Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

A Modular Optimization Framework for Localization and Mapping

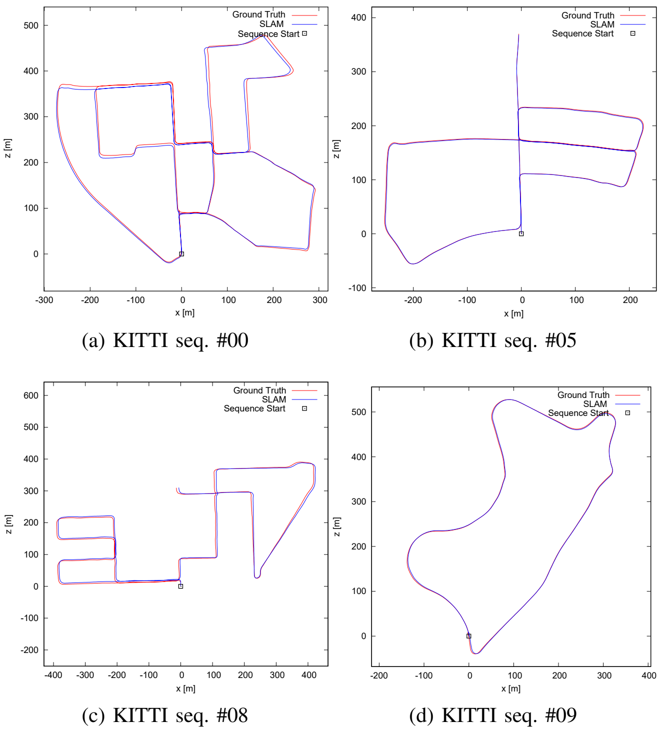
José Luis Blanco-Claraco
Engineering Department
University of Almería, Spain
Email: jlblanco@ual.es

***Abstract*—This work approaches the challenge of how to divide the problem of Simultaneous Localization and Mapping (SLAM) into its smallest possible constituents, in such a way that the reusability and interchangeability of each such module is maximized. In particular, most components in the proposed system should be not aware of details such that whether the map comprises a single global map or a set of local submaps, whether the state vector is defined in SE(2) or SE(3), with or without velocity, etc. Any number of heterogeneous sensors should be used together and their information fused seamlessly into a consistent localization solution. The resulting system would be useful for researchers, easing the development of reproducible research and enabling the quick adoption of state-of-the-art algorithms into product prototypes. Our implementation has been tested with different sensors against the KITTI, EuRoC, and KAIST datasets. In this paper we focus on an introduction to the framework and on experimental results for 3D LiDAR odometry and mapping. LiDAR SLAM for the KITTI datasets achieves typical translation errors of 1%–2% for most urban sequences, while processing the data at 1.5x the real-time rate with a reduced memory requirement thanks to our framework's capability to dynamically swap out from memory the parts of the map that are not immediately required, transparently loading them again when required. The framework will be released as open-source at https://github.com/MOLAorg/mola**

I. Introduction

Simultaneous Localization and Mapping (SLAM), structure from motion (SfM), and visual-inertial odometry have been the focus of an intense research work for the last decades [7]. Current state-of-the-art SLAM systems for a variety of sensors achieve excellent performance under challenging conditions. However, integrating an arbitrary number of sensors of different kinds into a unified SLAM solution remains an elusive goal, in part, due to theoretical-level obstacles such as different implicit assumptions of each SLAM solution regarding the robot sensory system, the underlying geometric representation of the map, or the model for the vehicle state vector.

At present, the closest to a general SLAM framework that we can find are the *graphical inference libraries*, well-known in the robotics community: g2o, introduced by Kümmerle et al. [27], GTSAM, by Dellaert [12], and Google Ceres [1]. Although they are often referred to as *SLAM back-ends*, this term will be used in this work to define to role of a SLAM component at a higher-level of abstraction than the underlying graphical models. Despite their versatility and popularity, there still exists a significant gap between the scope of those two libraries and a complete SLAM system, a gap that the present proposal intends to fill. On the other hand, one can find integrated systems which built upon the former graphical inference libraries to provide SLAM solutions; they are described in section II.



(a) KITTI seq. #00

(b) KITTI seq. #05

(c) KITTI seq. #08

(d) KITTI seq. #09

Fig. 1: Final maps obtained by the proposed system for a subset of the KITTI odometry datasets. Refer to the text (section VI-B) for details.

Next follows a summary of the main design goals addressed with the proposed system, whose open-source implementation is dubbed MOLA (*Modular Optimization framework for Localization and mApping*): Having a unified mathematical framework (and C++ API) for SLAM, decoupled from the use of SE(2) or SE(3) poses, or the choice between a global map or relative submapping; Ensuring the reusability of each component of a SLAM system, e.g. an ORB-based vision loop-closure detector such as [19] should be usable no matter whether the main map is built from LiDAR or visual-based SLAM; Supporting different sensors: 2D and 3D LiDAR, monocular and stereo cameras, odometry, IMU, and GPS; Allowing the user to select among different state vector representations: only poses, poses and linear velocity, or poses and linear and angular velocity; Having built-in support for relative coordinates (i.e. submapping), also allowing the storage of semantic information and a hierarchical representation of maps; Support for the most common map entities (structure-less, point landmarks, lines, planes), extensible by the user to other types; Supporting different optimization modes: smoothing vs. batch optimization; Having convenience tools: built-in stack trace reports on exceptions, detailed performance reports of each code segment, etc; Exposing a uniform API to SLAM modules, independently of sensory data coming from a live sensor or one of the major dataset formats (i.e. KITTI [20], EuRoC [6], ROS bag's, MRPT rawlog's); Transparently swapping out from dynamic memory, and back in when required, the largest part of data associated with map areas that have not being recently accessed.

The rest of this article is organized as follows. First, section II reviews the most prominent related works. In section III, we address the question of how to divide a SLAM system into modules to maximize the overall versatility and reusability, while section IV exposes the mathematical details of how to express different flavors of a SLAM system in the unifying framework of factor graphs. Some of the currently-implemented modules in our system are described in section V, experimental results are shown in section VI, while section VII ends with some final remarks and discussion of future works.

