Monocular-Based SLAM for Mobile Robots: Filtering-Optimization Hybrid Approach

Monocular-based SLAM for Mobile Robots: Filtering-Optimization Hybrid Approach

Rodrigo Munguia[1*], Juan-Carlos Trujillo[1†], Guillermo Obregón-Pulido[2'†], Carlos I. Aldana[1'†]

[1*]Department of Computer Science, CUCEI University of Guadalajara, Guadalajara, 44430, Jalisco, Mexico.
[2]Department of Electro-photonics, CUCEI University of Guadalajara, Guadalajara, 44430, Jalisco, Mexico.

*Corresponding author(s). E-mail(s): rodrigo.munguia@academicos.udg.mx;
†These authors contributed equally to this work.

Abstract

The SLAM or Simultaneous Localization and Mapping still remains one of the most important problems to be fully addressed in the path to building fully autonomous mobile robots. This work aims to contribute to the above objective by presenting a novel hybrid architecture for implementing monocular-based SLAM systems for mobile robots. The key idea is to take advantage in a complementary manner of the inherent properties of the two main methodologies available today for implementing visual-based SLAM systems: the filter-based methods, and the optimization-based methods. The proposed method differs from previous approaches among other aspects because filter-based and optimization-based functionalities run concurrently in separate processes, improving the modularity and robustness of the system due to a degree of redundancy. Also, due to its modularity, the proposed approach can be easily applied to different mobile robot platforms using a monocular camera as its main sensor and different setups of additional sensors. In this sense, experiments with real data obtained from a differential robot and a quadrotor are presented to show that the proposed SLAM system can perform well in out-of-the-lab environments by using low-cost sensors. The experiments also show that the SLAM system exhibits real-time and stable computational performance. Moreover, a ROS-2 C++ open-source implementation of the proposed system is provided.

1

Keywords: SLAM, Monocular Vision, Mobile robots, Hybrid, Filters, Optimization

1 Introduction

In recent years, mobile robots have emerged as cutting-edge technology with important implications in industry, research, and everyday life. These versatile and autonomous machines have witnessed significant advancements, demonstrating their potential to address diverse challenges and fulfill different tasks across various domains, from industrial automation and logistics [1] to healthcare assistance [2] [3] and exploration of hazardous environments [4], mobile robots have proven to be indispensable tools for enhancing efficiency, safety, and convenience.

The remarkable progress in mobile robotics is a result of continuous innovation in a multitude of fields, such as artificial intelligence and machine learning [5] [6], computer vision [7] [8], sensor technology [9], control theory [10], and navigation technologies [11] [12]. This has paved the way for the development of intelligent, adaptable, and reliable robots that can navigate complex environments, interact with humans, and execute intricate tasks with unparalleled precision. One relevant class of mobile robotic platforms is represented by Unmanned Aerial Vehicles (UAVs), commonly known as drones. Their importance lies in their diverse applications across various sectors, including agriculture, surveillance, disaster response, and logistics [13]. Also, several advances in different fields such as control and path planning of UAVs have been achieved in recent years (see [14], [15]).

In the above context, the Simultaneous Localization and Mapping problem, often referred to as SLAM, is a fundamental challenge in robotics and autonomous systems. It involves a mobile robot navigating an unknown environment while simultaneously building a map of the environment and determining its own position relative to that map. In other words, SLAM enables a robot to explore and create a map of an environment in real-time while also localizing itself within that map, even in the absence of external positioning systems like GPS. Despite that important advances

in SLAM have been achieved in recent years [16], there are several technical challenges regarding robustness and accuracy, real-time performance, data association, loop closure and map consistency, sensor integration, and fusion, dynamic environments, and multi-robot SLAM, among others, that remain open problems for the research community.

This work is intended to contribute to the path to fully solving the SLAM problem by presenting a novel filtering and optimization hybrid architecture for monocular-based SLAM systems. The idea is to take advantage of the respective strengths of the filtering and optimization SLAM techniques, while at the same time minimizing their drawbacks. The filtering part of the proposed hybrid architecture allows the potential integration of different onboard sensors that make it possible to implement ad-hoc visual-based SLAM systems for different mobile robotic platforms in a straightforward manner. On the other hand, the optimization part of the proposed hybrid architecture allows to build and manage maps of several thousands of visual features, as well as minimizing the accumulated error drift by detecting and closing trajectory loops. Moreover, the proposed SLAM methodology exhibits a stable computation time that allows real-time performance in consumer-grade hardware. Also, C++ ROS 2 open-source code is provided for the interested reader.

2 Related works

Currently, there are two main general methodologies for addressing the visual SLAM (Simultaneous Localization and Mapping) problem: Filter-based and optimization-based methods. The first type makes use of stochastic filtering techniques as the Kalman Filters in all its variants: Linear Kalman Filter [17, 18], Extended Kalman Filter [19–23], Unscented Kalman Filter [24–26], and also other filtering techniques such as the Particle Filter [27–29] or the Information Filter [30, 31]. Different from the previous approaches, the second type of visual-based SLAM techniques formulate the localization and mapping problems as decoupled optimization problems. Examples of optimization-based SLAM techniques are KeyFrame-based SLAM and Graph SLAM, see [32–38].

The use of both approaches, the filter-based and optimization-based methods come with different benefits and disadvantages. For instance, it is possible to design theoretically well-founded filter-based SLAM systems, because important system properties, such as stability and convergence [39, 40], or observability [41–43] can be studied by using different well-established mathematical tools coming from the systems and control theory. Also, due to the stochastic nature of filter-based SLAM methods, it is not difficult to fuse additional data coming from aiding sensors, such as altimeters, inertial, range, and other kinds of sensors, to improve the accuracy, robustness, and flexibility of the systems. In this sense, optimization-based methods are generally more heuristically designed, having for instance as a consequence that fusing data from aiding sensors requires more ad-hoc solutions (e.g., [44]). It is also worth noticing the almost absence of math-proof-type studies in the literature that provide theoretical foundations for these kinds of systems.

On the other hand, one big advantage of optimization-based SLAM methods has to do with the map size that these kinds of approaches can handle. In this case, the computational requirements for small maps are in general higher for optimization-based systems than for those required for filter-based systems [45]; however, as the number of map features increases, the computational cost of filter-based systems typically scales exponentially, while the computational cost of optimization-based systems scales better thanks to the use of local-optimization strategies and their decoupled localization and mapping nature. Thus, filter-based SLAM methods are often limited to local SLAM strategies.

In the previous author's work [46], one first attempt at a general hybrid visual-based SLAM architecture that exploits the advantages of both methodologies was introduced. In this case, the general idea was to use a filter-based SLAM sub-system for performing a continuous local SLAM process, and the use of optimization-based SLAM techniques for maintaining a consistent global map of the environment. In this work, the previous general idea of using stochastic filters for performing local SLAM and optimization-based techniques for global mapping is taken a step further in order

to present a novel and refined filtering-optimization hybrid approach for implementing a monocular SLAM system that can be applied to a broad class of mobile robots.

Some of the most important differences that can be highlighted with respect to the previous work are:

i) The system architecture is modified to better integrate the global mapping process and the loop correction process into a single sub-system called global SLAM. Also, the inter-process communication model was changed from a *shared-memory* type to a *message-passing* type. The above is to improve the modularity and robustness of the system as well as to facilitate its implementation using open-source robotics middleware suites such as ROS [47].

ii) For the local-SLAM process, instead of using a minimal state vector composed of pose and linear velocity, in this work, a general state vector is defined for better representing the state of different mobile robotic platforms. Also, the level arm effect is now taken into account in the definition of the measurement sensors models. The above also allows the use of the proposed SLAM system with different kinds of mobile robot platforms in a straightforward manner.

iii) Instead of using Ceres Solver [48] as a back-end for solving the optimization problems, in this work, GTSAM and factor graphs [49] is used as the back-end for implementing different optimization task in the global-SLAM sub-system.

iv) Two study cases of monocular-based SLAM systems are presented: The first is for a quad-rotor using the following aiding measurements: i) range, 2) attitude, and 3) altitude. The second is for a differential mobile robot using only wheel encoder readings as aiding measurements.

In [50] a hybrid filter-based and graph-based approach to SLAM is presented. The basic idea of this method is to use a switching scheme between an Information Filter and an optimizer. In this case, the proposed architecture differs in a significant manner from the above approach because the filtering-based part and the optimization-based part run concurrently in separate processes, improving the modularity, and also potentially improving robustness due to some degree of redundancy, meaning that the filter-based local SLAM can actually run independently from the

optimization global SLAM. In [51] a hybrid occupancy grid inertial (OGI)-SLAM and key-point optimization approach for indoor pedestrian localization with low-cost inertial measurement units (IMUs) is presented. The idea is to maintain the short and medium-term localization accuracy with OGI-SLAM and achieve long-term localization accuracy improvement with keypoint maps. In this case, different from the architecture proposed in this work, which is intended to be a monocular-camera-based general navigation solution to different mobile robotic platforms using diverse aiding sensors, the method described in [51] represents an ad-hoc IMU-based solution for indoor pedestrian localization. Another important difference is the use of sparse visual feature maps in the proposed method instead of occupancy grids as the referenced approach.

3 Materials and Methods

3.1 Architecture and system specification

In this section, the robot state vector to be estimated is described, then the sensor measurements used as system input are presented, and finally, the proposed system architecture is described from a higher-level perspective.

3.1.1 System state vector

Given a series of visual measurements $\mathbf{z}_{uv}$ as well as a series of aiding measurements $\mathbf{z}_k$ (both to be defined later), the objective of the proposed monocular-based SLAM system is to estimate the following robot state vector $\mathbf{x}_r$:

$$\mathbf{x}_r = \begin{bmatrix} \mathbf{e}^{NR} & \boldsymbol{\omega}^R & \mathbf{r}^N & \mathbf{v}^R \end{bmatrix}^T \tag{1}$$

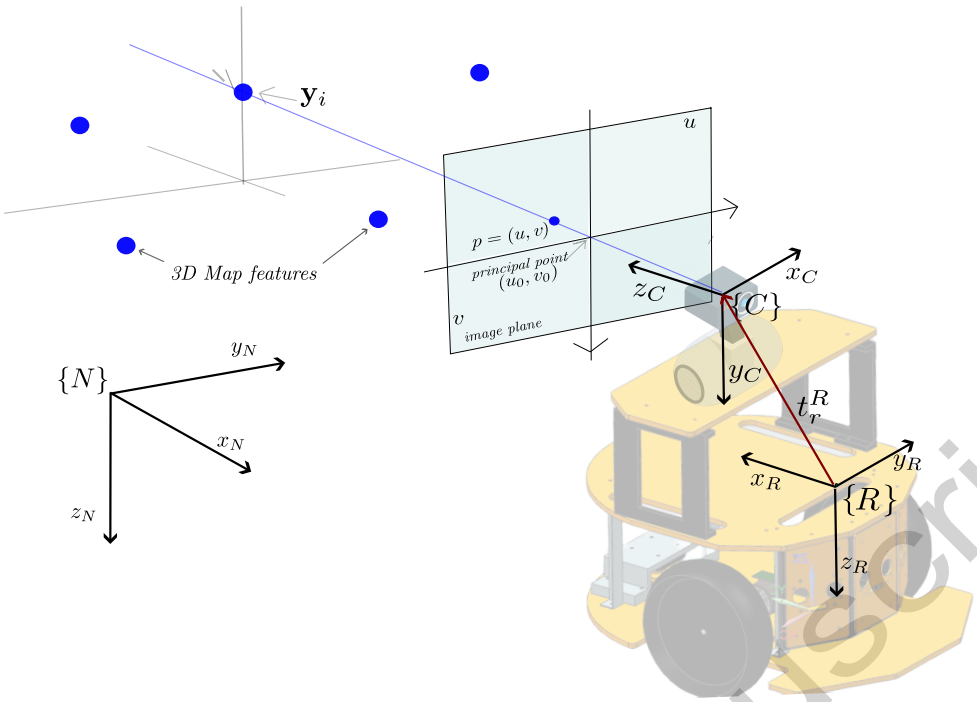

Fig. 1 System parameterization.

