ColSLAM: A Versatile Collaborative SLAM System for Mobile Phones Using Point-Line Features and Map Caching

Wanting Li
School of Information, Renmin University of China
Beijing, China
lindalee@ruc.edu.cn

Yongcai Wang*
School of Information, Renmin University of China
Beijing, China
ycw@ruc.edu.cn

Yongyu Guo
School of Information, Renmin University of China
Beijing, China
yongyuguo@ruc.edu.cn

Shuo Wang
School of Information, Renmin University of China
Beijing, China
shuowang18@ruc.edu.cn

Yu Shao
School of Information, Renmin University of China
Beijing, China
1428040443@qq.com

Xuewei Bai
School of Information, Renmin University of China
Beijing, China
bai_xuewei@ruc.edu.cn

Xudong Cai
School of Information, Renmin University of China
Beijing, China
xudongcai@ruc.edu.cn

Qiang Ye
Faculty of Computer Science, Dalhousie University
Halifax, Canada
qye@cs.dal.ca

Deying Li
School of Information, Renmin University of China
Beijing, China
deyingli@ruc.edu.cn

ABSTRACT

Over the past years, augmented reality (AR) based on mobile phones has gained great attention. When multiple phones are used in AR applications, collaborative simultaneous localization and mapping (SLAM) is considered one of the enabling technologies, i.e., multiple mobile phones complete the localization and mapping through collaboration. However, the state-of-the-art collaborative SLAM systems not only suffer from the delays introduced by a high-complexity graph optimization problem, but also may exhibit varying levels of accuracy across dissimilar environments or different types of mobile devices. In this paper, we propose a scalable and robust collaborative SLAM system, point-line-based Collaborative SLAM (ColSLAM). Technically, ColSLAM includes two innovative features that help achieve satisfactory scalability and robustness. First, a mapping cacher (MC) is designed for each agent on the server, which uses global keyframes to detect loop closures, updates the cached local map, and quickly responds to the agent's pose drifts. With MC, each agent's local pose is corrected using global knowledge in real-time. Secondly, to improve the robustness performance, ColSLAM employs point-line-fusion-based Visual Inertial Odometry (VIO), point-line-fusion-based NetVLAD loop detection, and an enhanced geometric verification and relative pose calculation method called PNPL. Empirical evaluations based on the EuRoc dataset and real degenerate environments demonstrate that ColSLAM outperforms the existing collaborative SLAM systems in terms of accuracy, robustness, and scalability.

CCS CONCEPTS

• **Human-centered computing** → **Mobile phones**; **Visual analytics**; • **Computing methodologies** → **Tracking**; **Reconstruction**; *Vision for robotics.*

KEYWORDS

Augmented Reality, Mobile Phones, Multi-agent, Collaborative SLAM, Map Caching, Point-line Features

ACM Reference Format:
Wanting Li, Yongcai Wang, Yongyu Guo, Shuo Wang, Yu Shao, Xuewei Bai, Xudong Cai, Qiang Ye, and Deying Li. 2023. ColSLAM: A Versatile Collaborative SLAM System for Mobile Phones Using Point-Line Features and Map Caching. In *Proceedings of the 31st ACM International Conference on Multimedia (MM '23), October 29–November 3, 2023, Ottawa, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3581783.3611995

1 INTRODUCTION

Augmented reality (AR) based on mobile phones has gained great attention for its potential application in smart navigation, intelligent factories, and immersive games[1]. When multiple phones are used in AR applications, collaborative simultaneous localization and mapping (SLAM)[2] is considered one of the fundamental supporting techniques. Collaborative SLAM aims to locate mobile phones accurately and in real-time, while constructing and maintaining the point-cloud map of the environment by fusing the multi-view image sequences collected by the phones. Technically, although collaborative SLAM is highly promising, it still faces serious challenges.

*Corresponding author

MM '23, October 29–November 3, 2023, Ottawa, ON, Canada.

