Multi-RC robot SLAM with A Customized Front-end and COVINS-G Backend

Marcelo Contreras Cabrera 1796906, Yunchen(Leo) Ge 1813817

***Abstract*—Multi-agent mobile robot systems are becoming more popular in research and industrial communities. The following sub-fields: *i)* multi-agent perception, *ii)* localization and mapping, *iii)* distributed control, and *iv)* system designs are the major fields of interest. In this project, we will investigate the advantages and complexities of a centralized multi-agent SLAM system (also known as collaborative SLAM) and analyze the possible improvements compared to single-agent SLAM. We propose leveraging the COVINS-G backend [1] as a centralized server that receives keyframes and images from the single-agent SLAM, searching for loop candidates and optimizing their paths in a single trajectory by Pose Graph Optimization. A potential experimental setup that includes the system design and leverages LiDAR scans as ground truth is proposed.**

I. Introduction

Multi-agent mobile robot systems have gained more interest from research and industrial communities in recent years due to the diversity of their applications, the potential of robot-robot interactions in a highly autonomous environment, and the design challenges of these systems. For the research topics in this field, the following research topics including *i)* multi-agent perception [2], [3] (perception of the environment and other robots, etc.), *ii)* localization and mapping [4], [5] (multi-agent localization problem, SLAM etc), *iii)* distributed control [6]–[9], and *iv)* system designs [4], [10], [11] (centralized/distributed communication, offline/cloud-based computations, etc) are catching attention. For industrial and commercial applications, multi-robot systems are becoming popular in scenarios such as autonomous warehouses and indoor/outdoor patrolling and monitoring, which require robust and computation-efficient methods that resolve the complexities introduced by multi-agent systems. In this work, we investigate the advantages and complexities of the multi-agent SLAM problem (also called collaborative SLAM) with a real RC car platform. We select a centralized collaborative SLAM framework that processes the inputs from the front ends of all agents on a central server. In particular, we verify the computation efficiency and the compatibility of our stereo visual odometry front-end with the publicly available multi-agent SLAM backend COVINS-G [1]. We also analytically compare the accuracy of the multi-agent SLAM framework with a highly accurate LiDAR SLAM framework that is regarded as ground truth. To summarize, we have the following objectives for this project:

- Designing a real Ackermann steering RC car platform that features hierarchical processing design which maximizes the processing capability of each part with extendability to variety of sensors
- Verifying the compatibility of our stereo visual SLAM front-end (customly crafted for this work) with the selected collaborative SLAM backend framework COVINS-G; analytically investigating the computational efficiency of the overall framework
- Numerically comparing the accuracy of the multi-agent SLAM framework with a highly accurate LiDAR SLAM framework

The remainder of the paper is organized as follows: Section II presents the related works in collaborative SLAM based on their design categories. Sections III-A to III-B present the overall system design that features the hierarchical processing and the kinematic model used by the low-level processing part for the RC car platform. Section III-C discusses the robot agent's SLAM, including problem statements and front-end and back-end designs from the high-level processing part. Section III-D discusses the centralized server COVINS-G that fuses the inputs from all mobile robot agents. The system setup and experimental results are presented in Section IV, and we conclude our project in Section V.

II. Related works

Collaborative SLAM comes in different flavors: i) delivering all agent map fusion procedures to a centralized server, ii) distributing the load uniformly across all agents, and iii) sending several copies of the joint map to each agent (no map splits) and letting them process it just with onboard computing.

For centralized collaborative SLAM systems, CVI-SLAM [12] paved out of the basis to establish cooperation between agents and a centralized server where landmarks, keyframes, and images were sent bidirectionally, such as any correction introduced by the server after PGO or BA could be sent back to the on-board SLAM. It introduced key elements such as place recognition for loop closing and PGO to align the agent's trajectory. It even removed information on redundant keyframes following information theory. Nevertheless, it was heavily ROS-dependent, and several modules could be optimized further. In COVINS [13], the authors introduced a p2p communication protocol to make the CVI-SLAM framework flexible and rely less on ROS. In addition, they included a routine to merge the agent's maps into a single one to lower the memory footprint. On top of that, they modified the optimization routine by first applying a series of PGO iterations and running a Global Bundle Adjustment at the end of the sequence to lower the computation burden. AdaptSLAM [14] introduced several novelties not in the architecture design but in the keyframe selection and map managing. The authors proposed quantifying the uncertainty introduced by each keyframe as a criterion for their selection and efficiently constructing local/global maps under resource constraints. CMD-SLAM [15] is another centralized visual SLAM but utilizes direct dense tracking instead of feature sparse and achieves a significant message compression between agent and server using LiDAR-based descriptor for the local map. Furthermore, to provide an informative initial position for PGO, their approach does a first alignment between agent trajectories with Iterative Closes Point, a LiDAR-based registration, after detecting loop closures. Distributed approaches send chunks of the global map of each agent and perform the loop-closing search within that region, thus splitting the computational load and achieving faster inference time. Kimera-Multi [16] is a dense-semantic distributed multi-agent SLAM that detects loops by sending DBoW2 descriptors whenever p2p communication is available between agents. Each agent is in charge of correcting their trajectory and map with the loop closure in a PGO, and then the refined map is propagated to the agents again. The optimized map is reused for local map matching, and the onboard trajectory achieves higher precision. In addition to all these contributions, the map contains semantic labels refined in the PGO for high-level scene understanding for motion planning. IRBCD [17] proposed using a Rienmann optimization partition algorithm to send an equal amount of load on PGO optimization to each robot. This solves the fundamental problem of having a subset of robots idle due to a hefty load in optimization/map merging for one of the robots. MAGIC-SLAM [18] takes some of the ideas of the previous work but adapts them to the context of Gaussian Splats instead of 3D point landmarks for scene reconstruction. Gaussian Splatting can render novel views with dense-pixel quality in real-time, in contrast with NeRF, and with a low GPU memory footprint. Nonetheless, procedures such as tracking, keyframe selection, and pose inference must be reformulated to account for the features of 3D Gaussian Splats. That includes the place recognition where the authors utilized DinoV2 [19], a foundational vision model, to compute descriptors and found loop candidates.