Where the vector $\mathbf{e}^{NR} = [\theta, \phi, \psi]$ represents the orientation of the robot coordinate frame ^{R} with respect to the navigation coordinate frame ^{N}.The angles θ, ϕ, ψ are the Tait-Bryan angles (commonly referred to as Euler angles). The angles θ, ϕ, ψ are also known as *roll*, *pitch* and *yaw*. In this work, the intrinsic rotation sequence $\psi - \phi - \theta$ is used. The vector $\boldsymbol{\omega}^R = [\omega_x, \omega_y, \omega_z]$ represents the velocity rotation of the robot, along the axes x_R, y_R, z_R, defined in the robot frame ^{R}. The vector $\mathbf{r}^N = [x_r, y_r, z_r]$ represents the position of the robot expressed in the navigation frame ^{N}. The vector $\mathbf{v}^R = [v_x, v_y, v_z]$ represents the linear velocity of the robot, along the axes x_R, y_R, z_R, expressed in the robot frame ^{R}.

Since the proposed system is intended for robot autonomous navigation local tasks, the local tangent frame is used as the navigation frame. The navigation coordinate frame ^{N} follows the NED convention (north-east-down), and the initial position of the robot defines its origin. In this work, all the coordinate systems are right-handed, and

the magnitudes expressed in the i)navigation frame, ii) robot frame, and iii)camera frame are denoted respectively by superscripts ^{N}, ^{R}, and ^{C} (See Figure 1).

3.1.2 Visual measurements

For estimating the robot state vector defined in Equation 1, it is assumed that the robot is equipped with a standard monocular camera. The position of a i-th (static) 3D point that belongs to the robot's environment, expressed in the navigation frame ^{N}, is defined by the vector $\mathbf{y}_i^N$ (See Figure 1):

$$\mathbf{y}_i^N = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T \tag{2}$$

The projective transformation that maps 3D points $\mathbf{y}_i^N = [x_i, y_i, z_i]$ in navigation coordinates ^{N} into 2D points $\mathbf{z}_{uv} = [u_i, v_i]$ in the image plane is defined by:

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \mathbf{f}_d(u', v', k_1, \ldots, k_n) \quad \text{with} \quad s \begin{bmatrix} u_i' \\ v_i' \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_c \\ r_{21} & r_{22} & r_{33} & y_c \\ r_{31} & r_{32} & r_{33} & z_c \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{3}$$

Where, the camera distortion model $\mathbf{f}_d$ depends on the undistorded point coordinates $[u_i', v_i']$ and the distortion parameters $k_1, ..., k_n$. The focal length and principal point of the camera are respectively f and $[u_0, v_0]$. Both, the distortion parameters and the intrinsic parameters of the camera are assumed to be known by calibration. In this work, the distortion model [52] is used. Also, in equation 3, the matrix $\mathbf{R}^{NC} = (\mathbf{R}^{RC}\mathbf{R}^{NR})$, is a rotation matrix with i-j elements r_{ij}, transforming from the navigation frame ^{N} to the camera frame ^{C}. $\mathbf{R}^{NR}(\theta, \phi, \psi)$ is the rotation matrix transforming from the navigation frame ^{N} to the robot frame ^{R} and it is computed from the angles of the system state vector. $\mathbf{R}^{RC}(\theta, \phi, \psi)$ is the robot frame ^{R} to camera frame ^{C} rotation matrix and it is assumed to be known. The position of the camera, expressed in the navigation frame ^{N}, is defined by vector $\mathbf{c}^N = [x_c, y_c, z_c]^T$, with $\mathbf{c}^N = \mathbf{r}^N + \mathbf{R}^{RN}\mathbf{t}_c^R$, and $\mathbf{R}^{RN} = (\mathbf{R}^{NR})^T$. The vector $\mathbf{t}_c^R$ represents the position of the camera expressed in the robot frame ^{R}, and it is assumed to be known.

3.1.3 Aiding measurements

The visual-based navigation system proposed in this work is intended to be applied to different types of mobile robots. Each mobile robot platform is equipped with different types of sensors, and it is important to take advantage of these additional sensory sources to improve the performance of the system. We will simply denote $\mathbf{z}_{\{s\}}$ to the measurement of the $\{s\}$ adding sensor. The presentation of all the models of the sensors that can be potentially used as aiding sensors in the proposed system is outside of the scope of this work. On the other hand, later in Section 3.2 the description of some common aiding sensors will be covered when two different robotic system implementation cases are introduced.

3.1.4 System architecture

The architecture of the proposed monocular-SLAM system is composed of two processes running in a concurrent manner: i) a filter-based local SLAM process, and ii) an optimization-based global SLAM process (See the figure 2).

The filter-based local SLAM process makes use of stochastic-filter techniques (e.g. EKF, UKF, EIF, etc) for building and maintaining a local map of the robot's environment, while at the same time, estimating the robot's state by fusing visual information obtained from a monocular (standard) camera as well as the information obtained from additional aiding sensors, such as inertial measurement units, altimeters, range sensors or others. The local SLAM process can operate in a completely independent manner from the global SLAM process to produce estimates of the robot's state (6-DOF pose, velocities, etc.) and the 3D position of the visual features observed by the camera. However as was already mentioned, the computational cost of filter-based SLAM methods typically scales poorly as the number of visual features increases in the system state. Therefore, to maintain a stable computational cost, old features are removed from the system state. The above imposes a constraint on the map size that a filter-based SLAM process can handle by itself, thus limiting its closing loop capabilities. Meaning that, if the robot returns to a previously visited area where the visual features have been removed, then the accumulated error in pose can not be minimized.

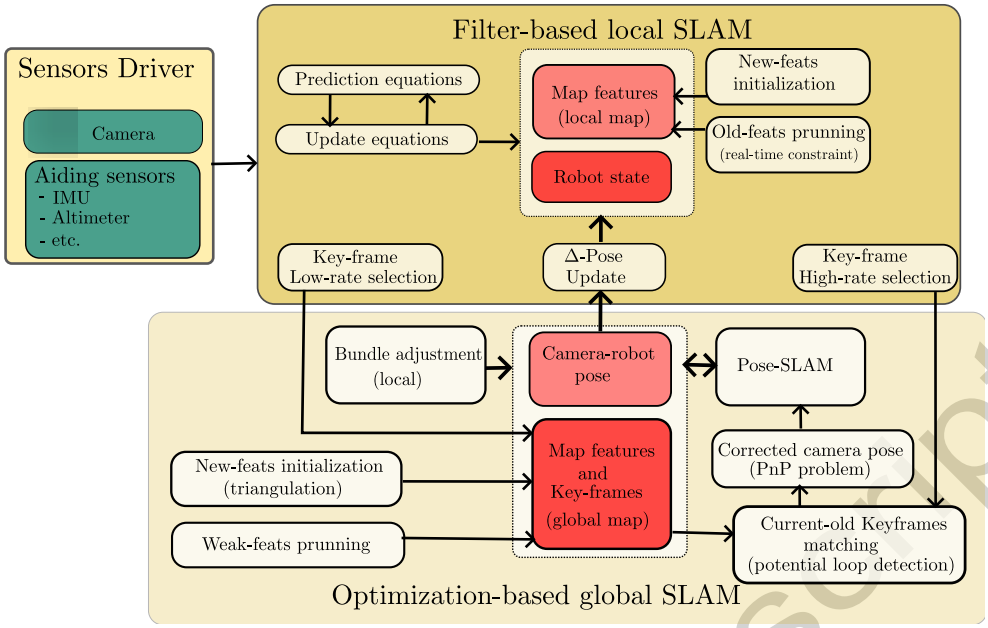

Fig. 2 Proposed architecture.

In the proposed system architecture, the global SLAM process makes use of (keyframe) optimization-based techniques for building and maintaining a global and persistent map of the robot's environment, as well as computing camera-robot pose corrections for the local SLAM process. To accomplish the above task, the global SLAM process takes as inputs a stream of keyframes generated by the local SLAM process. Each key frame contains its camera-pose estimate and it is generally generated at a much lower rate than the camera input fps. Meaning that the operation rate of the global SLAM process is decoupled from the operation rate of the local SLAM process. While the real-time performance of the local SLAM is constrained by the frequency of the raw sensory input, the global SLAM operates decoupled from the input sensors to compute asynchronously camera-robot pose updates and build the global and persistent map.

Internally, the global SLAM process implements two tasks running concurrently: i) Map-building, and ii) Loop-closing. When new keyframes are available, the first of the above tasks incrementally builds the global map by computing the 3D pose of new

map visual features by using a triangulation technique between the new and previously mapped keyframes. Also for the map-building task, a local bundle-adjustment technique is applied for the refinement of the global map. The maximum number of keyframes to be optimized by the local bundle adjustment depends on the computational power available, but in many cases, a moderate size is enough to enable the local bundle adjustment to mitigate error drift and even to close loops of medium size (bigger than the ones handled by the local SLAM process). Camera-robot position correction/updates computed by the local bundle adjustment are sent to the local SLAM process. Since the computational cost of the bundle adjustment still considerably increases with the number of keyframes to be optimized, the loop-closing task is implemented to provide the system with bigger loop-closing capabilities. In this case, every time new high-rate keyframes are available, this task looks for visual matches against the old keyframes composing the global map by using visual descriptors. In this work, ORB descriptors are used, but other kinds can be used instead. When a visual relation between an old and a new keyframe is established (loop detection) a PnP (Perspective-n-Point) technique is used for computing the "corrected" current camera pose with respect to the old keyframe. Then, the corrected camera pose is used as a closing loop measurement in a Graph-SLAM method. The graph is composed only of the keyframe poses of the global map but not the poses of the visual features, thus allowing the optimization of large graphs (camera poses) in an efficient manner. When a loop is successfully closed, a camera-robot pose correction/update is sent to the local SLAM process.