II. Related works

Only a selection of works that are useful to motivate the need for a versatile SLAM framework are discussed next. More comprehensive reviews can be found elsewhere ([3, 21, 13, 7]).

A. Visual SLAM

Regarding monocular visual SLAM, Montiel et al. [29] proposed an EKF-based solution where visual landmarks are modeled as points parameterized by polar coordinates with inverse depth, dramatically improving the statistical validity of the unimodal Gaussian error model for landmarks.

Monocular SLAM (without any additional sensor) has the limitation of not being able to solve for the scale of the map. Workarounds have been proposed for the case of cameras attached to vehicles with non-holonomic constraints, with Scaramuzza et al. [33] proposing exploiting such constrains to solve for the scale, when the vehicle moves with a curvature large enough to render the scale observable.

The kinematics of the camera, i.e. its linear ($\mathbf{v}$) and angular ($\boldsymbol{\omega}$) velocities, have been accounted for since the earliest successful MonoSLAM system [11, 26], for the key advantage that this information represents while solving the data association. In a recent seminar paper, Leutenegger et al. [28] proposed a visual-inertial SLAM solution where the camera state vector is augmented to account for the IMU biases, with a formulation that allows IMU measurements to be integrated into regular least-squares optimizers seamlessly. Their method renders SO(3) attitude increments and $\mathbb{R}^3$ velocity increments directly observable, except for a gauge freedom for the symmetry around the gravity vector. In any case, it reduces the seven degrees of gauge freedom of monocular SLAM [36] to only four degrees [42]. It also proposed a sliding-window optimizer where past poses and past IMU biases are marginalized out to keep the system complexity tractable. This idea was further improved with the introduction of a formal Lie group formulation for IMU factors in [17], reusing the concept of smart factors formerly introduced in [8].

The ORB-SLAM system, presented in [32, 31], comprises visual front-ends capable of tracking FAST features and efficiently matching them between consecutive and distant keyframes by comparing their associated ORB descriptors, using bags of binary words [19] for efficient place recognition. Landmarks are parameterized as points in Euclidean absolute coordinates, with the highly non-linear problem of monocular initialization dealt with by means of a special algorithm that ensures that mapping is not started until the parallax is enough to avoid numerical instabilities. The *maplab* framework [34] is another C++ extensible system aimed at researchers in visual-inertial SLAM.

Noteworthy are the so-called *direct* and *semi-direct* methods to visual SLAM [10, 16], which directly minimize the photometric reprojection error of all (or most part of) the image pixels to optimize for pose changes over time, in contrast to the more traditional approach based on a discrete set of features, i.e. points, lines, or planes. Finally, it must be also stressed that visual SLAM is possible using arbitrary rigid objects as features, e.g. see [38].

From this brief review, it is clear that the map model may include features or not (the so called *structure-less* problem), it may include just the SE(3) pose of the robot for each time step, or it may also hold additional kinematic information, or even IMU-related parameters, i.e. biases. All this has been accounted for in the proposed framework.

B. LiDAR SLAM

Despite the exceptional importance of vision-based SLAM, many other sensors have been used in the SLAM literature. LiDAR, in its 2D and 3D versions, irrupted into the mainstream robotics community about three and one decades ago, respectively, and their accuracy and robustness still make them an excellent sensor for mapping and localization, hence their popularity.

Current 3D LiDARs provide a large amount of raw information, which is typically first converted into 3D pointclouds before any further post-processing and interpretation. By their nature, feature-based SLAM is seldom applied to LiDAR point clouds; instead, it is more common to address SLAM in its structure-less version (with names such as pose-SLAM or graph-SLAM [21]) where each pointcloud is registered to its neighbors and the relative poses inserted as SE(3) pose constraints into the graph [41]. A popular implementation of

Velodyne-based SLAM is LOAM, introduced by Zhang and Singh [41]. *Google Cartographer* [22] and SegMap Dubé et al. [14] are other public SLAM frameworks for 2D/3D LiDARs.

The LiDAR module proposed in the present paper (see Section V) has many similarities to the aforementioned work Zhang and Singh [41], although our modular C++ interface between raw sensors and front-ends makes it possible to fully exploit features such as the natural ordering of point clouds into rings (one per individual laser in a 3D LiDAR), something that is not possible with the current publicly-available implementation of LOAM. Furthermore, it makes straightforward to use more than one LiDAR onboard, in contrast to existing implementations.

C. Map management

The idea of using local maps instead of a single, absolute frame of reference is recurrent in the literature. For example, Clemente et al. [9] uses an EKF to build local maps from monocular images, then merges them in a hierarchical manner. Eade and Drummond [15] also achieved a successful real-time monocular SLAM system based on a hierarchical decomposition of the map into submaps, which they called *nodes*. Relative bundle-adjustment has been also proposed in [4, 35]. In section 3 we introduce a unifying view of how different sensor front-ends can interact with the SLAM back-end in a way that is independent of whether absolute or local coordinates are used, and for the latter case, what particular policy is followed to split the world into submaps. Our solution represents a pragmatical tradeoff between "classic" key-frame-based pose-graph in global coordinates [27] and continuous-time relative SLAM [2]. The result is equivalent to one of the applications of smart factors, as introduced by Carlone et al. [8], for summarizing long sequences of poses with a selected set of "key-frames" to be optimized, although without explicitly removing the intermediary poses, which would still remain inside their corresponding submaps.

Walcott-Bryant et al. [40], among others, focuses on the particular problem of pose graph maintenance for non-static environments. That work proposed an algorithm for detecting segments in the sequence of key-frames that can be safely removed when they represent parts of the environment that has changed and, therefore, are not useful for localization. Our framework takes into account the important observation that no key-frame should be used as a reference frame (neither global nor for its submap), since all key-frames are subject to an eventual deletion. Hence our proposal for a different kind of map entity for reference purposes only (see Section III).