On the other hand, decentralized methods broadcast the global map of each agent but in chunks; therefore, each agent processes the whole load and tends to have a suboptimal runtime, though with higher robot independence under communication troubleshoots or outliers. DOOR-SLAM [20] leverages a pairwise consistent measurement set maximization to detect spurious loop candidates that have more chances of being in the pose graph when there is no full connectivity between robots or when a robot corrects its trajectory but has not been able to broadcast it entirely. In this way, a robot can detect its own set of outliers regardless of the pose graphs of its peers and achieve better individual performance. Swarm-SLAM [21] also uses a a maximum algebraic connectivity augmentation problem for outlier detection with an additional step of graph sparsification to achieve faster performance and reduce load on decentralized schemes. Furthermore, their pipeline exploits sensor input from several sources, including LiDAR, RGB-D, IMU, and wheel/legged odometry, as constraints in the PGO. D^{2}SLAM is a promising intersection of decentralized and distributed approaches. It builds upon p2p communication

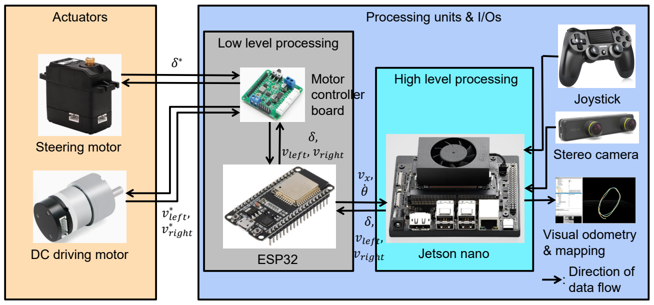


Figure 1: The overall system design for a multi-purpose Ackermann steering RC car.

protocols to detect loop candidates in a distributed way using NetVLAD and Ceres optimizer for PGO. However, the front-end and full BA is run on-board on each agent (decentralized). The systems could also distribute the BA when the number of outliers reaches a threshold, or local map matching quality is poor. In that sense, D^{2}SLAM has the advantages of both worlds.

III. Methodology

A. System design overview

We include the overall system design diagram in Fig. 1, which includes the following major components: one servo motor for front wheel steering, two DC motors for differential speed rear wheels, two motor controller boards for the servo motor and the DC motors, respectively, one ESP32 microcontroller for low-level computation tasks, one Jetson nano super for high-level computation tasks, one Xbox joystick for manual control mode of the RC car, and arbitrary on-board sensors for perception inputs. The system design mainly adopts the separation of the high-level processing part with the Jetson nano platform from the low-level processing part to maximize the computation performance of each part. The low-level part is responsible for the tasks requiring critical real-time performances, such as reading motor encoder values accurately and emergency brakes, and the high-level part is responsible for computation and memory-intensive tasks, such as visual odometry and path planning. The low-level part and high-level parts communicate with each other via ROS_serial package in a data format that is discussed in Section III-B.

The Ackermann steering at the RC car's front side utilizes a differential steering mechanism that turns the front left and front right wheels at different steering angles for a given turning radius of the RC car. The differential steering angles can be represented as an equivalent steering angle at the center of two front wheels, one of the control inputs (discussed in Section III-B). We select the position control mode with its source library on ESP32, available from the manufacturer, to control this target steering angle. A position refers to the high-resolution absolute encoder on the servo motor, which reads from 0 - 4095 for a full resolution. Therefore, a calibration process is required to map the servo motor's position to the actual steering angle at the center of the front wheels. Consider the two maximum steering angles at the center of the front

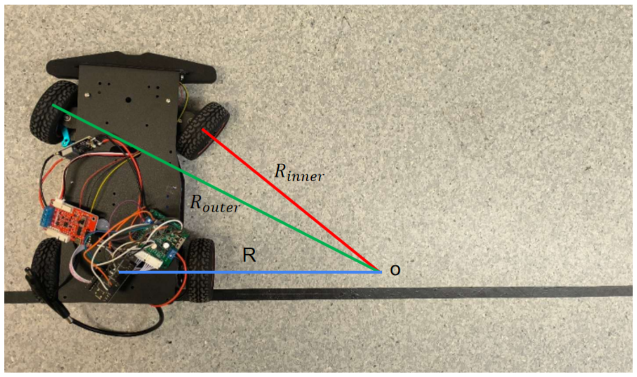


Figure 2: A calibration scene for mapping steering angles to servo motor positions. The estimated turning radius R at the center of rear wheels is determined from the estimated intersection of the turning radius of the inner turning wheel R_{inner}, the turning radius of the outer turning wheel R_{outer}, and the line of lateral extension at the center of rear wheels.

wheels for left turns ($\delta_{l,max}$) and right turns ($\delta_{r,max}$) that have the corresponding equivalent servo motor positions $p_{l,max}$ and $p_{r,max}$, respectively. We assume a linear mapping from input steering angles δ to servo motor positions p is adopted such that $p = p_{l,max} + \Delta_{\delta 2p}(\delta - \delta_{l,max})$ where $\Delta_{\delta 2p}$ is the slope of this linear mapping given by $\Delta_{\delta 2p} = \frac{p_{r,max} - p_{l,max}}{\delta_{r,max} - \delta_{l,max}}$. $p_{l,max}$ and $p_{r,max}$ are determined by the motor debugging software on the PC available from the manufacturer, and $\delta_{l,max}$ and $\delta_{r,max}$ are determined by the estimated turning radius R at the center of the rear wheels with $\delta = \arctan(\frac{L}{R})$ where L is the distance between the front and rear wheels of the RC car. A calibration scene is illustrated in Fig. 2.

The two DC motors that drive the rear wheels are controlled separately with velocity control mode using the motor controller board and ESP32 in the system design diagram. The DC motors have high-resolution AB phase encoders, each generating 15,000 pulses for a full resolution, so the encoder generates 60,000 pulses in total for a full resolution. We track half of the pulses (30000 pulses) to lower the computation load on ESP32 while simultaneously keeping a good rotation estimation resolution. Then, the instantaneous rotation velocity of each motor can be estimated from the number of pulses that have passed in a short period of time (e.g., 10 milliseconds) controlled by a timer function on ESP32, and we convert the rotation speed to linear velocity based on the measured wheel radius of rear wheels. A simple PID controller is designed to determine the pulse width modulation (PWM) value at discrete periods (e.g., 10 milliseconds) to control the rear wheels at target linear velocities denoted as v_{left} and v_{right} for the left and right rear wheels, respectively.