Also, observe in Figure 2 that a sensors driver module is defined to asynchronously manage the reading and communication of sensor data to the SLAM system. In this architecture, the arrows connecting the sensor driver, Local SLAM, and Global SLAM blocks represent inter-process communication through messages. In subsequent sections, the subsystems and related methods composing the system architecture will be presented.

11

3.2 Local SLAM

In this work, an Extended Kalman Filter (EKF) was used for implementing the local SLAM process but any other stochastic filter technique could be used instead. The implementation of the local SLAM process is defined by the typical loop of predictions and updates of the standard EKF-SLAM (See [53, 54]), along with some additional functionalities required by the proposed system architecture (See Figure 2). The specific implementation of the local SLAM process will depend on the robotic platform to be used. In this work, two implementation cases will be presented: i) A simple differential mobile robot equipped with wheel encoders, and ii) A drone equipped with an AHRS (Attitude and Heading Reference System), and Altimeter. Both platforms are equipped with standard monocular cameras.

To implement the local SLAM the robot state vector $\mathbf{x}_r$ defined in Eq. 1 is augmented with the states of n map features $\mathbf{y}_i$, $i \in \{1, .., n\}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_r^T & \mathbf{y}_1^T & \dots & \mathbf{y}_n^T \end{bmatrix}^T \tag{4}$$

In the SLAM literature, the system state augmentation is also called *feature initialization* and it is explained in section 3.2.3.

3.2.1 Prediction step

The estimate $\hat{\mathbf{x}} = \mathbb{E}\{\mathbf{x}\}$ of the state vector $\mathbf{x}$, along with an estimate of its covariance matrix $\mathbf{P} = cov(\mathbf{x}, \mathbf{x}) = \mathbb{E}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\}$, is propagated forward in time using the EKF prediction equations:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}^- &= \mathbf{f}\left(\hat{\mathbf{x}}^{\pm}, \mathbf{u}\right) \\ \dot{\mathbf{P}}^- &= \mathbf{A}\mathbf{P}^{\pm} + \mathbf{P}^{\pm}\mathbf{A}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \end{aligned} \tag{5}$$

where $\mathbf{f}\left(\hat{\mathbf{x}}^{\pm}, \mathbf{u}\right)$ is the dynamic/kinematic model of the robot, $\mathbf{u}$ is the input vector, $\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\hat{\mathbf{x}}^{\pm}, \mathbf{u})$ and $\mathbf{G} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\hat{\mathbf{x}}^{\pm}, \mathbf{u})$, and let $\mathbf{n}_k \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{Q})$ be the noise vector that affect the state. Also, in EKF equations the following notation is used: $(^-)$ for *a priori* values, $(^+)$ for *a posteriori* values, and $(^{\pm})$ for "last available value". In practice, Eq. 5 can be implemented using simple Euler integration.

Differential robot	**Quadrotor**
$\begin{cases} \mathbf{e}^{NR}_{k+1} = \mathbf{e}^{NR}_{k} + \mathbf{R}^{\omega e}\boldsymbol{\omega}^{R}_{k}\Delta t \\ \boldsymbol{\omega}^{R}_{k+1} = \begin{bmatrix} 0 & 0 & \omega_z \end{bmatrix}^T \\ \mathbf{r}^{N}_{k+1} = \mathbf{r}^{N}_{k} + \mathbf{R}^{RN}\mathbf{v}^{N}_{k}\Delta t \\ \mathbf{v}^{R}_{k+1} = \begin{bmatrix} v_x & 0 & 0 \end{bmatrix}^T \\ \mathbf{y}_{1[k+1]} = \mathbf{y}_{1[k]} \\ \vdots \\ \mathbf{y}_{n[k+1]} = \mathbf{y}_{n[k]} \end{cases}$ with: $v_x = \frac{r}{2}(\omega_r + \omega_l)$ $\omega_z = \frac{r}{L}(\omega_l - \omega_r)$	$\begin{cases} \mathbf{e}^{NR}_{k+1} = \mathbf{e}^{NR}_{k} + \mathbf{R}^{\omega e}\boldsymbol{\omega}^{R}_{k}\Delta t \\ \boldsymbol{\omega}^{R}_{k+1} = \boldsymbol{\omega}^{R}_{k}\,(1 - k_\omega \Delta t) \\ \mathbf{r}^{N}_{k+1} = \mathbf{r}^{N}_{k} + \mathbf{R}^{RN}\mathbf{v}^{N}_{k}\Delta t \\ \mathbf{v}^{R}_{k+1} = \mathbf{v}^{R}_{k}\,(1 - k_v \Delta t) \\ \mathbf{y}_{1[k+1]} = \mathbf{y}_{1[k]} \\ \vdots \\ \mathbf{y}_{n[k+1]} = \mathbf{y}_{n[k]} \end{cases}$

Table 1 Local SLAM: Differential robot and Quadrotor kinematic models.

Table 1 shows the kinematic models for both, the differential robot and the quadrotor. In both cases, the matrix $\mathbf{R}^{\omega e}$ establish the relationship between angular positions θ, ϕ, and ψ and the angular rates ω_x, ω_y, and ω_z:

$$\mathbf{R}^{\omega e} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/cos\theta & \cos phi/\cos\theta \end{bmatrix} \tag{6}$$

For the differential robot, it is assumed that the input $\mathbf{u} = [\omega_l, \omega_r]^T$ is available through the wheel encoders, where ω_l and ω_r are the angular velocities of the left and right wheel. Also, let r be the radius of the wheels, and L be the length of the wheel's axis. Uncertainties of the encoders and other unstructured uncertainties, like wheel slippages, are integrated into the system by means of the covariance matrix $\mathbf{Q} = \sigma^2_{\omega_{lr}}\mathbf{I}_{2\times 2}$ through parameter $\sigma_{\omega_{lr}}$.

For the quadrotor, an unconstrained constant acceleration model is used. In this case $\mathbf{Q} = \text{diag}(\Omega^R, \mathbf{V}^R)$, where $\Omega^R = \sigma^2_\omega \Delta t$ and $V^R = \sigma^2_v \Delta t$ respectively represent an unknown angular and linear velocity impulse with acceleration zero-mean and known-covariance Gaussian processes σ_ω and σ_v. Also, let k_ω and k_v be correlation time factors, which respectively model how fast the angular and linear velocities become zero in the absence of sensor measurements.

3.2.2 Update step

Every time a measurement $\mathbf{z}_{\{s\}_k}$ is available at step k, from a sensor $\{s\}$ with measurement model $\mathbf{h}_{\{s\}}(\hat{\mathbf{x}}_k^-)$, the estimates $\hat{\mathbf{x}}$ and $\mathbf{P}$ are updated through the EKF update equations:

$$\begin{aligned} \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}_{\{s\}}(\hat{\mathbf{x}}_k^-) \\ \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{C}_{\{s\}_k})\mathbf{P}_k^- \\ \mathbf{K}_k &= \mathbf{P}_k^-\mathbf{C}_{\{s\}_k}^T(\mathbf{C}_{\{s\}_k}\mathbf{P}_k^-\mathbf{C}_{\{s\}_k}^T + \mathbf{R}_{\{s\}})^{-1} \end{aligned} \tag{7}$$

where $\mathbf{K}$ is the Kalman gain and $\mathbf{C}_{\{s\}_k} = \frac{\partial \mathbf{h}_{\{s\}}}{\partial \mathbf{x}}(\hat{\mathbf{x}}_k^-)$. Also, let $\mathbf{R}_{\{s\}}$ be the measurement noise covariance matrix of the $\{s\}$ sensor.

Differential robot	**Quadrotor**
$\mathbf{h}_{uv}(\mathbf{x}) = \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{f}_d(u', v', k_1, \ldots, k_n)$	$\mathbf{h}_{uv}(\mathbf{x}) = \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{f}_d(u', v', k_1, \ldots, k_n)$ $\mathbf{h}_e(\mathbf{x}) = \begin{bmatrix} \theta & \phi & \psi \end{bmatrix}^T$ $\mathbf{h}_{z_r}(\mathbf{x}) = z_r$

Table 2 Local SLAM: Differential robot and Quadrotor measurement models.

Table 2 shows the measurement models used for both, the differential robot and the quadrotor. In both cases, the same visual measurement model $\mathbf{h}_{uv}(\mathbf{x})$ described in section 3.1.2 is used. Note that in addition to the visual measurements, the quadrotor SLAM system makes use of the following additional aiding sensors through their respective measurement models: i) $\mathbf{h}_e(\mathbf{x})$ models attitude measurements $\mathbf{z}_e = [\theta, \phi, \psi]^T$ obtained from the AHRS, and ii) $\mathbf{h}_{z_r}(\mathbf{x})$ models altitude measurements $\mathbf{z}_{z_r} = z_r$ obtained from the altimeter.

3.2.3 Features initialization

For initializing new map features $\mathbf{y}_{new}^N = [x_i, y_i, z_i]^T$ into the system state $\mathbf{x}$, an initialization function $\mathbf{y}_{new}^N = \mathbf{g}(\mathbf{x}, z_1, ..., z_n)$ must be defined, where z_i is the i-est measurement used in $\mathbf{g}(.)$ for computing the pose of $\mathbf{y}_{new}^N$. When $\mathbf{y}_{new}^N$ is computed,

then the system estate and covariance matrix is augmented as follows:

$$\mathbf{x}_{new} = \begin{bmatrix} \mathbf{x}_{old}^T \ ; \ (\mathbf{y}_{new}^N)^T \end{bmatrix}^T$$
$$\mathbf{P}_{new} = \mathbf{J} \begin{bmatrix} \mathbf{P}_{old} & 0 \\ 0 & \mathbf{R}_g \end{bmatrix} \mathbf{J}^T \tag{8}$$

where $\mathbf{J} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}, z_1, ..., z_n}(\mathbf{x}, z_1, ..., z_n)$, and let $\mathbf{R}_g$ be the measurement noise covariance for measurements $z_1, ..., z_n$.

In the case of the differential robot, the delayed initialization technique described in [55] is used, which makes use of monocular measurements uniquely. In the case of the quadrotor, the undelayed initialization technique described in [56] is used, which makes use of monocular measurements and range measurements obtained from an ultrasonic sensor.