ACM ISBN 979-8-4007-0108-5/23/10…$15.00
https://doi.org/10.1145/3581783.3611995

So far, there have been a number of successful attempts of building collaborative SLAM systems [3–5]. However, scalability and robustness remain to be the key barriers. The reason is that the computing devices involved in the systems have constrained computation capability and local errors could be exaggerated in the global optimization process. Note that existing collaborative SLAM systems generally adopt an agent-server architecture, where mobile phones work as agents. Each mobile phone carries out Visual Inertial Odometry (VIO) [6–8], and reports local trajectories and key frames to the server. The server conducts cross-device loop closure detection [9, 10] and solves a large-scale global graph optimization problem. It registers the local maps and sends the corrected poses to the agents to adjust their local drifts. As time goes by, the scale of the server's optimization problem continuously increases, making it more and more difficult for the server to correct the local drifts of the agents in real-time, even if only several mobile phones are involved. Meanwhile, in textureless environments, when an agent provides a poor VIO result or when the server conducts noisy loop closure detection, the accuracy of the collaborative SLAM system will significantly deteriorate due to error propagation in the global optimization process.

To address the scalability and robustness problem, we propose a versatile collaborative SLAM system, point-line-based Collaborative SLAM (ColSLAM). With ColSLAM, a map caching scheme is used to decouple the cached sub-graph (CSG) optimization process and the global map optimization process. In addition, a novel two-stage asynchronous optimization method is employed to return agent keyframe drifts in real-time, and send agent local map drift once map fusion is completed. This approach enhances mapping accuracy and reduces communication network bandwidth while satisfying real-time requirement. To improve the robustness performance, ColSLAM utilizes both point and line features extracted from the collected images, which is completely different than the original point-based SLAM framework. Line features are rich in man-made environments, which are potential to mitigate pose drifting when point features are insufficient. The main contributions of this paper can be summarized as follows:

- **Map caching**: ColSLAM includes a two-stage asynchronous optimization method for cached sub-graph (CSG) optimization and global map optimization. This method ensures real-time responses to agents while taking accuracy and communication efficiency into consideration, which significantly improves the scalability of collaborative SLAM.
- **Robust point-line features**: ColSLAM redesigns the whole collaborative SLAM framework to utilize point-line features. Technically, ColSLAM includes point-line-based VIO on the agents and the point-line fusion-based NetVLAD method to conduct loop detection on the server. Robust features are indispensable for multi-agent collaboration.
- **Collaborative loop detection**: With ColSLAM, we use an enhanced geometric verification and relative pose calculation method, called PNPL, to further improve the robustness and accuracy of the system.

The rest of the paper is organized as follows. We first discuss the related work in Section 2. An overview of ColSLAM is provided in Section 3. The details of the agent-side and server-side schemes are presented in Sections 4 and 5, respectively. Our experimental results are described in Section 6. Finally, Section 7 includes our conclusions.

2 RELATED WORK

Visual-Inertial SLAM (VI-SLAM) is a process that combines visual and inertial measurements to simultaneously estimate the motion of a camera and to create the map of the environment simultaneously. In this section, we present the related work on single-agent visual-inertial SLAM and multi-agent collaborative SLAM.

2.1 Single-agent Visual-Inertial SLAM

The mainstream methods for VI-SLAM generally conduct feature point extraction and matching in the front-end for visual odometry, and use EKF (Extended Kalman Filter) or graph optimization in the back-end for loop detection, global correction, and mapping. Representative works include MSEKF[12] and OpenVINS[13] using EKF backend, and VINS-Mono[6] and ORB-SLAM3[14] which use graph optimization backend. Although the point-based feature extraction and matching can generally provide accurate localization in most scenarios, key remained challenges for VI-SLAM are how to improve robustness in degeneracy environments with weak textures. Researchers have explored the integration of point and line features in VI-SLAM. Line features can provide more robust estimates in weak texture environments and help to perform better matching. Representative works include PL-VIO[15], PL-VINS[16] and IDLL[17].

