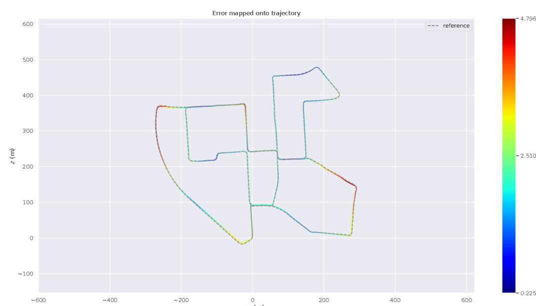
(c) ICL `lr-0`: ORB-SLAM2.



(d) KITTI `00`: ORB-SLAM2.



(e) ICL `lr-0`: our.



(f) KITTI `00`: our.

Fig. 3: Trajectory comparison between ProSLAM, ORB-SLAM2 and our approach in the context of Visual-SLAM - stereo and RGB-D.

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 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. Using the information coming from the second rangefinder increases the accuracy as expected, while system speed remains more than 10 times faster than the platform sensor frame-rate.

B. Visual SLAM: RGB-D and Stereo

Cameras represents one of the most employed sensor in the context of SLAM. Nowadays, stereo configurations and depth cameras (IR and ToF) are becoming always more used in a great variety of domains, from Robotics to consumer electronic. In the proposed architecture, we addressed both stereo and RGB-D data in a single-cue fashion. Still, in the near future we are expecting to enable multi-cues pipelines as in Sec. V-A. To evaluate the performances of our pipelines we compared the results obtained on the KITTI dataset [29] and on the ICL-NUIM dataset [30] with the state-of-the-art

APPROACH	KITTI-00 $l = 3724.187$ [m]	ICL-LR-0 $l = 6.534$ [m]
PROSLAM	ATE = 1.378 [m] RPE = 0.041 [m] **49.30** [Hz]	ATE = 0.049 [m] RPE = 0.003 [m] **62.97** [Hz]
ORB-SLAM2	ATE = **1.336** [m] RPE = **0.029** [m] 11.29 [Hz] (4 threads)	ATE = **0.007** [m] RPE = 0.004 [m] 45.28 [Hz] (4 threads)
OUR	ATE = 2.469 [m] RPE = 0.037 [m] 43.88 [Hz]	ATE = 0.016 [m] RPE = **0.002** [m] 49.98 [Hz]

TABLE II: Comparison of ATE and RPE between ProSLAM [27], ORB-SLAM2 [28] and our approach. For each approach, the last row reports the mean processing frame-rate.

system ProSLAM [27] and ORB-SLAM2 [28]. In Tab. II we reported the values of the ATE and RPE on the considered sequences, while in Fig. 3 we report a plot of the trajectories computed with each system. The comparison confirms that our architecture is able to achieve results comparable with other more mature state-of-the-art systems.

VI. Conclusions

In this paper we presented a novel architecture that aims to standardize multi-sensor SLAM. To achieve this goal, we firstly 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 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.

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 - in the context 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. 1, 5

[2] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1271–1278. IEEE, 2016. 1, 5

[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. 1

[4] 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. 2

[5] Agostino Martinelli. Closed-form solution of visual-inertial structure from motion. *International journal of computer vision*, 106(2):138–152, 2014. 2

[6] Raúl Mur-Artal and Juan D Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters (RA-L)*, 2(2):796–803, 2017. 2

[7] 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. 2

[8] 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. 2

[9] 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. 2

[10] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020. 2

[11] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2174–2181. IEEE, 2015. 2

[12] Paul Newman, David Cole, and Kin Ho. Outdoor SLAM using visual appearance and laser ranging. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1180–1187. IEEE, 2006. 2

[13] 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. 2

[14] Blanco-Claraco J.L. A Modular Optimization Framework for Localization and Mapping. In *Proc. of Robotics: Science and Systems (RSS)*, FreiburgimBreisgau, Germany, June 2019. 2

[15] 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. 36(2):416–446, 2019. 2

[16] 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. 2

[17] Mirco Colosi, Sebastian Haug, Peter Biber, Kai O Arras, and Giorgio Grisetti. Better Lost in Transition Than Lost in Space: SLAM State Machine. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2019. 2, 3

[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. 2

[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. 3

[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. 4

[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. 4

[22] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 4

[23] Belur V Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *IEEE Computer Society Tutorial*, 1991. 4

[24] María T Lázaro, Roberto Capobianco, and Giorgio Grisetti. Efficient long-term mapping in dynamic environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 153–160. IEEE, 2018. 5, 6

[25] Giorgio Grisetti, Tiziano Guadagnino, Irvin Aloise, Mirco Colosi, Bartolomeo Della Corte, and Dominik Schlegel. Least squares optimization: from theory to practice, 2020. 5

[26] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008. 5

[27] Dominik Schlegel, Mirco Colosi, and Giorgio Grisetti. Proslam: Graph slam from a programmer's perspective. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1–9. IEEE, 2018. 6, 7

[28] 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. 6, 7

[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. 6

[30] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Hong Kong, China, May 2014. 6