3.2.4 Additional functionalities

The following functionalities are implemented to adapt the EKF-SLAM method described above to the requirements of the local SLAM process defined by the system architecture (See the section 3.1.4):

- **Old-features pruning.** To maintain stable computational cost weak and old visual features are removed from the system state. The criterion for defining a "weak" feature is a high ratio between the number of times that a feature, expected to be observed by the camera, is matched or not. An "old" feature is defined as a feature with a high number of times that it has not been observed by the camera.
- **Keyframe selection.** A keyframe $\mathcal{K}$, is a frame selected regularly from the camera input video stream according to some particular criteria (eg., visual [33], spatial [32]). The jth keyframe $\mathcal{K}_j$ also stores the robot state $\mathbf{x}_{r_j}$ (and therefore also the camera state $\mathbf{c}_j$) corresponding to the moment when the keyframe was captured. Two kinds of Keyframes are sent to the global SLAM process: i) low-rate keyframes, and ii) high-rate keyframes. The first ones are selected if the criteria $(\|\mathbf{r}_k^N - \mathbf{r}_{k-1}^N\|)/(\overline{\mathbf{y}_i - \mathbf{c}^N}) > \lambda$ is met, meaning that a keyframe is selected if the ratio between the robot displacement and the average depth's features are bigger than

some threshold. Low-rate keyframes are used in the map-building task (see section 3.3.1). High-rate keyframes are used in the loop-closing task (see section 3.4) and are selected at a constant rate similar, to or slightly slower than the input camera frame rate.

- **Δ-Pose update.** As was already explained, since old features are removed from the system state, larger trajectory loops can not be closed by the local SLAM and therefore the accumulated drift in the estimated pose of the robot and the map features can not be minimized. A Δ-Pose $\Delta\mathbf{p} = [\Delta x, \Delta y, \Delta z]^T$ is a pose correction computed by the global SLAM process to minimize the error drift of the poses computed by the local SLAM process. Every time a Δ-Pose is available, the pose of the robot and (local) map features are simply updated by: $\mathbf{r}^N := \mathbf{r}^N + \Delta\mathbf{p}$ and $\mathbf{y}_i := \mathbf{y}_i + \Delta\mathbf{p}$.

3.3 Global SLAM

Different from the local SLAM process which implementation takes into account the particular characteristics of the robot platform (e.g. sensors), the global SLAM process implementation is independent of the robot platform, tanking solely as input the set of n keyframes $\boldsymbol{\mathcal{K}} = \{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_n\}$ computed by the local SLAM process. The global SLAM implements two concurrent tasks: i) map-building and, ii) loop-closing. Both tasks can compute correction poses $\Delta\mathbf{p}$ which are sent to the local SLAM process.

3.3.1 Map-building

The map-building task has as its objective to construct and maintain a persistent global map of the robot environment. The global map $\boldsymbol{\mathcal{G}}$ is composed by a set $\boldsymbol{\mathcal{K}}_L = \{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_n\}$ of n low-rate keyframes (subscripts $_1$ and $_n$indicate respectively the oldest and latest keyframe), and a set $\boldsymbol{\mathcal{A}} = \{\mathbf{a}_1^N, \mathbf{a}_2^N, \ldots, \mathbf{a}_m^N\}$ of m map features, where $\mathbf{a}_i^N = [x_i, y_i, z_i]^T$ is the 3D pose of the i-est global-map feature. It is important to differentiate between the local map features $\mathbf{y}_i$ used in the local SLAM process, and the global map features $\mathbf{a}_i$ used in the global SLAM process. To better clarify the above distinction, the latter ones will be referred to as *anchors*.

The map-building task consists of two steps: i) anchors initialization and pruning, and ii) local bundle adjustment, which is executed every time a new low-rate keyframe $\mathcal{K}_j$ is available.

i) Anchors initialization and pruning. To add new anchors $\mathbf{a}^N_{new} = [x_i, y_i, z_i]$ to the global map $\boldsymbol{\mathcal{G}}$, a triangulation technique is used. To triangulate the 3D position of a new map feature $\mathbf{a}^N_{new}$, it is assumed that it can be observed (and matched) in two different consecutive keyframes $\mathcal{K}_j$.

From model (3) we have that:

$$\begin{bmatrix} su' & sv' & s \end{bmatrix}^T = \mathbf{K} \begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}^T \quad \text{with} \quad \mathbf{K} = \mathbf{P}_c \begin{bmatrix} \mathbf{R}^{NC} | \mathbf{c}^N \end{bmatrix}$$

where $\mathbf{P}_c$ is the camera intrinsic matrix, and $\mathbf{a}'_i = [x_i, y_i, z_i, 1]^T$ is defined as the pose of a 3D point to be initialized and expressed in homogeneous coordinates. Also, let $\mathbf{K}^j$ be the j-projection matrix computed from the j-robot pose $\mathbf{x}_{r_j}$ associated to the j-keyframe, and let $[u'_j, v'_j]$ be the undistorted projection of the 3D point $[x_i, y_i, z_i]$ obtained from $\mathbf{K}^j$. Finally, let $\mathbf{k}^j_i$ be the i row of the projection matrix $\mathbf{K}^j$.

Now, we have that $su' = \mathbf{k}^j_1 \mathbf{a}'_i$, $sv' = \mathbf{k}^j_2 \mathbf{a}'_i$, and $s = \mathbf{k}^j_3 \mathbf{a}'_i = z_c$. Substituting $s = z_c$, we have that: $z_c u' = \mathbf{k}^j_1 \mathbf{a}'_i$ and $z_c v' = \mathbf{k}^j_2 \mathbf{a}'_i$. Considering two projections $[u'_j, v'_j]$ and $[u'_{j-1}, v'_{j-1}]$, of the same 3D point $\mathbf{a}_i$, the following linear system can be formed:

$$\begin{cases} p_{z_i} u'_j = \mathbf{k}^j_1 \mathbf{a}'_i \\ p_{z_i} v'_j = \mathbf{k}^j_2 \mathbf{a}'_i \\ p_{z_i} u'_{j-1} = \mathbf{k}^{j-1}_1 \mathbf{a}'_i \\ p_{z_i} v'_{j-1} = \mathbf{k}^{j-1}_2 \mathbf{a}'_i \end{cases} \tag{9}$$

Note that the above linear system can be augmented and solved for n 3D points $\mathbf{a}'_i$, $i \in \{1, .., n\}$.

Every visual match of an anchor $\mathbf{a}^N$ in a keyframe $\mathcal{K}_j$ introduces a spatial constraint to the minimization problem (i.e. bundle adjustment), thus contributing more to minimizing the estimation errors. Anchors that are not observed and matched in at least n additional keyframes from those used to triangulate it, where $n \geq 1$, are considered "weak anchors" and are removed from the global map $\boldsymbol{\mathcal{G}}$.

17

ii) Bundle adjustment. Bundle Adjustment (e.g., [57]) is a technique typically used to refine the estimates of camera intrinsic and extrinsic parameters (e.g., focal length, position, orientation), and the 3D positions of points in the scene (e.g., feature points, landmarks). The goal is to minimize the difference between the predicted positions of the 2D features (as projected by the cameras) and their observed positions in the images. In this work, a local bundle adjustment method is used to refine the global map $\boldsymbol{\mathcal{G}}$.

First let define $\boldsymbol{\mathcal{K}}_B \subseteq \boldsymbol{\mathcal{K}}_L = \{\mathcal{K}_n, \mathcal{K}_{n-1}, ..., \mathcal{K}_{n-a}, \mathcal{K}_{n-b}, ..., \mathcal{K}_{n-c}\}$, where $a < b < c$, as the subset of m keyframes to be included in the local bundle adjustment. Note that the subset $\boldsymbol{\mathcal{K}}_B$ is composed of the $a+1$ latest keyframes, and also the $c-b+1$ oldest visually-linked keyframes. Two keyframes $\mathcal{K}_j$ are visually linked if at least one anchor $\mathbf{a}_i^N$ can be projected and matched in both keyframes. In this case, $\mathcal{K}_{n-c}$ is the visually linked oldest keyframe with respect to $\mathcal{K}_n$. Visual constraints with old keyframes are introduced to allow the bundle adjustment process to implicitly close trajectory loops. To manage all the visual links between keyframes a visibility graph is implemented as it is described in [46].

Also let $\mathbf{v}_{ij}$ be the measured projection of the ith anchor on the jth keyframe, and let $\mathbf{h}_a(\mathbf{c}_j^N, \mathbf{a}_i^N)$ be the predicted projection of the ith anchor on the jth keyframe, where $\mathbf{c}_j^N$ is the camera pose associated with the jth keyframe.

The bundle adjustment minimizes the total reprojection error with respect to n anchors belonging to $\boldsymbol{\mathcal{A}}$ and the m camera pose $\mathbf{c}_j^N$ associated with the m keyframes belonging to $\boldsymbol{\mathcal{K}}_B$, or

$$\min_{\mathbf{c}_j^N, \mathbf{a}_i^N} \sum_{i=1}^{n} \sum_{j=1}^{m} d(\mathbf{h}_a(\mathbf{c}_j^N, \mathbf{a}_i^N), \mathbf{v}_{ij})^2 \tag{10}$$

where $d(\mathbf{a}, \mathbf{b})$ denotes the Euclidean distance between the image points represented by vectors $\mathbf{a}$ and $\mathbf{b}$. If the anchor i is not visible in the keyframe j then $d(.,.) = 0$. In this work, the minimization problems are solved using GTSAM [49], which is a BSD-licensed C++ library that uses factor graphs and Bayes networks rather than sparse matrices to optimize for the most probable configuration.

If $\hat{\mathbf{c}}_j^N$ is the optimized $j-$camera pose computed from (10), then a pose update can be computed from $\Delta\mathbf{p} = \hat{\mathbf{c}}_n^N - \mathbf{c}_n^N$ after performing the local bundle adjustment, where the subscript $_n$ corresponds to the most recent camera pose.

3.4 Loop-closing

The loop-closing task has as its objective to detect and close large trajectory loops, therefore minimizing the accumulated error drift during those trajectories with respect to the original mapped areas. The loop-closing task consists of three steps: i) potential loop detection, ii) corrected camera pose computation, and iii) Pose-SLAM.

i) Potential loop detection. Potential trajectory loops are detected by looking for visual matches between the ORB descriptors associated with the last available high-rate keyframe $\mathcal{K}_H$, and the subset $\boldsymbol{\mathcal{K}}_O \subset \boldsymbol{\mathcal{K}}_L = \{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_m\}$ of m not visually linked keyframes. The search is conducted from the oldest to the latest keyframe in $\boldsymbol{\mathcal{K}}_O$. Then, a fundamental matrix RANSAC method [58] is used to prune outliers from the visual match results. A potential trajectory loop between the keyframe $\mathcal{K}_H$ and a keyframe $\mathcal{K}_j \in \boldsymbol{\mathcal{K}}_O$ is detected if the number of inliers is bigger than a threshold. If this is the case, then a corrected camera pose is computed, otherwise, the potential loop detection step is repeated. The j-keyframe $\mathcal{K}_j \in \boldsymbol{\mathcal{K}}_O$ from which a potential loop was found is defined as $\mathcal{K}_L$.

ii) Corrected camera pose. At this step, after a potential loop is detected, it is assumed that the camera-robot has returned to a previously mapped area. A corrected camera pose $\hat{\mathbf{c}}_c^N$ is defined as the pose of the camera-robot computed from the anchors $\mathbf{a}_i^N$ visually linked to the keyframe $\mathcal{K}_L$. Computing $\hat{\mathbf{c}}_c^N$ is the same as solving the Perspective-n-Point (PnP) problem. The PnP problem involves determining the camera pose, given a set of known 3D points (i.e. anchors) in the scene and their corresponding 2D projections in the image captured by the camera [59].