D. Optimization

Strasdat et al. [37] showed that batch optimization (i.e. "bundle adjustment" [39], or "Graph-SLAM" [21]) achieved better accuracy in comparison to filtering for most large-scale SLAM problems; hence, our framework must (and does) support both, batch optimization and smoothing [25]. Filtering is left as a potential future work. The foundations of least-squares minimization, Gauss-Newton or Levenberg-Marquartd algorithms, are well-known and explained elsewhere [36, 27], as they are their extensions, by means of Lie group "retractions", to non-Euclidean state spaces [17].

III. Factoring out SLAM: the MOLA architecture

Once the last section has given a brief overview to the most relevant ways in which SLAM has been addressed, this section introduces a rationale of why a SLAM solution should be split into modules, exposing the proposed software architecture. Note that currently-implemented modules are introduced in Section V.

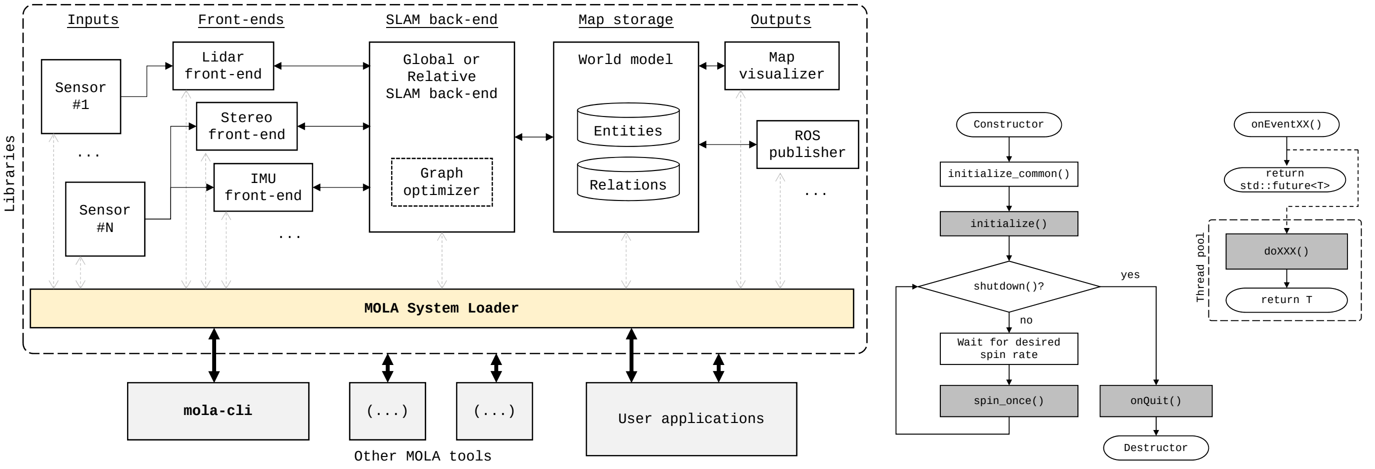
A. General overview

An overview of the proposed MOLA architecture is given by Figure 2(a). Firstly, we define the MOLA *system* as a set of *modules*, each module being the instantiation of a particular C++ class implementing the `ExecutableBase` interface. Based on the differentiated role of each module in the SLAM system, a number of prototypical virtual base classes are provided for users to define their own modules of each type, ensuring the existence of a common API as the key for compatibility, easy reusability and interchangeability. The *system loader* is in charge of interpreting a *SLAM-problem configuration file*, loading the required libraries, finding and creating the required modules and launching their life cycle routine. It also allows running modules to find each other (either by name or by service type) for peer-to-peer connections to be established and allow the information and signals to flow forth and back throughout the system.

Note that the entire system is designed as a single process, with each module being run in at least one thread, and message passing occurring with a zero-copy protocol by means of `std::shared_ptr<>` smart pointers. As illustrated in Figure 2(b), all modules inherit from a common base, `ExecutableBase`, which provides them a standardized life-cycle that allows an ordered system start-up and tear-down, the opportunity to work synchronously by running an algorithm at an exact constant rate –in `spin_once()` – or to dynamically react to incoming data or service requests by means of events, which may be attended immediately (blocking) or deferred to a per-module thread pool (non-blocking).

The entire system is accessible to a robotics researcher or application developer in two modes: from the command-line program `mola-cli`, which takes a SLAM problem definition file and executes it, or directly using the MOLA system loader as a C++ library that can be configured programatically.

Note that system dependencies were kept as reduced as possible, considering the ambitious scope of the proposal. In particular, MOLA depends on GTSAM and MRPT and requires a C++17 compiler. ROS1 and ROS2 were intentionally left out as *optional* dependencies, allowing the interconnection of a MOLA system with a ROS ecosystem but not imposing it as mandatory.



(a) (b)

Fig. 2: (a) A typical set-up with the proposed modular SLAM architecture. Refer to Section III for discussion. (b) Module life-cycle, as imposed by the common base class `ExecutableBase`. In most modules, API calls can be attended either synchronously (blocking) or asynchronously (non-blocking) on a local worker threads pool, in which case `std::future<T>` objects are used for inter-module synchronization. Refer to Section III for discussion.

B. MOLA arquitecture

Next, we describe the rationale behind why the SLAM system, inside the dotted box in Figure 2(a), is split into modules.

Starting from the leftmost side, we find *input* modules. They provide the raw data from either real sensors (e.g. cameras, GPS readings, odometry) or from offline datasets. Each input module can publish one or more kind of *observations*, pieces of timestamped raw sensory data. One sensor output can be used by several consumers, and each consumer subscribed to more than one sensor. Each kind of sensor has its own standardized C++ class that includes the raw data and key calibration data, e.g. distortion parameters for images, or pose within the vehicle body for a LiDAR or IMU. Redundant as it might seem, keeping the calibration and raw data together makes any data consumer independent of knowing details about the exact sensory system, avoiding the need for sensor-related configuration parameters on all modules except those directly in charge of accessing them.