But single-agent SLAM still has limitations in terms of limited perception range, mapping efficiency and accuracy. To overcome the limitations of single-agent SLAM, researchers investigate multi-agent collaborative SLAM, which utilize information exchange and collaboration of multiple agents to improve the accuracy and reliability of the SLAM system.

2.2 Multi-agent Collaborative SLAM

Collaborative SLAM can be further divided into centralized collaborative SLAM and distributed collaborative SLAM. In centralized SLAM, the agents conduct low-cost and real-time local VIO, and the computationally expensive tasks are typically executed on the server.

Early works propose fundamental methods. PTAMM [18] allows maps of many environments to be constructed in parallel and loop closure can be conducted across maps. CoSLAM [19] introduce inter-camera pose estimation and inter-camera mapping to deal with dynamic objects in the localization and mapping process. Multi-robot CoSLAM [20] proposes to track a mobile target via multi-robot collaborative SLAM. C^2TAM [21] proposes a cloud-based framework for cooperative mapping and tracking. They also investigate the accuracy and bandwidth balance.

Recent works refine the collaboration methods and improve the system performances. CVI-SLAM [22] and CCM-SLAM [23] propose agent-server collaborative SLAM architecture. They compute short-term visual odometry and local map on the robot. Then the robot sends keyframes and map points to the server. The server performs loop closure detection, map fusion, and global graph optimization. COVINS [24] uses VINS [6] framework on the agent

side and the server has the similar role. CORB2I-SLAM [25] enhances agent adaptivity by evaluating VO pose consistency. When the robot's local odometry tracking fails, the robot side can be re-initialized using the sub-map sent by the server to improve relocation ability.

For collaborative SLAM using mobile phones, Liu et al. propose a SLAM system called RAMA-SLAM[26]. In this system, smartphones use visual and IMU data to run monocular visual inertial odometry and transmit data wirelessly to a server. RAMA-SLAM is considered the most balanced method in terms of communication efficiency and accuracy. It chooses not to transmit the matching points, performs local BA on the server side, and only transmits offsets to the agents. The system in [27] also allows the server to send part of the server's historical information to the mobile phone side so that the phone can search for loop closures from historical information and use factor graph optimization to correct the current odometry. SwarmMap[28] addresses scalability challenges caused by data explosion with growing numbers of agents through edge offloading. The priority task scheduler and change-based server-agent synchronization mechanism in SwarmMap determine which tasks are transmitted to the agent preferentially, and only update tasks that have changed to reduce data transmission and processing overhead.

On the other branch, distributed collaborative SLAM methods execute the localization and mapping algorithm in a distributed manner, without a centralized server, which are gaining increasing attentions [29][30][31][32][33]. However, they require high computing capabilities of the nodes, and the data synchronization and fusion are more complex. More comparisons on distributed collaborative SLAM methods can be referred to a recent survey [34].

We focus on the agent-server based collaborative SLAM. The problem of delay increasing with the size of global graph and the unreliability problem in degeneracy environments are still key challenges for centralized collaborative SLAM. To overcome these challenges, this papers proposes a map caching scheme for the server to quickly response to the agents by using a two-stage asynchronous optimization method, which decouples the fast response and the global optimization. We also exploit point-line fusion to redesign collaborative SLAM in different aspects to improve the robustness in degeneracy environments.

3 OVERVIEW OF COLSLAM

In this section, we present the system overview of ColSLAM. The overall architecture of ColSLAM is shown in Fig. 1. It can be divided into the agent and the server tasks. On the agent, each agent performs short-term VIO; uploads local poses and point-line features to the server; and accepts pose drifts from the server to correct local poses. The server conducts computationally intensive tasks such as multi-agent loop closure detection, map fusion, and drift responding to the agents.