Let define the subset $\boldsymbol{\mathcal{A}}_V \subset \boldsymbol{\mathcal{A}}$ as the subset of m anchors $\mathbf{a}_i^N$ visually linked to the keyframe $\mathcal{K}_L$. And let define the subset $\boldsymbol{\mathcal{A}}_M \subseteq \boldsymbol{\mathcal{A}}_V$ as the subset of n anchors $\mathbf{a}_i^N$, belonging to $\boldsymbol{\mathcal{A}}_V$, that can be both projected and matched in the current high-rate keyframe $\mathcal{K}_H$ (See Figure 3). Also, let define respectively $\mathbf{h}_i(\mathbf{c}_H^N, \mathbf{a}_i)$ and $\mathbf{v}_i$, as the

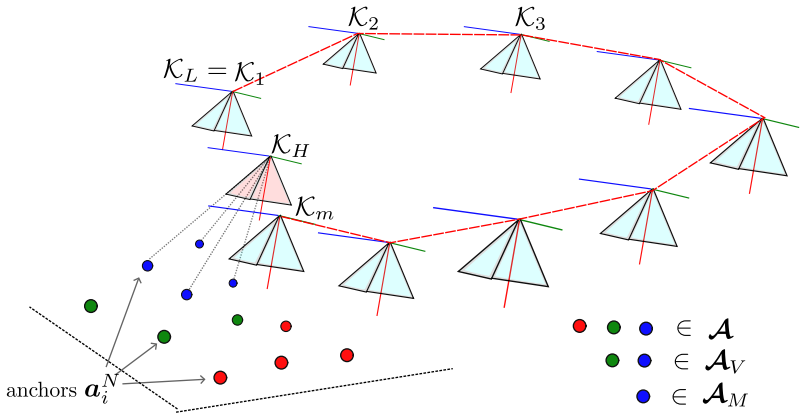

Fig. 3 A PnP technique is used for computing the corrected camera pose of the current high-rate keyframe with respect to older anchors.

predicted and the measured (matched) projection of the i-anchor $\mathbf{a}_i^N \in \boldsymbol{\mathcal{A}}_M$ in the keyframe $\mathcal{K}_H$, and let $\mathbf{c}_H^N$ the camera pose associated with the keyframe $\mathcal{K}_H$. Then, the PnP problem can be stated as the minimization of the total reprojection error $d(\mathbf{h}_i(\mathbf{c}_H^N, \mathbf{a}_i), \mathbf{v}_i)$ of the n anchors in $\boldsymbol{\mathcal{A}}_M$, or:

$$\min_{\mathbf{c}_H^N} \sum_{i=1}^{n} d(\mathbf{h}_i(\mathbf{c}_H^N, \mathbf{a}_i), \mathbf{v}_i)^2 \tag{11}$$

The corrected camera pose $\hat{\mathbf{c}}_c^N$ is the camera pose $\mathbf{c}_H^N$ that minimizes the PnP problem. In this work, the PnP-RANSAC method from the OpenCV library [60] is used to solve (11). In this case, a minimum number of four anchors in $\boldsymbol{\mathcal{A}}_M$ is required to solve the PnP problem, but a higher threshold can be set to improve robustness. Finally, the high-rate keyframe $\mathcal{K}_H$ used for computing the corrected camera pose is added to the global map, therefore becoming the m-keyframe $\mathcal{K}_m \in \boldsymbol{\mathcal{K}}_L$.

iii) Pose-SLAM. In the last step, the camera-robot pose $\hat{\mathbf{c}}_c^N$ associated with the latest keyframe $\mathcal{K}_m$ was computed with respect to previously mapped anchors. The corrected camera pose $\hat{\mathbf{c}}_c^N$ will allow the definition of a loop-closure constraint for minimizing the error drift accumulated over the robot trajectory. The technique used for correcting the global map after loop closure is based on the Pose-SLAM.

The Pose-SLAM represents the simplest form of a SLAM problem, which avoids building an explicit map of the robot environment. The general Pose-SLAM problem

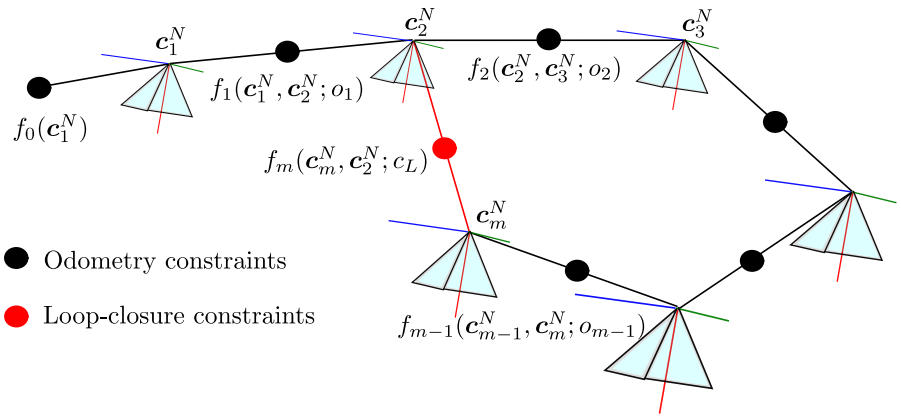

Fig. 4 Pose-SLAM formulation.

is formulated by defining two kinds of spatial constraints: i) the odometry which is defined between successive robot poses, and ii) loop-closure constraints which are defined when the robot re-visits a previous part of the environment (See Figure 4). In this work, the Pose-SLAM problem is solved by using a factor graph approach [49].

For modeling the factor graph, the m camera poses $\{\mathbf{c}_1^N, \mathbf{c}_2^N, ..., \mathbf{c}_m^N\}$ associated with the m keyframes $\{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_m\}$ belonging to the global map $\boldsymbol{\mathcal{K}}_L$ represent the poses of the robot over time. The unary factor $f_0(\mathbf{c}_1^N,)$ on the first pose $\mathbf{c}_1^N$ encodes the prior knowledge about $\mathbf{c}_1^N$. The binary factors $f_i(\mathbf{c}_i^N, \mathbf{c}_{i+1}^N; \mathbf{o}_i)$ relate the succesive poses $\mathbf{c}_i^N$ and $\mathbf{c}_{i+1}^N$, where $\mathbf{o}_i$ represent the i-odometry measurement. The binary factor $f_m(\mathbf{c}_m^N, \mathbf{c}_j^N; \mathbf{c}_L)$ represents the loop-closure constraint between the pose $\mathbf{c}_m^N$ and the pose $\mathbf{c}_j^N$, where $j < m$, given the loop-closure measurement $\mathbf{c}_L$.

The i-odometry measurement is computed by simply by $\mathbf{o}_i = \mathbf{c}_{i+1}^N - \mathbf{c}_i^N$. The loop-closure measurement is computed by $\mathbf{c}_L = \mathbf{c}_j^N - \hat{\mathbf{c}}_c^N$, where $\mathbf{c}_j^N$ is the camera pose associated with the j-keyframe $\mathcal{K}_j$ from which the loop-closure constraint has been established, and $\hat{\mathbf{c}}_c^N$ is the corrected camera posed computed in the previous step. The factor graph representing the Pose-SLAM problem is solved using the GTSAM library [49].

After the factor graph is optimized, a set of m corrected camera poses $\{\hat{\mathbf{c}}_1^N, \hat{\mathbf{c}}_2^N, ..., \hat{\mathbf{c}}_m^N\}$ is available. Then, a set of delta poses $\{\Delta\mathbf{c}_1^N, \Delta\mathbf{c}_2^N, ..., \Delta\mathbf{c}_m^N\}$ can

21

be computed by simply subtracting each initial pose from the corrected one, or $\Delta\mathbf{c}_i^N = \hat{\mathbf{c}}_i^N - \mathbf{c}_i^N$, where $i \in \{1, .., m\}$.

The following procedure is performed to update the global map $\boldsymbol{\mathcal{G}}$: Each i-camera pose, associated to the i-keyframe is simply updated with its optimized value, or $\mathbf{c}_i^N := \hat{\mathbf{c}}_i^N$, and each j-anchor $\mathbf{a}_j^N \in \boldsymbol{\mathcal{A}}$ is updated by $\mathbf{a}_j^N := \mathbf{a}_j^N + \Delta\mathbf{c}_i^N$, where $\Delta\mathbf{c}_i^N$ correspond to the pose update of the i-keyframe from which the j-anchor was initialized.

Finally, a Δ-Pose update $\Delta\mathbf{p} = \Delta\mathbf{c}_m^N$ is sent to the local SLAM process to correct the pose of the local SLAM state, where $\Delta\mathbf{c}_m^N$ is the pose update computed for the latest camera pose.

4 Results

The proposed monocular-based SLAM system was implemented in the C++ language and using ROS 2 (Robot Operating System 2) as the developing framework [47]. Each sub-system described in the system architecture (See Figure 2) is implemented as a ROS 2 component. The source code is available in [61].

Experimental results are presented for both robotic platforms: i) the differential robot, and ii) the quadrotor whose kinematic and measurement models were introduced in Section 3.2. In each case, for evaluating the performance of the proposed system the following general methodology was carried out: First, a known initial position was defined, then, the robot was (manually) driven far from its initial position and after that, it was driven back to the initial position, in both cases following different irregular paths. Finally, it was observed if the SLAM system was able to close the trajectory loop and how well was able to minimize the accumulated error drift by comparing the final estimated position with the actual one.

4.1 Differential robot experiment

A low-cost custom-built differential mobile robot was used for experiments (See figure 5). The robot sensors used in this configuration are: i) a lateral-pointing webcam

(Microsoft Studio), and ii) EMG30 motor/wheel encoders. From the monocular camera, 320x240 images were captured at 30 FPS. Wheel encoder data was captured at a frequency of 10Hz. The experiments were carried out in an unstructured indoor domestic environment (See figure 6, middle plots). In this case, the environment is mainly composed of low-texture elements such as walls, furniture, and floors, making it somehow challenging for visual-based robot applications.

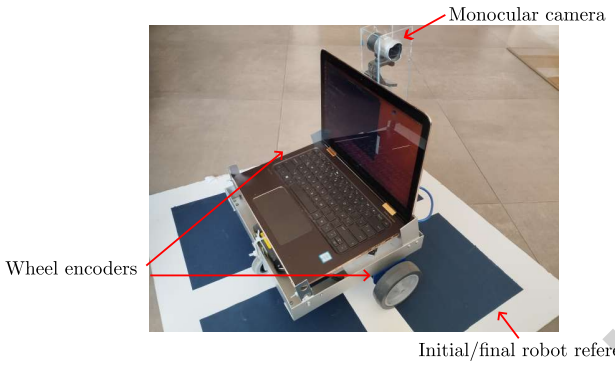

Fig. 5 A low-cost custom-built differential robot is used in experiments. In this case, a webcam and wheel encoders represent the system's sensory input. The initial/final trajectory reference also appears in the picture.