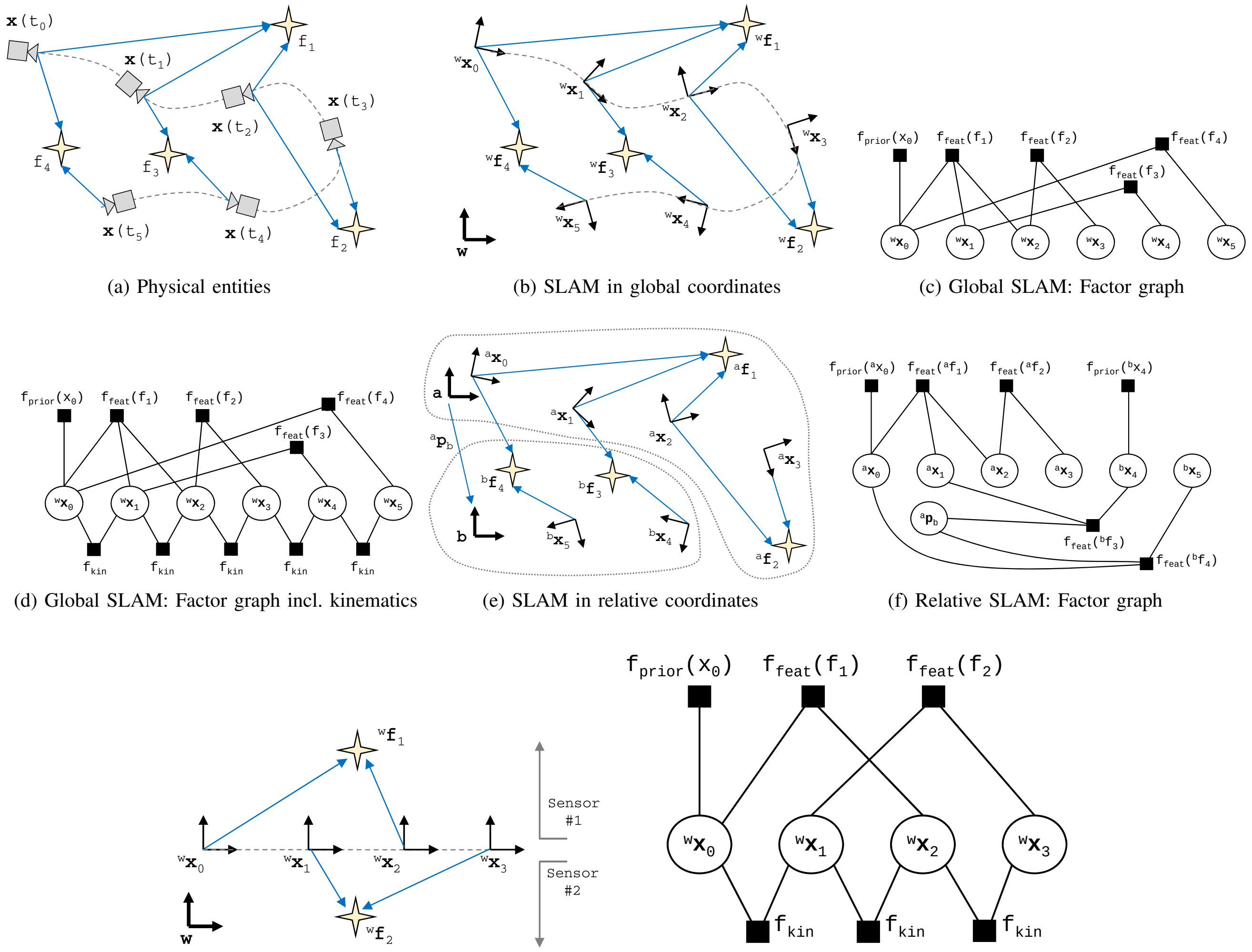
Following the direction of the data stream to the right, we find the layer of *SLAM front-ends*. These are the most hard-to-develop modules in the system, in charge of taking raw data and converting them into *SLAM API calls* (see Table I) to the *SLAM back-end*. The easy reusability of these front-ends, allowing several of them (of the same or different kind) to run concurrently depending on the number and kind of sensors installed on a robot or vehicle, and still feeding a single SLAM solver, is probably the single most significant contribution of the proposed system.

There should be only one *SLAM back-end* module in any given system (except if used for multi-robot SLAM), providing a particular mapping between MOLA *SLAM API calls* and an underlying graphical model. As discussed in Section IV, basically we can devise two options: SLAM in global coordinates, and SLAM with submaps. Notice that, in any case, the front-end will not notice the difference. Table I gives a glimpse into the kind of interactions that are possible between front-ends and back-ends.

`addKeyFrame()`	Creates a Key-frame in the World-Model, or returns an existing one for a matching timestamp if another sensor front-end already created it.
`addFactor()`	Creates a constraint (factor in a factor graph) of the given type and between the given variables. Note that factors at this level are abstract since the particular state-space (SE(2) vs SE(3), etc.) is unknown.
`advertiseLocalization()`	Publish a current pose estimate with respect to any existing key-frame.
`onSmartFactorChanged()`	Notify the change in the internal data of a smart factor [8], which summarizes multiple individual factors and variables.

TABLE I: Excerpt of key SLAM API calls that a front-end can invoke in a SLAM back-end.

Finally, the *World Model* module stores what is normally known as the "map" itself, although it is done in a way generic enough to allow the maximum number of SLAM algorithms to work with it. The World Model comprises two sets of objects: *entities* and *relations*, which most often can be thought of as directly mapping to *variables* and *factors* of a factor graph. Most common 3D *entities* are: `RefPose3` (a frame of reference), `RelPose3` (a relative pose, such as the relative pose of a sensor on a vehicle in a self-calibration SLAM problem), `RelPose3KF` (a robot key-frame, relative to a frame of reference), a `RelKinPose3KF` (a key-frame, with velocity information), or `LandmarkPoint3` (a point landmark). Section IV will show how a real SLAM problem can be written down as different combinations of these entities



(a) Physical entities

(b) SLAM in global coordinates

(c) Global SLAM: Factor graph

(d) Global SLAM: Factor graph incl. kinematics

(e) SLAM in relative coordinates

(f) Relative SLAM: Factor graph

(g) Set-up with two asynchronous sensors

(h) Two sensors + kinematic model: Factor graph

Fig. 3: Different parameterizations of a SLAM problem, together with their corresponding factor graphs. See Section IV.