B. *Kinematic model in low-level controller*

In this section, we discuss the data format that is transferred between the low-level processing part and the high-level processing part in the system design diagram, which requires the kinematic model of an Ackermann steering vehicle and the separation of autonomous control mode and manual control mode (using a joystick controller). We use the bicycle steering model of an Ackermann steering model as the motion model. The control inputs sent by the high-level processing part are the equivalent steering angle for the front wheels and the linear velocities of left and right rear wheels (i.e., $u = [\delta \ v_{left} \ v_{right}]^T$), and the instantaneous motion output of the vehicle sent by the low-level processing part is the longitudinal linear velocity and the angular velocity at the center of rear wheels (i.e. $y = [v_x \ \dot{\theta}]^T$). The following kinematic equations of a bicycle are used to convert between the quantities interchangeably. From $[\delta \ v_{left} \ v_{right}]^T$ to $[v_x \ \dot{\theta}]^T$:

$$R = \frac{L}{\tan(\delta)} \tag{1}$$

$$v_x = \frac{v_{left}}{\frac{R - \frac{D}{2}}{R}} = \frac{v_{right}}{\frac{R + \frac{D}{2}}{R}} \tag{2}$$

$$\dot{\theta} = \frac{v_x}{R} \tag{3}$$

and from $[v_x \ \dot{\theta}]^T$ to $[\delta \ v_{left} \ v_{right}]^T$:

$$R = \frac{v_x}{\dot{\theta}} \tag{4}$$

$$\delta = \arctan(\frac{L}{R}) \tag{5}$$

$$v_{left} = v_x \frac{R - \frac{D}{2}}{R} \tag{6}$$

$$v_{right} = v_x \frac{R + \frac{D}{2}}{R} \tag{7}$$

in which L and D are the distance between front and rear wheels and the distance between left and right wheels, respectively. The control inputs are transferred with a std_msgs/Float32MultiArray message in ROS which is easy to contain the synchronized input variables and to be directly executed by the motor controller and ESP32, while the motion outputs are transferred with a geometry_msgs/Twist message in ROS which is a commonly selected ROS message type subscribed by the ROS nodes on the high level processing part for tasks such as visual odometry and path planning.

The separation of autonomous control mode and manual control mode is required since the motion commands sent by the ROS nodes for autonomous controls are often with geometry_msgs/Twist messages. Linear and angular velocity must co-exist as long as the angular velocity does not equal zero. In contrast, manual control mode allows a joystick to turn the steering wheels without driving the rear wheels. We design a ROS node that sets up the control mode, and if manual control is selected, we first map the joystick's steering angle axis input (e.g., $[-1, \ 1]$ from the axis corresponding to $[-0.5 \ rad, \ 0.5 \ rad]$ for the steering angle) to turn the steering motor. After that, if the linear velocity is not zero, we calculate the instantaneous turning radius of the vehicle from Eq. 1. Then, we choose the rear driving wheel that corresponds to the outer turning radius to be mapped from the joystick's velocity axis input (e.g. $[-1, \ 1]$ from the axis corresponding to $[-1 \ m/s, \ 1 \ m/s]$ for the linear velocity) based on the calculated instantaneous turning radius. Finally, the turning radius determines the instantaneous linear velocity of the other rear driving wheel from Eq. 6 and Eq. 7.

C. Agent SLAM overview

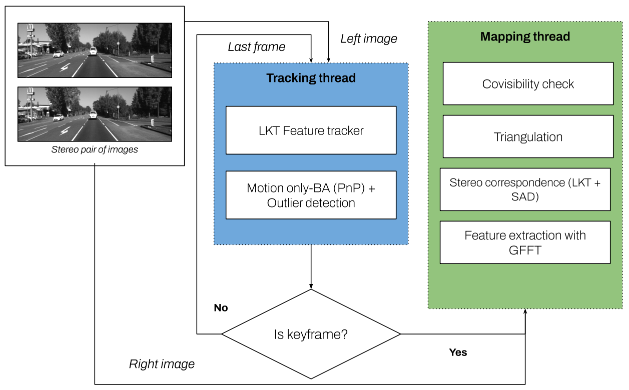


Figure 3: SLAM system architecture composed of a visual front-end that takes incoming images and search for correspondences with the last keyframe while the back-end process the keyframes by feature detection, stereo registration, landmark creation and covisibility check.

The proposed visual state estimation framework, which includes the visual front-end and back-end as two separate threads, is provided in this section and depicted in Fig. 3.

In the developed visual odometry pipeline, $\mathbf{T}^w_{c_k}, \mathbf{R}^w_{c_k}$ and $\mathbf{t}^w_{c_k}$ denote the pose, rotation, and translation of the k^{th} camera frame w.r.t the world frame w, set to be the first camera frame initialized at zero position and identity rotation. $\mathbf{z}^{\mathcal{I}_k}$ denotes detected image features of the k^{th} image frame, $\mathbf{l}^w_i$ represents the 3D position of the i^{th} landmark in the world frame, and d^{c_k} is the depth information of the k^{th} camera frame.

1) Preliminaries and problem statement: The aim is to estimate the current robot pose $\mathbf{T}^w_{c_k} \in \text{SE}(3)$ in the body frame c_k at camera frequency (at time instant $t_k = kT_s$ with the sampling time T_s) with respect to the world frame w, where the relative motion of the body could be introduced as the following compact form with the rotation $\mathbf{R}^w_{c_k} \in \mathbb{SO}(3)$ and translation $\mathbf{t}^w_{c_k} \in \mathbb{R}^3$ entries:

$$\mathbf{T}^w_{c_k} = \begin{bmatrix} \mathbf{R}^w_{c_k} & \mathbf{t}^w_{c_k} \\ 0 & 1 \end{bmatrix} \tag{8}$$