Figure 6 shows the results of one of the experiments. The upper plot shows a lateral (x-z) view of the estimates obtained from the proposed system. The middle plots show some sample frames captured by the monocular camera at different points of the robot's trajectory. The lower plot shows an upper (x-y) view of both the estimated robot trajectory and the estimated local/global map. It is important to note that in this figure, only the final (after loop-closure) mapping results are displayed. In this case, the red dots represent the local map at the final pose of the robot, meaning that at this time, several old map features had been already removed. Also, note that only very few physical spots of the environment were mapped to the final global map (green dots). This is due to the lack of texture in several elements of the environment, causing a lot of weak anchors to be pruned. In the figure, to facilitate the interpretation of the estimated map, the relation of some clusters of anchors to their physical counterparts in the sample camera frames is highlighted. In this experiment, the proposed SLAM

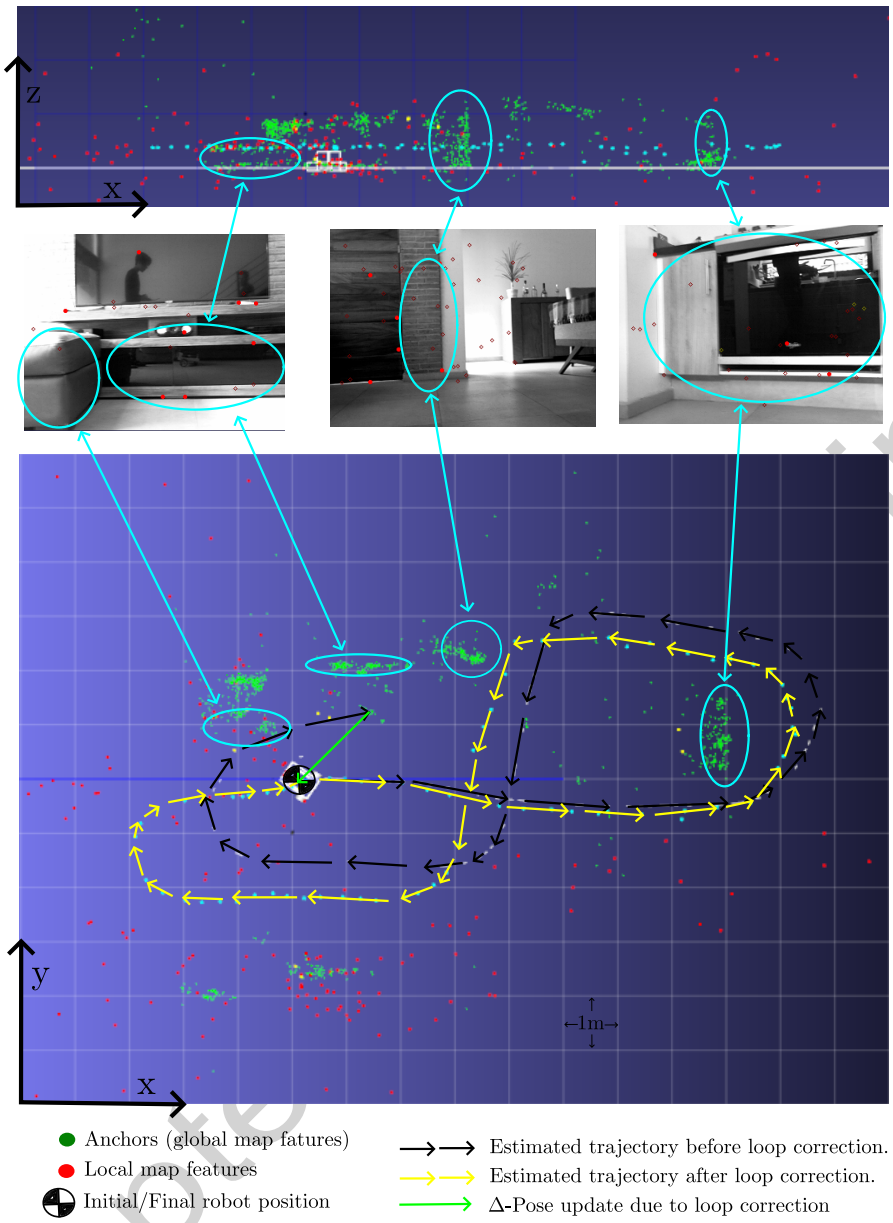

Fig. 6 The upper plot shows a lateral view (x-z) of the system estimates. Middle plots show sample frames captured by the robot's camera. The lower plot shows an upper view (x-y) of the system estimates. In this case, it is important to observe the comparison between the estimated trajectory before and after loop correction. Also, in this figure, it can be appreciated how the 3D map captures some of the structural elements of the environment.

24

system was able to recognize and close the loop when the robot returned to its initial position, therefore minimizing the accumulated estimation error drift in both the robot pose and map (global and local). In this case, arrows were added to the plot to better illustrate the estimated robot trajectory after loop correction (yellow arrows) and the robot trajectory before loop correction (black arrows).

4.2 Quadrotor experiment

A Bebop 2 quadrotor from Parrot [62] was used for experiments (See Figure 7, left plot). The quadrotor sensors used in this configuration are: i) the main monocular camera, ii) the altimeter, and iii) the AHRS. The camera was set to point down for capturing 320x240 images at 30 FPS. Altitude measurements were captured at 5Hz. Attitude measurements were captured from the AHRS at 5Hz. The experiments were carried out in an unstructured outdoor recreational park environment. To have a better interpretation of the results, the quadrotor was manually piloted from its initial position trying to follow a pedestrian circuit of the park to return near to the starting point.

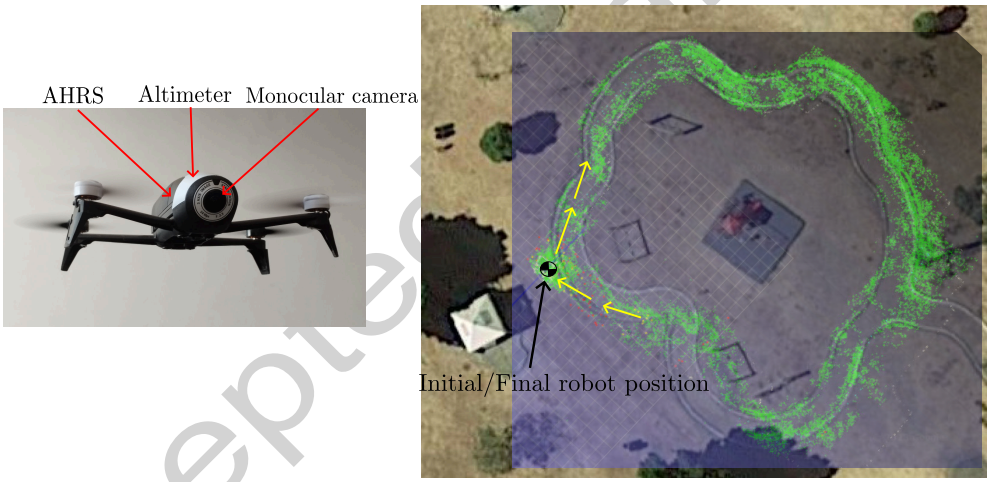

Fig. 7 The left plot shows the Bebop-2 from Parrot used in experiments. In this case, the altimeter, front camera, and AHRS of the drone were used as input sensors. The right plot shows an aerial view of the experimental environment as well as the overlapped image of the estimated Global/local map.

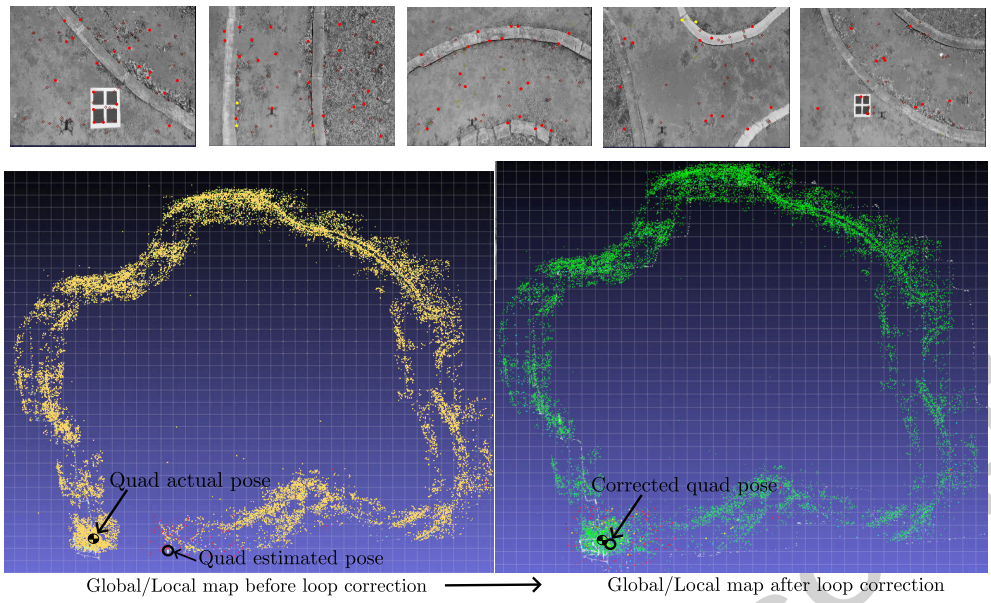

Fig. 8 The upper plots show some sample frames captured by the quadrotor's camera during its flight trajectory. The lower plots show a comparison between the system estimates before and after loop correction. In this case, it can be appreciated how the error position is minimized after the loop is closed.

Figure 7 shows the estimated final Global/local map superposed to an aerial view of the environment obtained from Google Maps. To improve the chances of detecting and closing the trajectory loop, the quadrotor performed a short overfly of the initial/final location both, at the beginning and at the end of the trajectory. In this experiment, the distance traveled by the quadrotor was 260 meters, and the duration of the flight was 278 seconds.

Figure 8 (upper plots) shows some of the frames captured by the quadrotor camera, from the beginning of the trajectory (left plot) to the end of the trajectory (right plot). The lower plots show the output of the system, just before the loop correction (left plot) and after the loop correction (right plot). Observe that at the moment before the loop correction, the quadrotor has returned near to its initial position, however, the estimates show an error drift of about 6 meters. On the other hand, when the system is able to detect and close the loop, the error drift is considerably minimized.

Fig. 9 Local SLAM process: number of map features per frame (upper plot), and computational time per frame (lower plot). In this case, it can be observed how the state size of the local SLAM is limited and therefore the computation time is maintained stable.