Agent task: The agent collects IMU and camera data from own sensors and performs IMU pre-integration, point and line feature extraction, and triangulation to estimate the feature depth (only in initialization). After initialization, the agent poses and features are optimized through bundle adjustment in a sliding window composed of N keyframes. The pose of an agent calculated at the kth keyframe is denoted by $\mathbf{X}_k = (p_k^T, q_k^T, v_k^T)^T \in \mathbb{R}^{10}$, where $p_k \in \mathbb{R}^3, q_k \in \mathbb{R}^4, v_k \in \mathbb{R}^3$ represent the position, rotation, and velocity of the agent respectively. The i-th 3D point feature and the j-th 3D line feature in the kth frame are denoted by $\mathbf{P}_i^k = (u_i, v_i, \lambda_i)^T \in \mathbb{R}^3$ and $\mathcal{L}_j^k = (u_{s_j}, v_{s_j}, \lambda_{s_j}, u_{e_j}, v_{e_j}, \lambda_{e_j})^T \in \mathbb{R}^6$, where λ_i is the inverse depth of the ith point. s_j and e_j denote the two end points characterizing a line segment.

Local BA is executed in a sliding window of N keyframes, which ensures the computational efficiency on the mobile phone. Each agent can also calculate poses and local map by itself in real-time, even when it is not connected to the server. The agent keeps transmitting the newest pose $\mathbf{X}^N$ and the newest feature $\mathbf{P}^N$ and $\mathcal{L}^N$ of the N-th keyframe in the sliding window to the server for multi-agent global map fusion. Meanwhile, the agent also receives two kinds of drift corrections from the server for local pose correction. (1) *keyframe drift correction* ($R_{\text{diff}}^k \in \mathbb{R}^{3\times3}, \mathbf{t}_{\text{diff}}^k \in \mathbb{R}^3$, for $k \in \{1, \cdots, N\}$), which are the fast returned drifts of the N poses in the sliding window, responded by the cached map thread of the server. (2) *local map drift correction* ($R_{\text{w}} \in \mathbb{R}^{3\times3}, \mathbf{t}_{\text{w}} \in \mathbb{R}^3$) which are slowly responded from the global optimization thread of the server. It transforms the whole local map. These responses correct local drifts by collaborative information on the server.

Server task: On the server end, the server maintains a fixed size, *cached sub-graph (CSG)* for each agent. The CSG includes the poses $\mathbf{X}$, point-and-line features $\mathbf{P}$, $\mathcal{L}$ of a N-keyframe sliding window, which are all received from the agent, and convert them to the global map coordinate system π_w by agent's local map drift $(R_w^i, \mathbf{t}_w^i)$. The difference from the agent is that the loop closure edges based on cross-agent, point-line based loop closure detection are added into the CSG, and the CSG is in global frame. BA on the CSG is calculated in real-time when new key-frame arrives. Then the corrections, i.e., $\{(R_{\text{diff}}^k, \mathbf{t}_{\text{diff}}^k), k \in 1, \cdots, N\}$ are sent back to the agent in real-time to correct the local drifts when new key-frame arrives. Meanwhile, the server runs global map fusion in another thread to detect and fuses the maps of multiple agents asynchronously. This enables the server to generate a comprehensive map of the entire environment and give back the i-th agent's local map drift $(R_w^i, \mathbf{t}_w^i)$.

Note that more details about the agent-side process will be given in Section 4. In addition, the specifics of the server-side modules will be discussed in Section 5.

4 AGENT-SIDE MODULE OF COLSLAM

Each agent conducts local point-line based VIO in a sliding window with a fixed number of N keyframes. It then packs the information of the newest keyframe of the sliding window, including its pose, around 200 point features, and 100 line features of the N-th keyframe and transmits to the server by ROS bridge. Meanwhile the agent accepts *keyframe drift correction* fast returned from the server and receives *local map drift correction* asynchronously returned from the server. By adjusting local poses and map using these correction information, each agent achieves accurate local trajectory and corrected local mapping results using the server knowledge, while using low communication costs.