and factors for absolute and relative SLAM. The most common *relation* used in this paper is FactorRelativePose3, a relative pose constraint between two key-frames. Note that *relations* can be used to reflect *semantic* information in addition to those having a pure geometrical meaning, e.g. the inclusion of several key-frames into a place with a significance for human users, such as ”kitchen” or ”bedroom” [18]. Keyframes keep accurate timestamp information and the raw sensory data captured by the robot at that instant. Pure metric maps, such as point clouds, occupancy grids, octomaps, etc. can be always recovered from these data, hence they are not explicitly stored in the map. Those key-frames whose raw sensory information has not been accessed by any front-end or back-end during a certain time frame, are swapped out to disk using an efficient binary serialization protocol, and are transparently re-loaded when needed later one, e.g. when closing a large loop. This is only possible if the map is stored in its own independent module.

IV. Graphical Models

In this section we discuss how a SLAM problem can be modeled as different factor graphs depending on the choice for: (a) global vs. relative coordinates, and (b) components of the state-vector. SLAM back-ends in the proposed system are in charge of converting SLAM API calls like those in Table I into their corresponding graphical models. Users could extend the possibilities in the future, if so needed, by modifying the existing backend modules or writing new ones from scratch.

Consider the SLAM problem in Figure 3(a), with a camera describing a trajectory while observing four landmarks. Firstly, our system assumes a discrete time implementation, hence the need to define *key-frames* whenever the front-end so requests it. A representation of the problem in global coordinates is shown in Figure 3(b): a reference frame $\mathbf{w}$ is created as the absolute frame of reference (this is what a RefPose3 means in the WorldModel), and the i-th key-frame is defined as a relative pose ${}^{w}\mathbf{x}_i$ with respect to it. Each such variable can be a SE(2) or SE(3) pose, or the Cartesian product of such manifolds with the kinematic variables ${}^{w}\mathbf{v}_i$ ($\mathbb{R}^3$ velocity vector w.r.t. $\mathbf{w}$) and ${}^{b}\boldsymbol{\omega}_i$ ($\mathbb{R}^3$ angular velocity vector w.r.t. body frame). Notice that the cost of including the angular velocity into the state vector is not worth when an IMU is available.

In the particular case of not including kinematic variables in the state vector, a global SLAM problem such as Figure 3(b) should be mapped into the factor graph in Figure 3(c), with a prior factor for the first key-frame and with one smart factor per observed feature. Alternatively, landmarks could be also added as variables and simple factors added for each pose-landmark observation, but this approach is generally more costly for large maps [8]. If kinematics are to be included into the state vector, the SLAM back-end will add kinematic factors between consecutive key-frames, as shown in Figure 3(d). Our current implementation employs a constant-velocity factors as kinematic constraint, with the following error function:

$$\mathbf{h_1}(\mathbf{x}_i, \mathbf{x}_j) = \begin{pmatrix} \mathbf{p}_i + \mathbf{v}_i \delta_t - \mathbf{p}_j \\ \mathbf{v}_j - \mathbf{v}_i \end{pmatrix}_{6\times 1} \tag{1}$$

where $\mathbf{x}_i = (\mathbf{R}_i, \mathbf{p}_i, \mathbf{v}_i)$ is the state vector for the i-th key-frame, δ_t is the time interval between both key-frames, and with Jacobians:

$$\frac{\partial \mathbf{h}_1}{\partial \mathbf{p}_i} = \begin{pmatrix} \mathbf{I}_3 \\ \mathbf{0}_3 \end{pmatrix} \qquad \frac{\partial \mathbf{h}_1}{\partial \mathbf{v}_i} = \begin{pmatrix} \delta_t \mathbf{I}_3 \\ -\mathbf{I}_3 \end{pmatrix} \tag{2}$$

$$\frac{\partial \mathbf{h}_1}{\partial \mathbf{p}_j} = \begin{pmatrix} -\mathbf{I}_3 \\ \mathbf{0}_3 \end{pmatrix} \qquad \frac{\partial \mathbf{h}_1}{\partial \mathbf{v}_j} = \begin{pmatrix} \mathbf{0}_3 \\ \mathbf{I}_3 \end{pmatrix} \tag{3}$$

In case of a kinematic state-vector that includes the angular velocity ${}^{b}\boldsymbol{\omega}_i$ to model the evolution of the SO(3) orientation $\mathbf{R}_i$ over time, we would need to use the equation, from rigid-body kinematics, $\dot{\mathbf{R}}_i = \mathbf{R}_i {}^{b}\boldsymbol{\omega}_i$, which using a first-order approximation would lead to the following error function for a constant velocity model:

$$\mathbf{h_2}(\mathbf{x}_i, \mathbf{x}_j) = \begin{pmatrix} \log\left(\mathbf{R}_j^{-1} \mathbf{R}_i \exp\left({}^{b}\boldsymbol{\omega}_i \ \delta_t\right)\right) \\ \mathbf{p}_i + \mathbf{v}_i \delta_t - \mathbf{p}_j \\ \mathbf{v}_i - \mathbf{v}_j \\ \mathbf{R}_i {}^{b}\boldsymbol{\omega}_i - \mathbf{R}_j {}^{b}\boldsymbol{\omega}_j \end{pmatrix}_{12\times 1} \tag{4}$$

Relative pose factors follow the common practice of being linearized on the SE(3) Lie-group retraction (typically, the matrix logarithm) for residuals $\mathbf{M}_{ij}^{-1}\mathbf{P}_i^{-1}\mathbf{P}_j$, with $\mathbf{M}_{ij}$, $\mathbf{P}_i$, and $\mathbf{P}_j$ being the SE(3) matrices for the measurement and the i-th and j-th key-frame estimated poses, respectively.