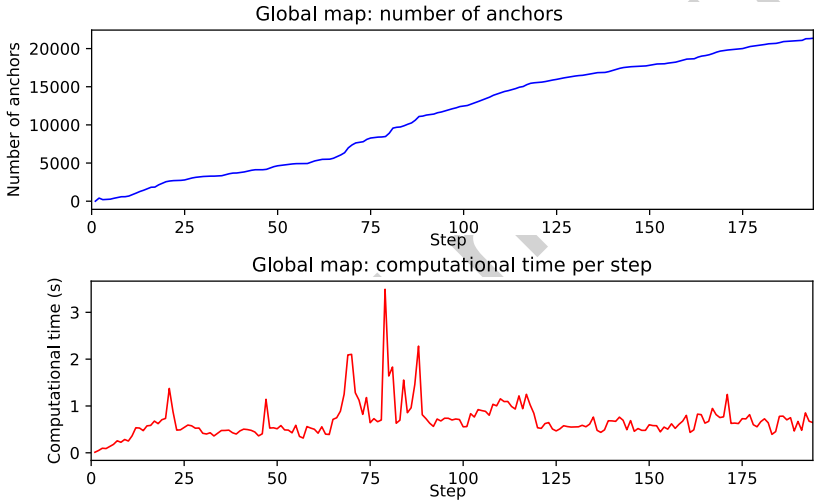

Fig. 10 Global mapping process: number of anchors composing the global map (upper plot), and computational time per step (lower plot). In this case, it is interesting to observe that the computation time is maintained stable despite the global map size increasing in time.

To empirically evaluate the computational performance of the proposed system some statistics were recorded for each process. Experiments were carried out in a PC

equipped with an Intel i9-9900K CPU. According to the OS task manager, the execution of the whole system required an average of 20% CPU usage for the quadrotor's experiment.

Figure 9 shows the evolution over time of the number of map features managed by the local SLAM process, as well as the computational time per frame. In this case, it can be observed that the computational time remains constant due to the number of features being maintained bounded. Moreover, the total computation time expended by the Local SLAM process was 112.77 seconds, so considering that the total execution time was 278 seconds, it can be seen that real-time performance can be easily achieved with the hardware used in experiments.

Figure 10 shows the evolution over time of the number of anchors composing the global map, as well as the computational time per step of the global mapping process. In this case, it is important to note that despite some peaks, the computational time per step remains stable with respect to the global map size. For this experiment, the final global map was composed of 21357 anchors, and the total computation time was 134.03 seconds, so the real-time performance was easily achieved by the global mapping process.

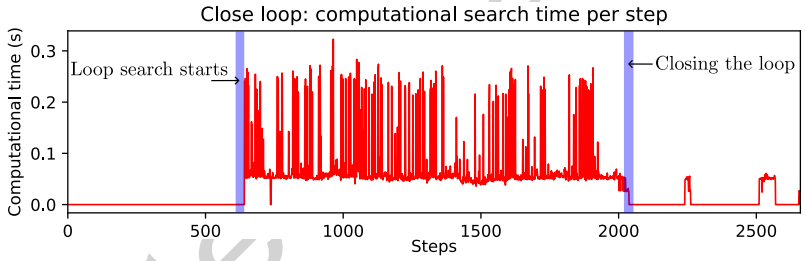

Fig. 11 Close loop search: computational search time per step. In this case, the heuristics derived from the use of a visibility graph allow to limit times when the loop search is carried out and also to maintain the computation time stable.

Figure 11 shows the evolution of computational time per step for the potential loop detection task described in Section 3.4. In this case, the other tasks belonging to the loop correction process were not included in this statistic because they are only performed if a loop is detected. Therefore the loop detection task represents the major

computational consumption for the loop correction process. Also, it is important to note that the loop detection (search) task is only performed after the robot moves a minimum distance from its initial position or the last closing loop point. Observe that for this experiment the computation time remains stable, and the total computation time was 105.09 seconds.

5 Discussion

In the above section, experimental results obtained from a differential robot and a quadrotor operating each one in a different (indoor/outdoor) unstructured environment were presented. In both cases, the trajectories of the robots were estimated reasonably well after the trajectory loops were detected and closed. Here, it is important to note that, for the proposed architecture, the minimization of error pose after loop closure greatly depends on the accuracy and precision of the method used for correcting the camera pose (see Section 3.4). For instance, in the case of the quadrotor experiments, a pose error from 1 to 2 meters was typically obtained from the PnP technique used in the current implementation, which after hundreds of meters of flight travel might not be a big problem for certain applications. However, if less error is required, the correction camera pose task should be improved. Fortunately, the modularity of the proposed architecture allows us to potentially use other techniques for implementing the same task. At this point, it is important to remark that most of the parameters have been heuristically set the same to obtain good performance with the robotic platforms and environments used in experiments, but still, some degree of manual tuning can be needed to achieve better performance for a particular scenario.

One of the major goals of the proposed methodology is to achieve real-time performance while building and maintaining maps of several thousands of visual features. In this case, experimental results show that this goal can be achieved in practice using consumer-degree hardware. On the other hand, for a more theoretical analysis purpose, Table 3 shows the computational complexity for typical implementations of the

Task	Complexity	Variables
Local SLAM:		
EKF-SLAM	$\mathcal{O}(f(n,m))$ $f \rightarrow$ Quadratic function	$n \rightarrow$ state dimension $m \rightarrow$ measurement state dimension
Global SLAM:		
Bundle-adjustment	$\mathcal{O}(f(n,m,k))$ $f \rightarrow$ Cubic function	$n \rightarrow$ 3D points $m \rightarrow$ camera poses $k \rightarrow$ 2D image features
Loop detection	$\mathcal{O}(n)$	$n \rightarrow$ keyframes
PnP problem	$\mathcal{O}(n^3)$	$n \rightarrow$ 3D points
Graph SLAM	$\mathcal{O}(n+o+nT)$	$n \rightarrow$ robot poses $o \rightarrow$ odometry measurments $T \rightarrow$ iterations per convergence

Table 3 The theoretical computational complexity of each of the main tasks composing the proposed method.

main tasks of the proposed architecture. Here, it is important to note that the dimension of the variables involved in the following tasks: *EKF-SLAM*, *Bundle-adjustment*, and *PnP problem* are upper bounded for each iteration, therefore in practice the complexity for each of these tasks is constant ($\mathcal{O}(1)$). Regarding the *Loop detection* tasks a small but linearly increasing computational time per step was expected, but experimental results showed more like a constant time performance (see Table 11). In this case, it seems like the restrictions such as the only use of visually-linked keyframes are restricting in practice the computational cost of this task, which is of course a desirable behavior. Finally, it can be observed that assuming a maximum number of iterations, which is the most common approach, the complexity of the *Graph SLAM* task is linear. In this case, it is important to note that this task is executed only each time that a loop is closed and also that in practice, it can be computed very fast for thousands of variables, making it also suitable for the goals of the proposed methodology.

30

6 Conclusions

In this work, a full monocular-based SLAM system for mobile robots has been presented. The architecture of the proposed system is composed of two main processes running concurrently: i) a filter-based local SLAM process, which takes as inputs all the available sensory inputs to produce high-rate estimates of the robot's state and a local map, and ii) an optimization-based global SLAM process, which takes keyframes as inputs for building and maintaining a consistent global map of the robot's environment as well as to minimize the error drift accumulated in the system by detecting and correcting trajectory loops. The main idea behind the proposed architecture is to minimize the drawbacks and take advantage of the respective strengths of the two main methodologies for implementing SLAM systems: the filter and optimization methods.

The proposed scheme can be easily applied to different mobile robotic configurations by taking advantage of the inherent properties of the filter-based SLAM methods for fusing diverse sensor inputs. Specifically, in this work, implementation and experimental details are given for two platforms: i) a differential mobile robot, and ii) a quadrotor.

A ROS-2-based C++ open-source implementation of the proposed system is provided. Experimental results with real data taken in non-controlled (out-of-lab) environments show that the proposed system can be potentially used in real low-cost robotic applications. Particularly, the computational performance was empirically analyzed to show proposed scheme performs well as the size of the mapped area increases.

7 Future work

The presented research could open up several avenues for future investigations and improvements. Here, we outline some potential areas for future work: i) To improve the accuracy and precision of pose estimation after loop closures, alternative techniques, and strategies could be studied for implementing the method for correcting the camera-robot pose at loop closure. ii) To improve the applicability of the proposed visual-based SLAM architecture to a wider class of robotic platforms, the potential use of other

kinds of visual sensors as stereo systems, depth cameras, etc. could be considered. iii) Future work also could point to extending the current system architecture to include a control process that uses as feedback the state estimate by the SLAM system to allow the robot to perform fully autonomous exploration missions.

Supplementary information. Source code is available in https://github.com/rodrigo-munguia/Hybrid_VSLAM thought MIT license.

Acknowledgments. The first author wants to thank to Roderic Munguia for his contribution to this work as a research assistant.

Declarations

Funding. This research was conducted with no funding.

Competing Interests. All the authors declare they have no financial interest.

Author Contributions. The contribution of each author is: Conceptualization, R.M. and J.-C.T; methodology, G.O.-P and C.I.A.; software, R.M; validation, J.-C.T, G.O.-P. and C.I.A.; investigation, R.M. and G.O.-P; resources, C.I.A.; writing—original draft preparation, R.M.; writing—review and editing, J.-C.T. and C.I.A.; visualization, G.O.-P; supervision, R.M. and J.-C.T; lab resources, R.M. All authors have commented on previous versions of the manuscripts. All authors read and approved the final manuscript.

Ethics approval. Not applicable.

Consent to participate. Not applicable.

Consent to publish. Not applicable.

32

References

[1] Jiang, D. *et al.* Manipulator grabbing position detection with information fusion of color image and depth image using deep learning. *Journal of Ambient Intelligence and Humanized Computing* **12**, 10809–10822 (2021).

[2] Parikh, P., Trivedi, R., Dave, J., Joshi, K. & Adhyaru, D. Design and development of a low-cost vision-based 6 dof assistive feeding robot for the aged and specially-abled people. *IETE Journal of Research* **0**, 1–29 (2023). URL https://doi.org/10.1080/03772063.2023.2173665.

[3] An, X. & Wang, Y. Smart wearable medical devices for isometric contraction of muscles and joint tracking with gyro sensors for elderly people. *Journal of Ambient Intelligence and Humanized Computing* 1–12 (2021).

[4] Kumar, A. Real-time performance comparison of vision-based autonomous landing of quadcopter on a ground moving target. *IETE Journal of Research* **0**, 1–18 (2021). URL https://doi.org/10.1080/03772063.2021.1963332.

[5] Rao, D. & Gupta, M. Neuro-fuzzy controller for control and robotics applications. *Engineering Applications of Artificial Intelligence* **7**, 479–491 (1994). URL https://www.sciencedirect.com/science/article/pii/0952197694900272.