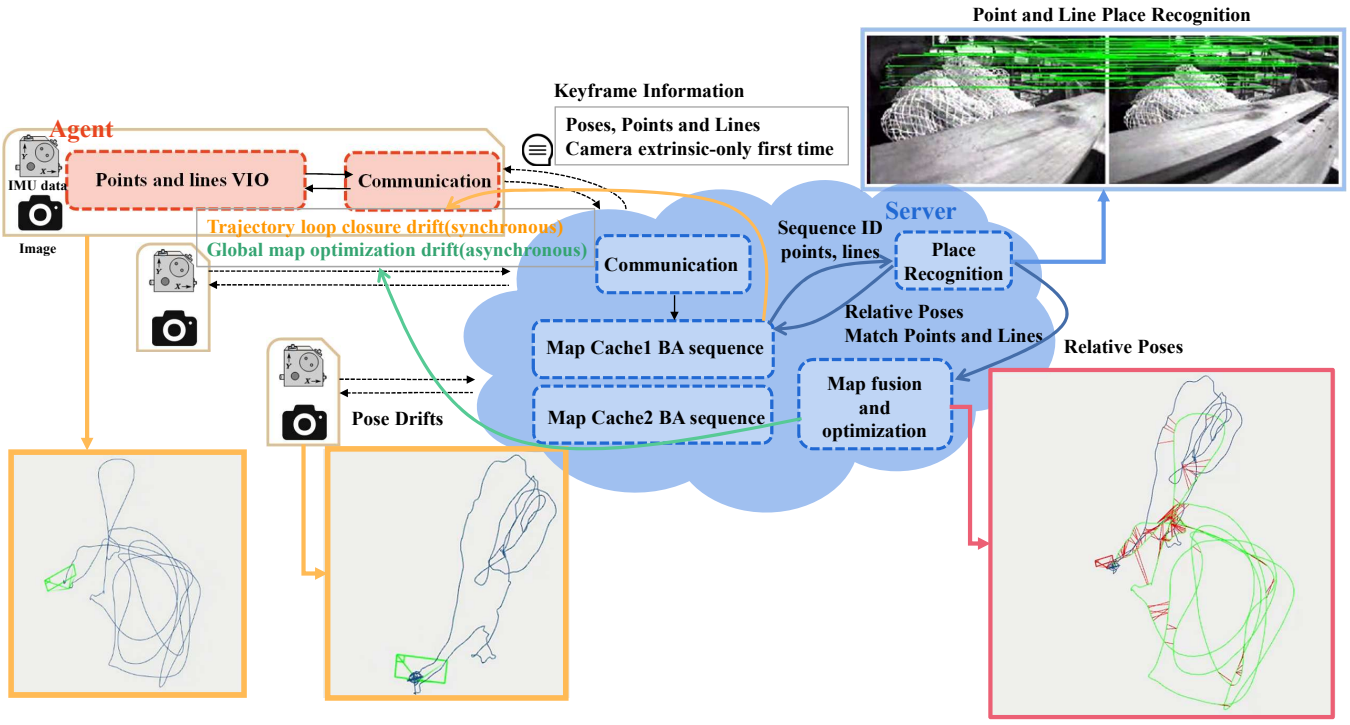

Figure 1: Overview of ColSLAM. Each agent uses its own sensor information to perform points and lines VIO and then transmits the results to the server through a communication module. The server stores the information of the agent through map caching. The server performs place recognition, map fusion, and optimization based on point and line features. Then, the pose drifts are transmitted to the agents for correcting the agents' local drifts.

4.1 Point-line based VIO and Local Mapping

Point-line based VIO is developed for the agent side, which consists of two threads. The first thread extracts data from the camera and IMU sensors, and employs Shi-Tomasi[35] and KLT[36] algorithms to detect and track point features, as well as LSD detector[37] and LBD descriptor[38] by KNN[39] to identify and track line features. Inliers are found using RANSAC-based epipolar geometry constraint[40]. The IMU information is pre-integrated to be aligned with the visual information.