If using a relative SLAM backend, the problem of Figure 3(a) should be mapped into something like Figure 3(e). In this example, there are two frames of coordinates ($\mathbf{a}$ and $\mathbf{b}$), each being the frame of reference of the key-frames in its own submap. Regarding the corresponding factor graph, in Figure 3(f), notice the inclusion of two prior factors (one for each submap), and the existence of a new variable to model the relative pose of the submaps. Factors for landmarks seen from key-frames in different submaps will depend on these relative-pose variables as well, rendering the underlying linear problem Hessian matrix less sparse. Therefore, a criterion to split key-frames into submaps should be the minimization of co-visibility across submap boundaries.

Note that enhancing the state-vector with kinematic information, and using a kinematic model (e.g. constant-velocity) automatically solves the otherwise indeterminate problem of fusing information from asynchronous sensors without shared landmarks, e.g. when they observe disjoint parts of the environment, a set-up sketched in Figure 3(g). As shown in the factor graph of Figure 3(h), it is only by means of the kinematic factors that information can be shared between the otherwise independent sequences of poses.

V. Implemented modules

This section briefly introduces some of the key libraries and modules that are available in our open-source implementation.

`mola-kernel` library: It comprises the `WorldModel` module, and declarations of the most-common entities and relations, as well as the generic containers `Entity` and `Relation`, implemented as C++17 `std::variant` to allow the efficient storage of large number of such objects without dynamic memory, but retaining the advantages of polymorphism. A special type is also included into the variant to allow for user-defined types, at the cost of being allocated dynamically.

Module `G2ODataset`: Allows reading a G2O plain-text file with a SLAM problem, and re-plays it, by mapping the vertex and edge definitions into MOLA SLAM API calls.

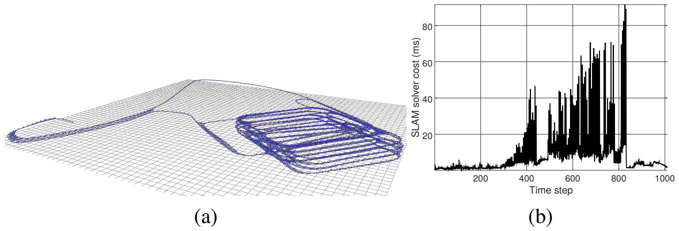
Module `EurocDataset`, `KaistDataset`, and `KittiOdometryDataset`: Provide image, LiDAR, and IMU readings from the EuRoC [6], Kaist [24], and the KITTI datasets [20], respectively.

Module `GenericSensor`: An interface to `mrpt-hwdrivers` generic sensor classes, allowing the direct connection of many kinds of monocular and stereo cameras, IMU sensors, SICK and Hokuyo scanners, and OpenNI-enabled RGB-D cameras. Noteworthy is the support for Velodyne LiDARs, both live or from offline PCAP logs, including support for dual-return scans.

Module `PreintegratedIMU`: A front-end to parse raw IMU data and feed a preintegrated IMU smart factor.

Module `ASLAM_gtsam`: An implementation of a SLAM back-end for absolute coordinates, based on the GTSAM library. A number of custom factors have been defined to map the different kinematic models described above. The user can select two solver modes: a full batch estimator (based on Levenberg-Marquardt), or the iSAM2 smoother.

Module `VisualORB_Stereo`: A front-end to handle stereo cameras. It takes stereo image, rectifies them if needed, and looks for FAST features, matching them by their ORB descriptors. We have experimented with two working modes: (a) by using a local feature optimizer and sending the resulting SE(3) pose constraints to the SLAM back-end, and (b), by directly storing visual smart factors in the World Model and relying on the SLAM back-end solver. Preliminary results shown

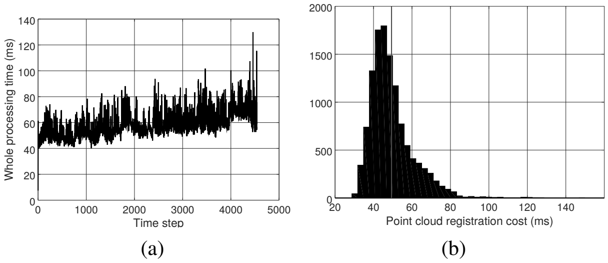


(a) (b)

Fig. 4: (a) View of the final 3D map obtained for the "garage" dataset from [27]. (b) Computational cost of the SLAM solver.

in the accompanying video are for the former alternative, but we expect the latter to eventually become the definitive, most robust solution.

Module `LidarOdometry3D`: A front-end to perform odometry and mapping from 3D point clouds from a Velodyne LiDAR or alike. A direct use of ICP for such point clouds leads to unacceptable results due to both, computation time, and accuracy. In particular, the strong differences in sample densities inherent to 3D radial scanners may lead to poor alignments if nearby points dominate the cost function. To overcome this, we devised a multi-layer ICP algorithm, where each pointcloud to be registered is split into three layers ("edges", "planes", and "raw-decimated") and each layer is registered against points in its corresponding layer on the other pointcloud. For each input pointcloud, we divide the 3D space into a regular grid of voxels (of $1 \times 1 \times 1$ meters), and all points in each voxel are firstly decimated and then stored into one of the layers depending on a classifier. In particular, each voxel is classified as either edges or planes by checking the ratio between the largest and smallest eigenvalues of the covariance matrices that model the dispersion of points in the voxel. Finally, all voxels are also stored into the "full-decimated" layer. The intuition behind this algorithm, inspired by [41], is that by drastically reducing the number of potential candidates for each subset of points, the alignment can be done much more efficiently, and with a higher chance of selecting a correct match for each point. We use KD-trees to look for candidate pairings in each layer, and store the point-clouds (and their precomputed KD-trees) in the WorldModel key-frames, to ease posterior checks for potential loop-closures, triggered by the detection of key-frames within a given threshold distance whose topological distance in the WorldModel graph (computed with the Dijkstra algorithm) is large enough to look like a loop-closure candidate. Checks for loop-closures and additional SE(3) constraints with past key-frames are queued in a worker thread pool to avoid delays in the LiDAR odometry. Most of this functionality (i.e. odometry, loop closure, classifier) will be split into different modules to increase the possibilities of researchers improving the whole 3D LiDAR frontend. At present, at the core of the ICP algorithm we use the OLAE method [30], an alternative to the well-known Horn's quaternion method [23], but relying on the Gibbs/Rodrigues vector and allowing the direct integration of plane normals and line directions seamlessly with point-to-point correspondences. More details on this proposed scheme can be found in a separate technical report [5].