The front-end relies on the minimization of reprojection error between the detected image features $\mathbf{z}^{\mathcal{I}_k}_i = [u, v]^\top \in \mathbb{R}^2$ and 3D landmark points $\mathbf{l}^w_i = [x, y, z]^\top \in \mathbb{R}^3$ in consecutive timestamps to estimate the camera pose $\hat{\mathbf{T}}^w_{c_k}$ with the pinhole camera model denoted by $\pi(\cdot)$ [22]:

$$\hat{\mathbf{T}}^w_{c_k} = \underset{\mathbf{T}^w_{c_k}}{\arg\min} \sum_{i \in \mathcal{L}_k} ||\mathbf{z}^{\mathcal{I}_k}_i - \pi({\mathbf{T}^w_{c_k}}^{-1} \boxplus \mathbf{l}^w_i)||^2 \tag{9}$$

$$\hat{\mathbf{z}}^{\mathcal{I}_k}_i \triangleq \pi(\mathbf{l}^{c_k}_i) = \begin{bmatrix} \frac{x f_x}{z} + c_x \\ \frac{y f_y}{z} + c_y \end{bmatrix} \tag{10}$$

in which f_x, f_y are the horizontal and vertical focal lengths, and c_x, c_y are the horizontal and vertical principal points from the camera intrinsic parameters $\mathbf{K}$. The set $\mathcal{L}_k$ is the correspondence pair indices between features and landmarks estimated by tracking of keypoints or matching of descriptors for the camera frame c_k, and $\boxplus$ is the composite operator defined for SE(3) [23]. The features $\mathbf{z}^{\mathcal{I}_k}$ are the result of detecting distinctive sparse pixels of the image for motion tracking, such as corners, edges, or blobs [24]–[26]. The 3D landmark is a metric representation of the environment. It can be re-computed by the back projection of features $\mathbf{z}^{\mathcal{I}_k}_i$ from image plane $\mathcal{I}_k$ to the camera frame c_k using the inverse pinhole camera model $\pi(\cdot)^{-1}$ and depth value $d^{c_k}_i$ estimated by stereo triangulation.

$$\mathbf{l}^{c_k}_i = \begin{bmatrix} \frac{(u - c_x) d^{c_k}_i}{f_x} \\ \frac{(v - c_y) d^{c_k}_i}{f_y} \\ d^{c_k}_i \end{bmatrix} = \pi^{-1}(\mathbf{z}^{\mathcal{I}_k}_i, d^{c_k}_i) \tag{11}$$

2) Front-end: The front-end assumes that features have been detected in the last keyframe $\mathcal{K}_{j-1}$ and propagated till the last image $\mathcal{I}_{k-1}$. The thread tracks features to the current image by Lucas Kanade Tracker (LKT) [27] with the premise that only the source image needs features while the LKT will compute the ones for the target frame. As a means to detect outliers, i) we verify that the tracks do not report an LKT error above 3 pixels and ii) backtrack the current features $\mathbf{z}^{\mathcal{I}_k}$ to previous image $\mathcal{I}_{k-1}$ to discard the tracks with more than 0.5 error pixels. For each feature, we propagate the last associated 3D landmark till the track is lost due to the outlier filtering previously described. Using those 3D points and their respective 2D measurements in $\mathcal{I}_k$, we estimate the camera pose $\mathbf{T}^w_{c_k}$ at k^{th} time by a motion-only BA, an alternative formulation to Perspective-to-N-points (PnP) that could account for 3D points not necessarily on the previous frame c_{k-1} since they are represented w.r.t the world frame w. In contrast to regular BA, the motion-only variant does not optimize the 3D points, just the camera pose. The optimization discards outlier edges with χ^2 test by checking if the reprojection error of each measurement is lower than the respective threshold $\chi^2(\text{2DoF}, 95\%) = 5.955$ and removes it from the factor graph. This process is repeated 5 times as it was found experimentally that results converge to the optimal value while fulfilling the real-time requirements. On average, the front-end thread takes 10ms to finish in a 720p image. As the last step, the front-end decides if the frame should be selected as a keyframe and sent to the back-end if it fulfills two criteria: *i)* less than 50% of the points from last keyframe were successfully tracked, *ii)* the relative displacement is larger than 0.1m and if the median cosine of parallax angle between current frame and last keyframe is less than a threshold (e.g., 0.9998).

3) Back-end: The back-end is triggered when a new keyframe $\mathcal{K}_j$ is accepted or in initialization. The thread starts by taking the first image $\mathcal{I}_0$ as keyframe $\mathcal{K}_0$ and detecting features from Shi-Tomasi corners (GTTF). Suppose features were successfully propagated from previous frames through LKT to the new keyframe. In that case, we create a region of interest (ROI) as a rejection mask around each feature with a radius of 15 pixels. In that way, we keep the robust tracks for a more extended period, reduce runtime by limiting the detectable area, and distribute the feature evenly across the image. Next, the back-end searches for correspondences between stereo pairs of images in a similar way as in the front-end through relying on LKT but with the singularity that we initialize the feature location in the right image with either i) feature matched by a line-search using Sum-of-absolute differences or if the first try fails ii) we set it to the same location as the left feature. The initialization enables us to get solutions closer to the convergence basin, which translates into faster and more accurate estimations. After that, stereo features are triangulated using the calibration matrix $\mathbf{K}$ and stereo baseline b to 3D landmarks in camera frame $l_i^{c_k}$. At last, we search for more feature correspondences using the same criteria as the visibility check from [28]. To do so, we previously computed the BRIEF descriptor on all features, including those previously tracked, as they are appearance-dependent (i.e., pixel intensity of current image). If all steps were executed successfully, the back-end optimizes the last 10 keyframes in a fixed lag-smoother using Bundle Adjustment.