The second thread is the point-line based tightly-coupled VIO. We use the VIO module of IDLL[17] in this thread. IDLL uses inverse depths of the end points of a line segment to represent the line, which reduces the line representation to two dimension of freedoms for efficiency. IDLL also provides good accuracy and is compatible with android, windows, and linux. By using IDLL, we run BA in a fixed size sliding window to minimize the measurement residuals to achieve accurate pose estimation and local mapping. Three tasks are accomplished on the sliding window: (1) construction of the sliding window; (2) initialization of features, and (3) optimization calculation of the states $\{\mathbf{X}^k, k \in \{1, \cdots, N\}\}$. We also designed a communication using ROSbridge to communicate with the server. The newly calculated pose, point and line features of the latest frame in the sliding window are transmitted to the server.

4.2 Pose Drift Correction

The agent receives two types of drift corrections from the server: (1) *keyframe drift correction* and (2) *local map drift correction*. The first type of drift correction is returned in real time from the map caching thread on the server. It includes pose corrections to all the N keyframes in local frame, which is denoted by: $\{(R^k_{\text{diff}}, \mathbf{t}^k_{\text{diff}}), k \in 1, \cdots, N\}$. After receiving this, the agent adjusts the pose of each keyframe in its sliding window by:

$$\begin{aligned} R_{q2r}(\hat{q}^k) &= R^k_{diff} R_{q2r}(q^k) \\ \hat{p}^k &= R^k_{diff} p^k + \mathbf{t}^k_{diff} \end{aligned} \tag{1}$$

where $\hat{q}^k$ and $\hat{p}^k$ are adjusted quaternion and position states of the k-th frame in the sliding window with $k \in 1, \cdots, N$. $R_{q2r}(q)$ is a function that transforms a quaternion q to its rotation matrix. Updating the poses in the sliding window will correspondingly update the point and line feature states through the following BA process in the sliding window. So the corrections to the features are not transmitted for efficiency.

The second type of correction, *local map drift correction*, is a local-to-global transformation drift returned from the server's global optimization thread that aligns the agent trajectory to the world coordinate system. It is denoted by $(R^i_w, \mathbf{t}^i_w)$ and offsets the entire local map of the ith agent.

$$\begin{aligned} R_{q2r}(\hat{\mathbf{q}}_{vio}) &= R^i_w R_{q2r}(\mathbf{q}_{vio}) \\ \hat{\mathbf{p}}_{vio} &= R^i_w \mathbf{p}_{vio} + \mathbf{t}^i_w \end{aligned} \tag{2}$$

where $(\mathbf{q}_{vio}, \mathbf{p}_{vio})$ and $(\hat{\mathbf{q}}_{vio}, \hat{\mathbf{p}}_{vio})$ are uncorrected and corrected pose trajectory of the agent respectively.

5 SERVER-SIDE COLLABORATIVE MAPPING MODULE

The server creates a cached sub-graph (CSG) for each agent, which is composed by the N nearest keyframes received from the agent. The loop closure edges will be detected and will be added into CSG. The server also conducts loop closure detection, global optimization for map fusion, communication with agents and global keyframe saving.

5.1 Agent map caching

On the server side we create an agent handler for each agent to receive keyframe pose and point and line information from the agent side, and create a thread to optimize the cached sub-graph (CSG). Both CSG and agent-side local map use the IDLL framework to optimize the keyframe states using N-keyframes sliding window. The difference is shown in Fig. 2. We add loop closure edges into the CSG. The loop closure edge is the visual reprojection error of the matching points and lines between the loop keyframes of the global map and the corresponding keyframes in the CSG. Because the result of loop detection is with the keyframes of the global map, we have to translate the keyframe poses, points and lines within the CSG to the world coordinate system of the global map. Therefore, the keyframes in CSG are in global coordinate system transformed by $\{R_w^i, \mathbf{t}_w^i\}$ of each keyframe.

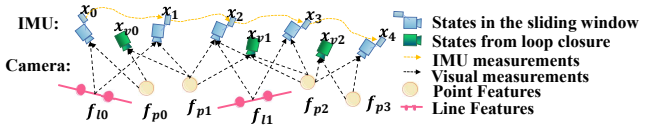

Figure 2: The complete state variables in sliding windows for agent map caching.