(a) (b)

Fig. 5: Selected performance results from our LiDAR front-end processing the KITTI seq. #00.

VI. Experimental results

This section exposes some results obtained with the proposed SLAM system for different datasets and set-ups. All performance statistics have been collected using the MOLA built-in profiling system, which enables the optional generation of detailed performance reports for most function calls. Reported CPU times are for an Intel Core i7-3770 at 3.40GHz. The MOLA system creates about 20 threads for each experiment, but the average number of active threads is within 1 and 4, with most other threads being only briefly active for GUI updates and to allow asynchronous API calls. The accompanying video illustrates these experiments, as well as preliminary results for stereo SLAM.

A. Garage G2O dataset

The results in Figure 4 illustrate the final map obtained for the "garage" dataset [27] (in G2O format), together with the cost of updating the smoother (iSAM2) for each time step inside the `ASLAM_gtsam` module.

B. LiDAR-based odometry & SLAM

We validated our entire system, and in particular the LiDAR odometry and mapping front-end, with the KITTI datasets. The configuration of the MOLA system is defined in Listing 1, and some example maps (compared to ground-truth) are shown in Figure 1. The translation errors for each dataset sequence have been evaluated using the scripts provided by Geiger et al. [20], and are summarized in Table II. It can be seen that our method performs well in urban, quasi-static environments, with an excellent rotational error in all cases and a translation error in the range 0.4% to 2.0% for 9 out of the 11 sequences. The large error in Seq. 01 is due to a vehicle overtaking at the highway in a particularly feature poor segment.

Regarding the real-time performance of the LiDAR front-end, Figure 5(a) shows how one entire iteration of our front-end takes an average of 60 ms, allowing processing the original KITTI LiDAR data stream at almost twice its original rate of 10 Hz. Figure 5(b) shows the histogram (and the average value as a vertical line) for one of the most important piece of the

Seq. no.	00	01	02	03	04	05	06	07	08	09	10
Trans. error (%)	1.09 %	12.8 %	1.37 %	1.32 %	1.98 %	0.81 %	0.38 %	0.88 %	1.59 %	1.59 %	4.09 %

TABLE II: Summary of translation and rotation errors for the KITTI datasets using the LiDAR sensor only.

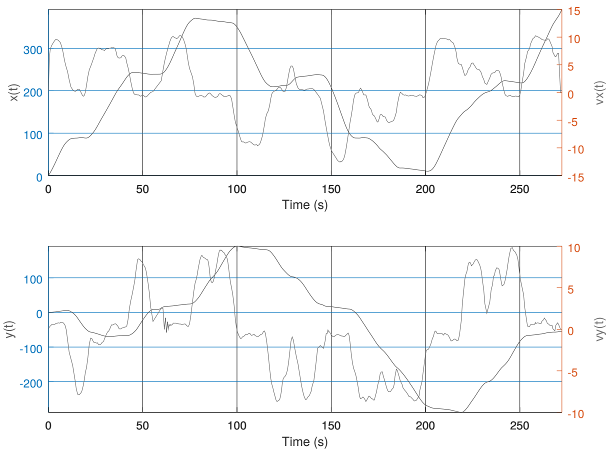


Fig. 6: An extract of pose and kinematic state (velocities) estimated by our generic SLAM back-end for the KITTI #00 sequence when selecting a the `DynSE3` state vector model.

front-end: the multi-layer ICP algorithm. Regarding the voxel-based filter to classify incoming point-clouds voxels into edges and plane patches, its average processing time is 5 ms.

C. Kinematic model

To demonstrate how the state-space of the SLAM solution can be easily changed in the proposed framework, we replaced `SE3` with `DynSE3` in the `state_vector` parameter of the SLAM back-end in Listing 1, and re-processed the KITTI seq. #00. As a result, the state space for each timestap $\mathbf{x}_i$ is extended with velocity information, which is then automatically estimated using the constant velocity model discussed above; refer to Figure 6 for a plot of the estimated velocity.

D. Smart memory management

Finally, Figure 7 illustrates how the memory requirement of our LiDAR mapping solution is drastically reduced from 4 GiB to 0.5 GiB thanks to the automatic map swapping-out mechanism implemented in the WorldModel module. This is a key feature required when handling maps of large areas, e.g. for autonomous vehicle navigation.

VII. Conclusion and Future Works

This paper presented a proposal for a factorization of SLAM into a set of so-called “modules”, with their roles and interfaces designed such that their reusability is maximized for a large number of possible combinations of SLAM problems. Experimental validation has demonstrated that the system is functional in its present form and capable of processing LiDAR datasets. Future works include the completion of a relative-SLAM module, and the development of modules for other sensors such as odometry, GPS, IMUs, monocular, and stereo cameras.

```
modules:
  - name: backend # -- SLAM back-end --
    type: ASLAM_gtsam
    execution_rate: 1 # update rate [Hz]
    params:
      # iSAM2 vs Lev-Marq
      use_incremental_solver: true
      state_vector: SE3
  - name: map   # -- World Model --
    type: WorldModel
    params:
      age_to_unload_key-frames: 50 # [s]
  - name: lidar_fe # -- SLAM front-end --
    type: LidarOdometry3D
    raw_data_source: kitti_input
    raw_sensor_label: lidar
    params:
      # ...
  - name: kitti_input # -- Data sources --
    type: KittiOdometryDataset
    execution_rate: 50 # Hz
    params:
      base_dir: ${KITTI_BASE_DIR}
      sequence: 00
      time_warp_scale: 1.5
```

Listing 1: Example configuration (YAML) file for running LiDAR odometry/SLAM on sequence 00 of the KITTI dataset.