Bundle adjustment: BA is a non-linear least square problem originally formulated as Structure-from-Motion in the photogrammetry field. The problem estimates jointly the camera poses $\mathcal{T}$ and landmarks $\mathcal{L}$ by minimizing the reprojection error $e_{(c_k,l)}$ using the Mahalanobis distance with information matrix $\mathbf{\Sigma} = \mathbf{I}_{2\times2}$.

$$\begin{aligned}\{\hat{\mathcal{T}}, \hat{\mathcal{L}}\} &= \arg\min \sum_{c_k\in\mathcal{T}} \sum_{i\in\mathcal{L}_k} \rho(e_{(c_k,l)}), \\ e_{(c_k,i)} &= ||\mathbf{z}_i^{\mathcal{I}_k} - \pi(\mathbf{T}_{c_k}^w \boxminus \mathbf{l}_i^w)||^2_{\mathbf{\Sigma}},\end{aligned} \tag{12}$$

where $\boxminus$ is the retraction operator defined for SE(3). The general purpose Ceres optimizer [29] is selected as the solver utilizing the Levenberg–Marquardt algorithm, suitable for non-linear least square problems, and Cholesky Decomposition to leverage Hessian sparsity and speed up the process. Following 10 iterations, the back-end stops the optimization and discards any edge with an error that does not fulfill the χ^2 test. Five more iterations are run after this filtering. After optimization, outliers are classified again, and if detected, the observation is removed from its corresponding landmarks from the $\mathcal{L}$ map.

D. Centralized server

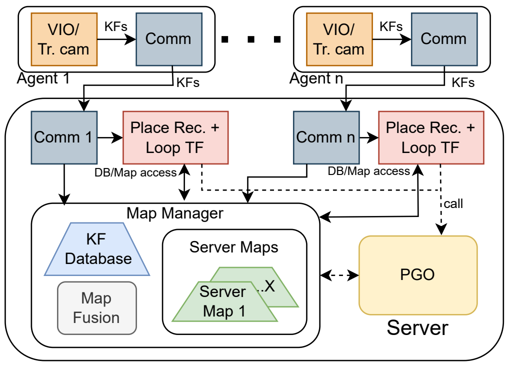


Figure 4: COVINS-G [1] centralized server which is charge of fusing the SLAM results from each agent into a unified map.

We employed COVINS-G [1] (see Fig.4) as a centralized server that receives incoming keyframes from the visual SLAM of each agent running onboard. COVINS-G is a generalized version of COVINS [13] that builds upon a front-end agnostic sever by only relying on keyframes and images. Thus, different kinds of SLAM solutions (feature-based, dense-based, or learned-based) could map the environment jointly. Although this framework is less arcuate than its former version, our integration with front-end SLAM is seamless and requires minimal to no code modifications as it only requires publishing the synchronized image-keyframe pair.

The agents are communicated from a peer-to-peer (p2p) communication protocol, and information is sent in batches to maintain consistent sending frequencies. The server receives the keyframe pose and the image and detects a new set of ORB features over the image to use later for place recognition and loop closing detection. To perform it, keyframes are saved in a database, and any loops are searched between the database and a query (new) keyframe by the Bag-of-words [30] algorithm fed from the ORB features [26]. Since the server does not store 3D landmarks, standard algorithms to estimate relative pose in a loop such as 3D-2D Ransac (as in VINS-Mono [31] or ORB-SLAM 3 [32]) are unfeasible. The 5-point algorithm (2D-2D RANSAC) will not solve the issue since it estimates the relative pose up to a scale factor. Instead, COVINS-G proposed leveraging several neighborhood keyframes between the candidate from the database and the query to form a multi-camera system (within the agent trajectory or between two agents) and infer the pose using the 17-point algorithm. The method leverages 17 2D-2D correspondences through the generalized epipolar constraint, an extension of the epipolar constraint without a central projection point. The server wraps this inference with a RANSAC scheme to detect outliers and retrieve the most extensive set of inliers with its corresponding pose. If 100 inliers were found at least, the candidate pair of keyframes is accepted and sent as a constraint to the Pose Graph Optimization (PGO) for path refinement.

1) Pose Graph Optimization: PGO could be considered a special case of BA without optimizable landmarks, drastically reducing its computational complexity and runtime.

$$\begin{aligned}\{\hat{\mathcal{A}}\} &= \arg\min \sum_{(i,j)\in\mathcal{O}} e_{(i,j)} + \sum_{(k,l)\in\mathcal{C}} \rho(e_{(k,l)}), \\ e_{(1,2)} &= ||\log(\mathbf{T}_1^2 \boxplus \mathbf{T}_2^w \boxminus \mathbf{T}_1^w)||^2_{\mathbf{\Sigma}},\end{aligned} \tag{13}$$

where $\mathcal{A} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_M\}$ is the joint set of agent keyframe poses, $\mathcal{O}$ is the set of odometry measurements as edges that connect two keyframes and $\mathcal{C}$ is the set of loop candidates. The log operator defines a Euclidean error for SE(3), and we wrap the loop closures with a robust function for outlier detection, such as in BA. This optimization solver is solved using the same settings as the BA, including the Ceres solver, sparsity handling, and number of iterations. When the optimization is completed, the map of all agents is merged into a single one, which is aligned with the first agent map.

IV. Experimental results

A. Setup

We equipped the RC Car with a sensor suite composed of one stereo ZED Mini camera, which provides a 30 Hz stream

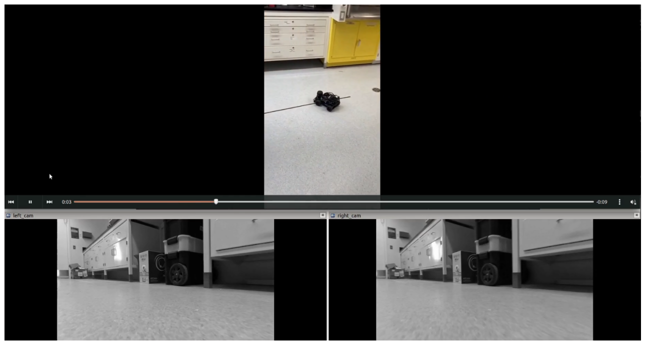


Figure 5: Three views of the RC Car including top view taken from an external camera, left and right onboard camera views.

of 1280×720p image with built-in 9-DoF IMU and one ROS-compatible 2D rotary LiDAR that outputs laser scans at 10 Hz. Fig. 5 shows a synchronized camera stream from the left and right lenses. The ZED ROS driver is in charge of calibrating cameras, delivering those settings to the camera configuration memory, and synchronizing and rectifying the stereo views to allow finding stereo correspondences in a line search. That visual stream is sent to the onboard SLAM to provide keyframes to the COVINS-G server.