More specifically, CSG receives the keyframe poses, points and lines in the agent's local frame and transforms them to the world coordinate system π_w. Then loop detection to the global keyframes stored in the server is conducted. The detected loop frames in the server will be copied to the CSG. Then CSG is optimized with these added loop closure edges and keyframes. The specific bundle adjustment equation is in (3).

$$\begin{aligned}
&\min_{\mathcal{X}} \rho(\|r_p - J_p\mathcal{X}\|^2_{\sum p}) + \sum_{i\in B} \rho(\|r_b(z_{b_i b_{i+1}}, \mathcal{X})\|^2_{\sum b_i b_{i+1}}) \\
&+ \sum_{(i,j)\in F} \rho(\left\|r_f(z^{c_i}_{f_j}, \mathcal{X})\right\|^2_{\sum^{c_i}_{f_j}}) + \sum_{(i,j)\in L} \rho(\left\|r_l(z^{c_i}_{l_j}, \mathcal{X})\right\|^2_{\sum^{c_i}_{l_j}}) \\
&+ \sum_{(f,v)\in Loop} \rho(\left\|r_L(\hat{z}^v_f, \mathcal{X}, \hat{q}^w_v, \hat{p}^w_v)\right\|^2_{\sum^{c_v}_f}) \\
&+ \sum_{(l,v)\in Loop} \rho(\left\|r_L(\hat{z}^v_l, \mathcal{X}, \hat{q}^w_v, \hat{p}^w_v)\right\|^2_{\sum^{c_v}_l})
\end{aligned} \tag{3}$$

where $r_b(z_{b_i b_{i+1}}, \mathcal{X})$ is the pose measurement residual between the keyframe body state x_i and x_{i+1}. B is the set of all pose measurements within the sliding window. $r_f(z^{c_i}_{f_j}, \mathcal{X})$ and $r_l(z^{c_i}_{l_j}, \mathcal{X})$ are the point feature re-projection residual and the line feature re-projection residual, respectively. F and L are the sets of point features and line features extracted from camera frames. $r_L(\hat{z}^v_f, \mathcal{X}, \hat{q}^w_v, \hat{p}^w_v)$ is loop point re-projection residual and $r_L(\hat{z}^v_l, \mathcal{X}, \hat{q}^w_v, \hat{p}^w_v)$ is loop line re-projection residual, where *Loop* represents the set of loop closure frames. (f, v) represents the f-th matched feature point detected by loop closure frame v and (l, v) represents the l-th matched feature line detected by loop closure frame v. After marginalizing a frame from the sliding window, $\{r_p, J_p\}$ is prior information that can be calculated [41], and the prior Jacobian matrix J_p is from the Hessian matrix after the previous optimization. We can use ρ (Cauchy robust function) to constraint outliers.

By above optimization, after detecting each loop closure frame between a keyframe in the sliding window and a keyframe in the server's keyframe set, we copy the relative pose and matched point-lines of the server's keyframe into the sliding window optimization and compute the new poses $(\hat{p}^k_w, \hat{q}^k_w) \in \pi_w$ for all keyframes in the sliding window. Nonlinear optimization continues to run in CSG until a new keyframe arrives. When a new keyframe arrives, the $(R^k_{diff}, \mathbf{t}^k_{diff})$ update for all keyframes in the sliding window are computed by Eq(4) and sent back to the agent.

$$\begin{aligned}
R^k_{diff} &= (R^i_w)^T R_{q2r}(\hat{q}^k_w) R_{q2r}(q^k_i)^T \\
\mathbf{t}^k_{diff} &= (R^i_w)^T(\hat{p}^k_w - \mathbf{t}^{\mathbf{i}}_{\mathbf{w}}) - R^k_{diff} p^k_i
\end{aligned} \tag{4}$$

This approach ensures the accuracy of the short-term drift computation for the agent's local VIO, while avoiding the need for global pose optimization to be computed at every incoming keyframe, thus ensures the efficiency of the algorithm.