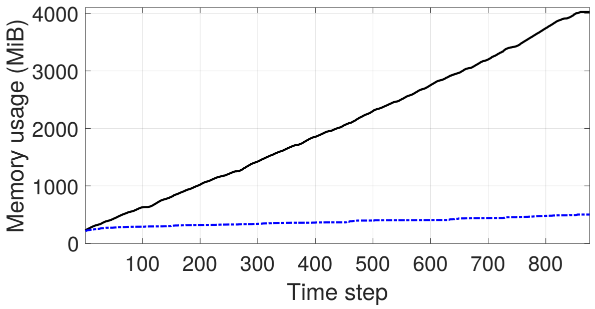


Fig. 7: Dynamic (heap) memory use during the LiDAR mapping process for KITTI-00, without (solid) and with (dashed) the automatic swap-out mechanism featured by the `WorldModel` module.

References

[1] Sameer Agarwal, Keir Mierle, et al. Ceres solver. 2012. URL http://ceres-solver.org/.

[2] Sean Anderson, Kirk MacTavish, and Timothy D Barfoot. Relative continuous-time slam. *The International Journal of Robotics Research*, 34(12):1453–1479, 2015. URL http://journals.sagepub.com/doi/abs/10.1177/0278364915589642.

[3] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[4] José-Luis Blanco, Javier González-Jiménez, and Juan-Antonio Fernández-Madrigal. Sparser relative bundle adjustment (srba): constant-time maintenance and local optimization of arbitrarily large maps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 70–77. IEEE, 2013. URL https://ieeexplore.ieee.org/abstract/document/6630558/.

[5] José-Luis Blanco-Claraco. OLAE-ICP: Robust and fast alignment of geometric features with the optimal linear attitude estimator. *arXiv preprint*, 2019.

[6] 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*, 35(10):1157–1163, 2016.

[7] 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 Transactions on Robotics*, 32(6):1309–1332, 2016.

[8] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert. Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4290–4297. IEEE, 2014.

[9] Laura A Clemente, Andrew J Davison, Ian D Reid, José Neira, and Juan D Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems*, number 2, 2007. URL http://www.roboticsproceedings.org/rss03/p38.pdf.

[10] Andrew I Comport, Ezio Malis, and Patrick Rives. Real-time quadrifocal visual odometry. *The International Journal of Robotics Research*, 29(2-3):245–266, 2010.

[11] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 29(6):1052–1067, 2007.

[12] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[13] Gamini Dissanayake, Shoudong Huang, Zhan Wang, and Ravindra Ranasinghe. A review of recent developments in simultaneous localization and mapping. In *Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on*, pages 477–482. IEEE, 2011.

[14] Renaud Dubé, Andrei Cramariuc, Daniel Dugas, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmap: 3d segment mapping using data-driven descriptors. *arXiv preprint arXiv:1804.09557*, 2018.

[15] Ethan Eade and Tom Drummond. Monocular slam as a graph of coalesced observations. In *IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.6056&rep=rep1&type=pdf.

[16] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22. IEEE, 2014.

[17] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

[18] Cipriano Galindo, Alessandro Saffiotti, Silvia Coradeschi, Pär Buschka, Juan-Antonio Fernandez-Madrigal, and Javier González. Multi-hierarchical semantic maps for mobile robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2278–2283. IEEE, 2005.

[19] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361. IEEE, 2012.

[21] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010. URL https://ieeexplore.ieee.org/abstract/document/5681215/.

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

[23] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *Josa a*, 4(4):629–642, 1987.

[24] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. Complex urban dataset with multi-level sensors from highly diverse urban environments. *The International Journal of Robotics Research*, 38:642–657, 2019.

[25] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[26] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[27] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613. IEEE, 2011. URL https://ieeexplore.ieee.org/abstract/document/5979949/.

[28] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3): 314–334, 2015. URL https://journals.sagepub.com/doi/abs/10.1177/0278364914554813.

[29] JM Martínez Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. In *Robotics: Science and Systems*. Robotics: Science and Systems, 2006. URL http://roboticsproceedings.org/rss02/p11.pdf.

[30] Daniele Mortari, F Landis Markley, and Puneet Singla. Optimal linear attitude estimator. *Journal of Guidance, Control, and Dynamics*, 30(6):1619–1627, 2007.

[31] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. URL https://ieeexplore.ieee.org/iel7/8860/4359257/07946260.pdf.

[32] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. URL https://ieeexplore.ieee.org/iel7/8860/4359257/07219438.pdf.

[33] Davide Scaramuzza, Friedrich Fraundorfer, Marc Pollefeys, and Roland Siegwart. Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints. In *IEEE 12th International Conference on Computer Vision*, pages 1413–1419. IEEE, 2009. URL https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/19362/eth-7807-01.pdf.

[34] Thomas Schneider, Marcin Dymczyk, Marius Fehr, Kevin Egger, Simon Lynen, Igor Gilitschenski, and Roland Siegwart. maplab: An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters*, 3(3):1418–1425, 2018. URL https://arxiv.org/pdf/1711.10250.

[35] Dieter Sibley, Christopher Mei, Ian D Reid, and Paul Newman. Adaptive relative bundle adjustment. In *Robotics: science and systems*, volume 32, page 33, 2009. URL http://www.roboticsproceedings.org/rss05/p23.pdf.

[36] Hauke Strasdat, J Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, 2, 2010. URL http://roboticsproceedings.org/rss06/p10.pdf.

[37] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2657–2664. IEEE, 2010.

[38] Henning Tjaden, Ulrich Schwanecke, Elmar Schömer, and Daniel Cremers. A region-based gauss-newton approach to real-time monocular multiple object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[39] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustmenta modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.230.4821&rep=rep1&type=pdf.

[40] Aisha Walcott-Bryant, Michael Kaess, Hordur Johannsson, and John J Leonard. Dynamic pose graph slam: Long-term mapping in low dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1871–1878. IEEE, 2012. URL https://ieeexplore.ieee.org/abstract/document/6385561/.

[41] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9, 2014.

[42] Zichao Zhang, Guillermo Gallego, and Davide Scaramuzza. On the comparison of gauge freedom handling in optimization-based visual-inertial state estimation. *IEEE Robotics and Automation Letters*, 3(3):2710–2717, 2018.