Since we do not have access to a Motion Capture System for beacon tracking nor a 3D LiDAR for high-precision localization, we compute the ground-truth trajectory from the 2D LiDAR scans with Hector SLAM [33]. Hector SLAM leverages an occupancy grid as a discrete map from which LiDAR scans are matched and then fed into an optimization problem to infer the robot pose. Since occupancy grids have a limited resolution to save memory footprint, the authors proposed an interpolation method to achieve sub-grid cell accuracy. We need to synchronize the LiDAR and camera estimations for error computation and align their trajectories since they use different frame conventions (see Fig.7). To synchronize them, we employed the ROS Time Synchronizer now as `Approximate Message filter` that matches sensor messages when their timestamps are close enough within a margin of 5ms. Then, camera poses $\mathbf{T}^{w}_{c_k}$ are transform to the LiDAR frame $\mathbf{T}^{\mathrm{L}_0}_{\mathrm{L}_k}$:

$$\mathbf{T}^{\mathrm{L}_0}_{\mathrm{L}_k} = \mathbf{T}^{\mathrm{L}}_{c}\mathbf{T}^{w}_{c_k}(\mathbf{T}^{\mathrm{L}}_{c})^{-1} \tag{14}$$

where $\mathbf{T}^{l}_{c}$ is the extrinsic matrix between the LiDAR and camera frames, measured previously while setting up the sensors.

B. Localization accuracy

Trajectory estimation accuracy is estimated through two metrics: i) Absolute Trajectory Error (ATE) and ii) Relative trajectory Error (RTE). While the former measures the trajectory offset at each pose and is heavily influenced by global corrections such as Full Bundle Adjustment or Loop Closing with Pose Graph Optimization, the latter measures deviations in terms of relative transformations (which could be interpreted as velocity in a differential form) and its precision is coupled with the features tracking and Motion-Only BA. Generally, the APE is an accumulated measure of RPE and exhibits drifts as time progresses.

$$\mathrm{ATE}_i = ||\mathbf{E}^{-1}_{\mathrm{est},i}\mathbf{E}_{\mathrm{gt},i} - \mathbf{I}_{4\times4}||_2 \tag{15}$$
$$\mathrm{RTE}_{i,j} = ||(\mathbf{E}^{-1}_{\mathrm{gt},i}\mathbf{E}_{\mathrm{gt},j})^{-1}(\mathbf{E}^{-1}_{\mathrm{est},i}\mathbf{E}_{\mathrm{est},j}) - \mathbf{I}_{4\times4}||_2 \tag{16}$$

Specific error measures from translation and rotation could be extracted from ATE and RTE by matrix slicing in case of translation, and for rotation, you must parametrize the error rotation $\mathbf{E_r}$ by the tangent space of $\mathrm{SO}(3)$ known as Lie Algebra ($\mathrm{so}(3)$). We compute and plot both errors using the Evo package [34].

We compared two trajectories recorded over the same localization and simulated two different agents. We processed them sequentially until the COVINS-G found a loop closure between them and closed it with PGO (see Fig.6). However, they could also be processed in parallel if we had two RC cars. Tabs. I and II report the error statistics (Min., RMSE, Max.) before and after the map merging in the centralized server. While RPE slightly increases and rotation APE does not change drastically, the translation APE gets a reduction of 31% **for Agent 1#** and 36% **for Agent 2#**. This happens because PGO only enhances the APE performance while RPE could be manipulated towards minimizing the APE, even if that means increasing the RMSE RPE. Regarding the rotation part, since the dominant motion component is the translation (you can think of the trajectory as a rectangle with four 90-degree rotations), the excitation level is not enough for the PGO to minimize it jointly with the translation.

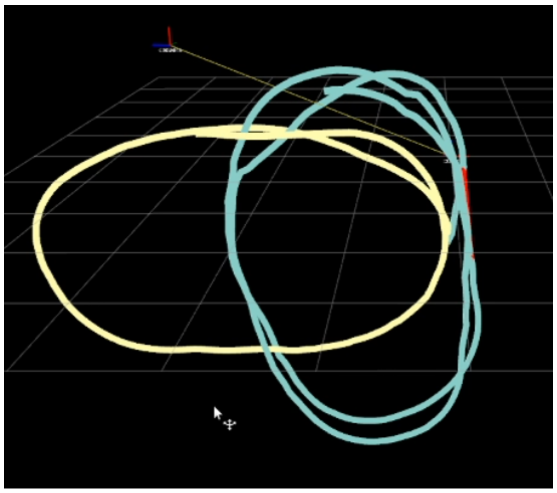
Table I: Trajectory error of agents before map fusion in centralized server

Agent		**APE**$_\mathrm{t}$[m]	**RPE**$_\mathrm{t}$[m]	**APE**$_\mathrm{R}$[rad]	**RPE**$_\mathrm{R}$[rad]
I	Min.	0.024	0.001	0.077	0.001
	RMSE	0.195	0.015	0.181	0.010
	Max.	0.335	0.109	0.258	0.043
II	Min.	0.036	0.001	0.144	0.001
	RMSE	0.118	0.014	0.179	0.010
	Max.	0.221	0.049	0.227	0.043

Table II: Trajectory error of agents after map fusion in centralized server

Agent		**APE**$_\mathrm{t}$[m]	**RPE**$_\mathrm{t}$[m]	**APE**$_\mathrm{R}$[rad]	**RPE**$_\mathrm{R}$[rad]
I	Min.	0.012	0.008	0.083	0.002
	RMSE	0.137	0.032	0.176	0.015
	Max.	0.322	0.096	0.249	0.061
II	Min.	0.012	0.011	0.137	0.002
	RMSE	0.075	0.028	0.168	0.013
	Max.	0.176	0.058	0.219	0.030

Fig.9 depicts how the APE varies along each agent path. Their error is homogeneously distributed along the trajectory. It increases with a slow phase till it reaches its maximum values at the end of the sequence, as expected since it is the point where the most drift is accumulated. More important than just the error and drift, it is worth noting that the


(a)