[6] Ding, H. Motion path planning of soccer training auxiliary robot based on genetic algorithm in fixed-point rotation environment. *Journal of Ambient Intelligence and Humanized Computing* **11**, 6261–6270 (2020).

[7] Tong, C. Three-dimensional reconstruction of the dribble track of soccer robot based on heterogeneous binocular vision. *Journal of Ambient Intelligence and Humanized Computing* **11**, 6361–6372 (2020).

[8] Hamidi, K. E., Mjahed, M., Kari, A. E., Ayad, H. & Gmili, N. E. Design of hybrid neural controller for nonlinear mimo system based on narma-l2 model. *IETE Journal of Research* **69**, 3038–3051 (2023). URL https://doi.org/10.1080/

03772063.2021.1909507.

[9] Singh, S., Khosla, A. & Kapoor, R. Visual-thermal fusion-based object tracking via a granular computing backed particle filtering. *IETE Journal of Research* **0**, 1–16 (2022). URL https://doi.org/10.1080/03772063.2022.2030251.

[10] Khan, M. F., ul Islam, R. & Iqbal, J. *Control strategies for robotic manipulators*, 26–33 (2012).

[11] Asha, C. S. & Narasimhadhan, A. V. Visual tracking using kernelized correlation filter with conditional switching to median flow tracker. *IETE Journal of Research* **66**, 427–438 (2020). URL https://doi.org/10.1080/03772063.2018.1492356.

[12] Li, X. Robot target localization and interactive multi-mode motion trajectory tracking based on adaptive iterative learning. *Journal of Ambient Intelligence and Humanized Computing* **11**, 6271–6282 (2020).

[13] Yağ, İ. & Altan, A. Artificial intelligence-based robust hybrid algorithm design and implementation for real-time detection of plant diseases in agricultural environments. *Biology* **11**, 1732 (2022).

[14] Belge, E., Altan, A. & Hacıoğlu, R. Metaheuristic optimization-based path planning and tracking of quadcopter for payload hold-release mission. *Electronics* **11**, 1208 (2022).

[15] Altan, A. *Performance of metaheuristic optimization algorithms based on swarm intelligence in attitude and altitude control of unmanned aerial vehicle for path following*, 1–6 (IEEE, 2020).

[16] Gupta, A. & Fernando, X. Simultaneous localization and mapping (slam) and data fusion in unmanned aerial vehicles: Recent advances and challenges. *Drones* **6**, 85 (2022).

[17] Ullah, I., Su, X., Zhang, X. & Choi, D. Simultaneous localization and mapping based on kalman filter and extended kalman filter. *Wireless Communications and*

Mobile Computing **2020**, 12 (2020).

[18] Kim, P., Coltin, B. & Kim, H. J. *Linear rgb-d slam for planar environments* (2018).

[19] Chen, M., Yang, S., Yi, X. & Wu, D. *Real-time 3d mapping using a 2d laser scanner and imu-aided visual slam*, 297–302 (2017).

[20] Motlagh, H. D. K., Lotfi, F., Taghirad, H. D. & Germi, S. B. *Position estimation for drones based on visual slam and imu in gps-denied environment*, 120–124 (2019).

[21] Zhou, H. *et al.* Structslam: Visual slam with building structure lines. *IEEE Transactions on Vehicular Technology* **64**, 1364–1375 (2015).

[22] Bloesch, M., Burri, M., Omari, S., Hutter, M. & Siegwart, R. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research* **36**, 1053–1072 (2017). URL https://doi.org/10.1177/0278364917728574.

[23] Quan, M., Piao, S., Tan, M. & Huang, S.-S. Accurate monocular visual-inertial slam using a map-assisted ekf approach. *IEEE Access* **7**, 34289–34300 (2019).

[24] Holmes, S., Klein, G. & Murray, D. W. *A square root unscented kalman filter for visual monoslam*, 3710–3716 (2008).

[25] Lee, S.-H. Real-time camera tracking using a particle filter combined with unscented Kalman filters. *Journal of Electronic Imaging* **23**, 1 – 19 (2014). URL https://doi.org/10.1117/1.JEI.23.1.013029.

[26] Tang, M., Chen, Z. & Yin, F. Robot tracking in slam with masreliez-martin unscented kalman filter. *Int. J. Control Autom. Syst.* **18**, 2315–2325 (2020).

[27] Celik, K., Chung, S.-J., Clausman, M. & Somani, A. K. *Monocular vision slam for indoor aerial vehicles*, 1566–1573 (2009).

[28] Wen, S., Chen, J., Lv, X. & Tong, Y. Cooperative simultaneous localization and mapping algorithm based on distributed particle filter. *International Journal of Advanced Robotic Systems* **16**, 1729881418819950 (2019). URL https://doi.org/10.1177/1729881418819950.

[29] Liu, W. Slam algorithm for multi-robot communication in unknown environment based on particle filter. *J Ambient Intell Human Comput* (2021).

[30] Eustice, R. M., Singh, H., Leonard, J. J. & Walter, M. R. Visually mapping the rms titanic: Conservative covariance estimates for slam information filters. *The International Journal of Robotics Research* **25**, 1223–1242 (2006). URL https://doi.org/10.1177/0278364906072512.

[31] Zhou, W., Valls MirÓ, J. & Dissanayake, G. Information-efficient 3-d visual slam for unstructured domains. *IEEE Transactions on Robotics* **24**, 1078–1087 (2008).

[32] Klein, G. & Murray, D. *Parallel tracking and mapping for small ar workspaces*, 225–234 (2007).

[33] Mur-Artal, R., Montiel, J. M. M. & Tardós, J. D. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics* **31**, 1147–1163 (2015).

[34] Concha, A. & Civera, J. *Dpptam: Dense piecewise planar tracking and mapping from a monocular sequence*, 5686–5693 (2015).

[35] Lin, X., Wang, F., Guo, L. & Zhang, W. An automatic key-frame selection method for monocular visual odometry of ground vehicle. *IEEE Access* **7**, 70742–70754 (2019).

[36] Zhang, Y., Xu, X., Zhang, N. & Lv, Y. A semantic slam system for catadioptric panoramic cameras in dynamic environments. *Sensors* **21** (2021). URL https://www.mdpi.com/1424-8220/21/17/5889.

36

[37] Liao, Z., Wang, W., Qi, X. & Zhang, X. Rgb-d object slam using quadrics for indoor environments. *Sensors* **20** (2020). URL https://www.mdpi.com/1424-8220/20/18/5150.

[38] Zhao, Y. & Vela, P. A. *Good feature selection for least squares pose optimization in vo/vslam*, 1183–1189 (2018).

[39] Reif, K., Gunther, S., Yaz, E. & Unbehauen, R. Stochastic stability of the discrete-time extended kalman filter. *IEEE Transactions on Automatic Control* **44**, 714–728 (1999).

[40] Kluge, S., Reif, K. & Brokate, M. Stochastic stability of the extended kalman filter with intermittent observations. *IEEE Transactions on Automatic Control* **55**, 514–518 (2010).

[41] Huang, G. P., Mourikis, A. I. & Roumeliotis, S. I. Observability-based rules for designing consistent ekf slam estimators. *The International Journal of Robotics Research* **29**, 502–528 (2010). URL https://doi.org/10.1177/0278364909353640.

[42] Huang, G. P., Mourikis, A. I. & Roumeliotis, S. I. A quadratic-complexity observability-constrained unscented kalman filter for slam. *IEEE Transactions on Robotics* **29**, 1226–1243 (2013).

[43] Lee, S.-M., Jung, J., Kim, S., Kim, I.-J. & Myung, H. Dv-slam (dual-sensor-based vector-field slam) and observability analysis. *IEEE Transactions on Industrial Electronics* **62**, 1101–1112 (2015).

[44] Poulose, A. & Han, D. S. Hybrid indoor localization using imu sensors and smartphone camera. *Sensors* **19** (2019). URL https://www.mdpi.com/1424-8220/19/23/5084.

[45] Strasdat, H., Montiel, J. & Davison, A. J. Visual slam: Why filter? *Image and Vision Computing* **30**, 65–77 (2012). URL https://www.sciencedirect.com/science/article/pii/S0262885612000248.

37

[46] Munguia, R., Trujillo, J.-C., Guerra, E. & Grau, A. A hybrid visual-based slam architecture: Local filter-based slam with keyframe-based global mapping. *Sensors* **22** (2022). URL https://www.mdpi.com/1424-8220/22/1/210.

[47] Macenski, S., Foote, T., Gerkey, B., Lalancette, C. & Woodall, W. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics* **7**, eabm6074 (2022). URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[48] Agarwal, S., Mierle, K. & Others. Ceres solver. http://ceres-solver.org.

[49] Dellaert, F. & Kaess, M. Factor graphs for robot perception. *Foundations and Trends in Robotics* **6**, 1–139 (2017). URL http://dx.doi.org/10.1561/2300000043.

[50] Conte, F. & Martinelli, A. *A hybrid filter-based and graph-based approach to slam*, 999 (2010).

[51] Ding, Y., Xiong, Z., Xiong, J., Cui, Y. & Cao, Z. Ogi-slam2: A hybrid map slam framework grounded in inertial-based slam. *IEEE Transactions on Instrumentation and Measurement* **71**, 1–14 (2022).

[52] Bouguet, J. *Camera calibration toolbox for matlab* (2008). URL http://www.vision.caltech.edu/bouguetj/calib_doc.

[53] Durrant-Whyte, H. & Bailey, T. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine* **13**, 99–110 (2006).

[54] Bailey, T. & Durrant-Whyte, H. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine* **13**, 108–117 (2006).

[55] Munguia, R. & Grau, A. Monocular slam for visual odometry: A full approach to the delayed inverse-depth feature initialization method. *Mathematical Problems in Engineering* **2012** (2012).

38

[56] Urzua, S., Munguía, R. & Grau, A. Vision-based slam system for mavs in gps-denied environments. *International Journal of Micro Air Vehicles* **9**, 283–296 (2017). URL https://doi.org/10.1177/1756829317705325.

[57] Triggs, B., McLauchlan, P. F., Hartley, R. I. & Fitzgibbon, A. W. Triggs, B., Zisserman, A. & Szeliski, R. (eds) *Bundle adjustment — a modern synthesis.* (eds Triggs, B., Zisserman, A. & Szeliski, R.) *Vision Algorithms: Theory and Practice*, 298–372 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2000).

[58] Hartley, R. & Zisserman, A. *Multiple View Geometry in Computer Vision* 2 edn (Cambridge University Press, 2004).

[59] Wu, H. Z., Y. Pnp problem revisited. *J Math Imaging Vis* **24**, 131–141 (2006).

[60] Itseez. Open source computer vision library. https://github.com/itseez/opencv (2015).

[61] Source code. https://github.com/rodrigo-munguia/Hybrid_VSLAM.

[62] Parrot bebop drone for developers. https://developer.parrot.com/docs/SDK3/.

39