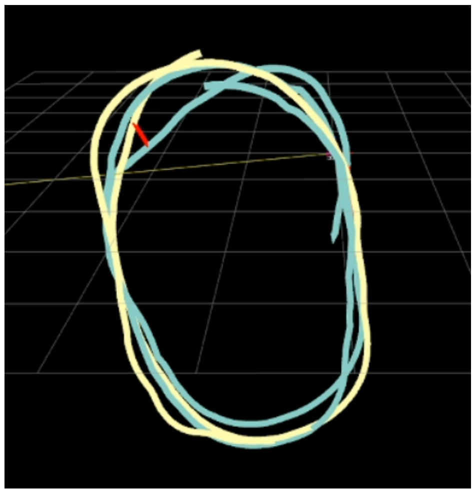

(b)

Figure 6: **Rviz visualization from COVINS-G backend**: Trajectories of the first agent and second agent have loop closures between each other as red rods. Loop closures could also be set within a single agent for drift reduction. a) Before map merging, b) After map merging

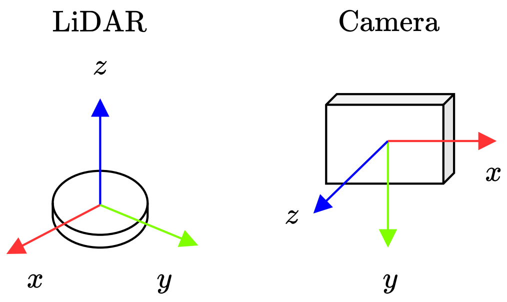


Figure 7: Coordinate frames involved in the sensor setup

COVINS-G aligns the trajectory of different initial poses into a single one (see Fig8) on real-time such as the collaborative navigation tasks rely less on post-run processing routines. We suspect that error results could be enhanced further if 3D landmarks are included in a collaborative BA instead of just PGO, but with the cost of losing flexibility for any on-board SLAM.

C. Runtime analysis

The on-board system was crafted to guarantee real-time performance, keeping in mind reducing search space during feature tracking and only detecting features for keyframes. Fig.10 depicts the run-time histograms of Agent 1 and 2 while processing their trajectories. They have two modes, one localized within 0.01 and 0.05s associated with feature tracking with pose estimation and the other one spread across 0.08 to 0.015s, which happens when a new keyframe is created, and local BA is executed to refine the moving window of poses. Unfortunately, COVINS-G does not provide a tool to report PGO or Place recognition run-time. However, experiments have shown that it takes approximately 400-1000ms both procedures to be completed.

V. Conclusions

In this work, we have presented the implementation of a collaborative visual SLAM algorithm composed of a custom designed on-board stereo visual SLAM that leverages sparse feature detection and fast search correspondence by Lucas Kannade and a centralized server (COVINS-G) with a pair of keyframe-image It is capable of fusing trajectories from different agents by place recognition and pose graph optimization. To test it, we designed a multi-purpose Ackermann steering RC car with a low-level processing for driving and steering motor control with an ESP32 and a high-level processing in a Jetson Nano Super that communicates to the low-level and receives sensor stream and joystick commands. We compared our trajectory estimation against ground truth generated by Hector SLAM with 2D LiDAR scans and reported a performance boost of $\sim$30% for the translation APE. At the same time, the rest of the metrics remain within the same value. Our system can run in real time and align the trajectory of several agents in parallel without extensive code changes on the onboard stereo SLAM. As for future works, we seek to run experiments including the landmarks in the agent map fusion to determine how considerable their influence is and if their addition is worth sacrificing flexibility in the framework.

References

[1] M. Patel, M. Karrer, P. Bänninger, and M. Chli, “Covins-g: A generic back-end for collaborative visual-inertial slam,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2076–2082, 2023.
[2] Y. Zhou, J. Xiao, Y. Zhou, and G. Loianno, “Multi-robot collaborative perception with graph neural networks,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2289–2296, 2022.

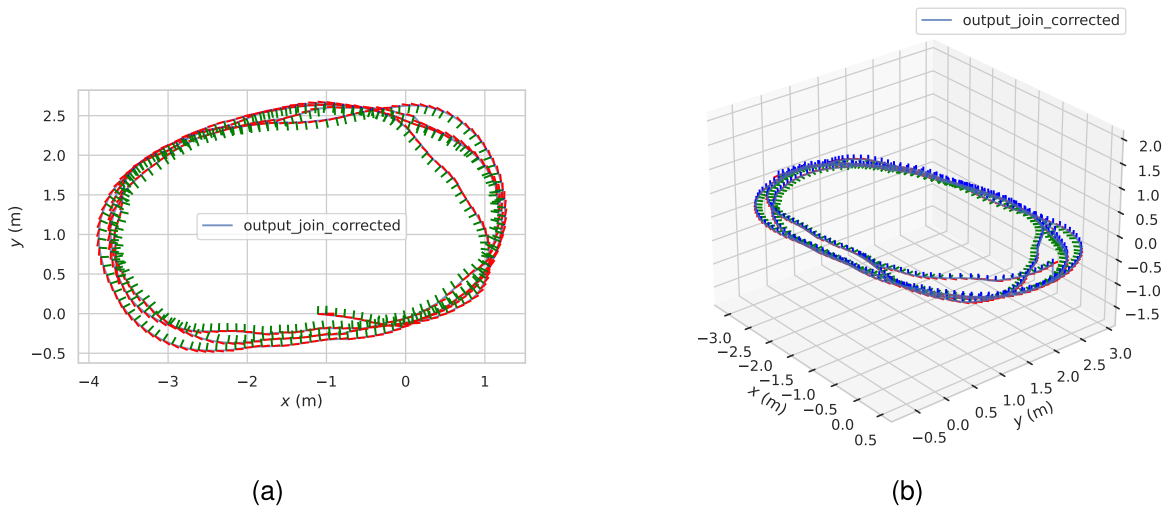


Figure 8: **Output trajectory after fusing the agent paths into a single session**: a) 2D-view, b) 3D-view. Noted that the fused trajectory has minimal drift on the z axis due to the loop closing.

[3] H. Zhao, B. Ivanovic, and N. Mehr, "Distributed nerf learning for collaborative multi-robot perception," *arXiv preprint arXiv:2409.20289*, 2024.

[4] R. Murai, J. Ortiz, S. Saeedi, P. H. Kelly, and A. J. Davison, "A robot web for distributed many-device localisation," *IEEE Transactions on Robotics*, 2023.

[5] G. S. Martins, D. Portugal, and R. P. Rocha, *mrgs: A Multi-Robot SLAM Framework for ROS with Efficient Information Sharing*, pp. 45–75. Cham: Springer International Publishing, 2021.

[6] H. Ebel, "Distributed control and organization of communicating mobile robots: design, simulation, and experimentation," 2021.

[7] X. Shao, J. Zhang, and W. Zhang, "Distributed cooperative surrounding control for mobile robots with uncertainties and aperiodic sampling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 18951–18961, 2022.

[8] W. Xiao, Q. Zhou, Y. Liu, H. Li, and R. Lu, "Distributed reinforcement learning containment control for multiple nonholonomic mobile robots," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 2, pp. 896–907, 2021.

[9] J. Hu, P. Bhowmick, and A. Lanzon, "Group coordinated control of networked mobile robots with applications to object transportation," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8269–8274, 2021.

[10] X. Zhou, W. Liang, I. Kevin, K. Wang, Z. Yan, L. T. Yang, W. Wei, J. Ma, and Q. Jin, "Decentralized p2p federated learning for privacy-preserving and resilient mobile robotic systems," *IEEE Wireless Communications*, vol. 30, no. 2, pp. 82–89, 2023.

[11] M. Moniruzzaman, A. Rassau, D. Chai, and S. M. S. Islam, "Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey," *Robotics and Autonomous Systems*, vol. 150, p. 103973, 2022.

[12] M. Karrer, P. Schmuck, and M. Chli, "Cvi-slam—collaborative visual-inertial slam," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, 2018.

[13] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli, "Covins: Visual-inertial slam for centralized collaboration," 2021.

[14] Y. Chen, H. Inaltekin, and M. Gorlatova, "Adaptslam: Edge-assisted adaptive slam with resource constraints via uncertainty minimization," 2023.

[15] Z. Jiang and Y. Shan, "Cmd-slam: A fast low-bandwidth centralized multi-robot direct stereo slam," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, pp. 920–926, 2024.

[16] Y. Tian, Y. Chang, F. H. Arias, C. Nieto-Granda, J. P. How, and L. Carlone, "Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems," 2021.

[17] C. Li, P. Yi, G. Guo, and Y. Hong, "Distributed pose-graph optimization with multi-level partitioning for collaborative slam," 2024.

[18] V. Yugay, T. Gevers, and M. R. Oswald, "Magic-slam: Multi-agent gaussian globally consistent slam," 2024.

[19] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2024.

[20] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "Door-slam: Distributed, online, and outlier resilient slam for robotic teams," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1656–1663, 2020.

[21] P.-Y. Lajoie and G. Beltrame, "Swarm-slam: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems," *IEEE Robotics and Automation Letters*, vol. 9, p. 475–482, Jan. 2024.

[22] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 3 2004.

[23] J. L. Blanco-Claraco, "A tutorial on $\mathbf{SE}(3)$ transformation parameterizations and on-manifold optimization," 2022.

[24] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[25] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 11 2004.

[26] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[27] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), p. 674–679, Morgan Kaufmann Publishers Inc., 1981.

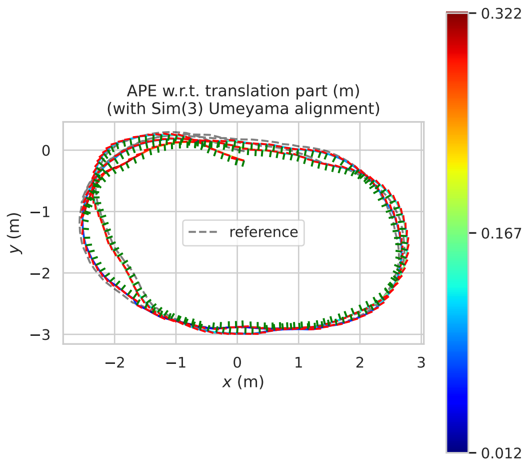
[28] M. Contreras, N. P. Bhatt, and E. Hashemi, "Dynanav-svo: Dynamic stereo visual odometry with semantic-aware perception for autonomous navigation," *IEEE Transactions on Intelligent Vehicles*, 2024.

[29] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 10 2023.

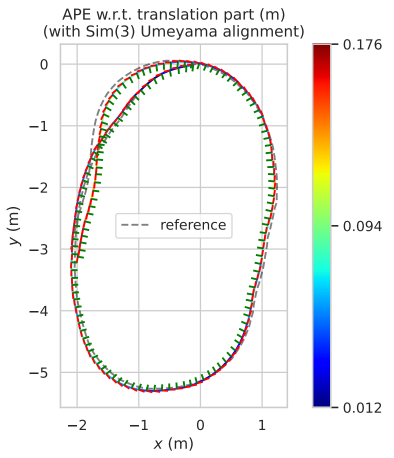
[30] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[31] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[32] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.



(a)



(b)

Figure 9: **Optimized trajectory after alignment with the ground-truth trajectory**: a) Agent I, b) Agent II. The heatmap represents the APE of translation.

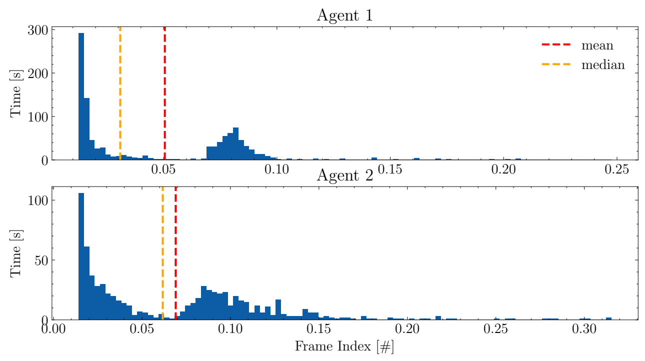


Figure 10: Execution time histogram of two agents. Mean and median have been highlighted for better understanding. The distributions have two well located modes: 1) close to 0s corresponding to just tracking and 2) in 0.075s corresponding to tracking + mapping + BA.

[33] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, November 2011.

[34] M. Grupp, "evo: Python package for the evaluation of odometry and slam.." https://github.com/MichaelGrupp/evo, 2017.