5.2 Loop detection & place recognition

We introduce the point-line based loop closure detection process in this subsection. The existing loop closure detection framework firstly searches for candidate frames that match the query frame and saves the index of the most similar candidate frame. Then, it performs outlier rejection and relative pose computation on the matching points on the candidate frames, and finally adds the qualified keyframes to the graph to add the loop closure residual edges.

We implement a more robust loop closure detection process by fusing point and line features based on the existing framework. Firstly, we propose a point-line fusion loop candidate selection method based on NetVLAD[42]. Then, the RANSAC-PNPL method is proposed to check whether the current frame and the highest-scored candidate frame really form a loop. It removes outliers from the matched points and lines of the current frame and the highest-scored candidate frame and rejects the loop closure hypothesis if too many outliers are detected.

Point and Line NetVLAD: NetVLAD is a method that compresses several local features into a specific-sized global feature. By clustering using neural networks and training by supervised data, we can group features belonging to the same object into one cluster and assign features belong to different categories large inter-disances. However, the existing NetVLAD methods directly extract features by the neural networks for better embedding the entire feature set. In ColSLAM, we exploit traditional handcrafted features, including Fast+Brief for points and LSD+LBD for lines to speed up the process. Therefore, we extract traditional features to train the NetVLAD and the overview is shown in Fig. 3. In particular, we extract N traditional features and their corresponding D-dimensional descriptors, and arrange the $N \times D$ descriptors into a feature map based on their distance from the top-left corner of

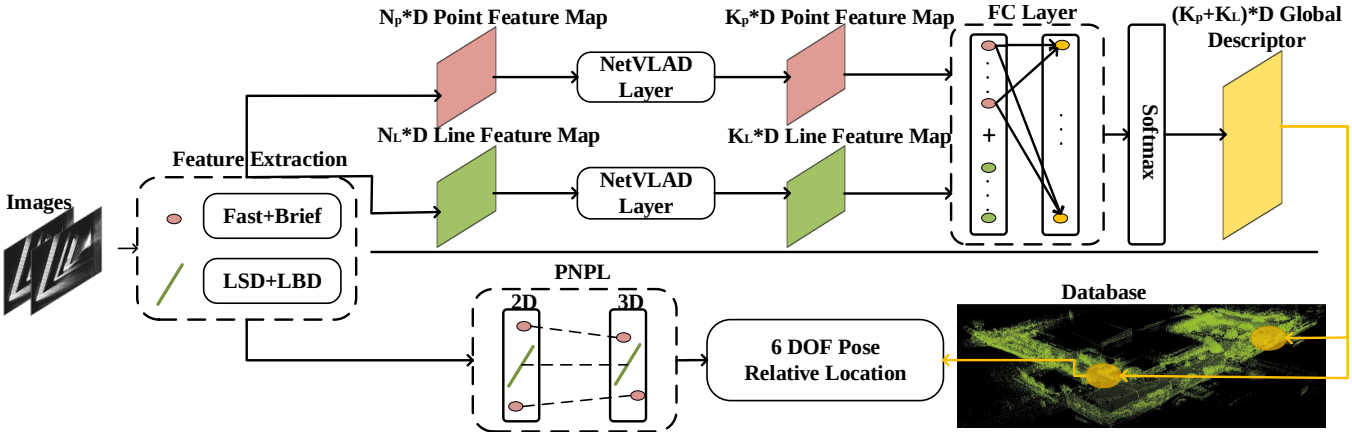

Figure 3: Overview of the proposed PL-NetVLAD.

the image. This feature map is then input into the NetVLAD layer for clustering and dimensionality reduction. In this way, we obtain the $K \times D$ dimensional descriptors of the point and line features after dimensionality reduction in the form of a feature map. Next, we use deep learning feature fusion to connect the point and line feature descriptors with a fully connected layer for fusion, resulting in a feature map of the fused point-line features. Finally, we use a triplet loss function (5) for weakly supervised training to obtain the loop detection model.

$$L(x_i, x_i^+, x_i^-) = \max(0, m + d(x_i, x_i^+) - d(x_i, x_i^-)) \tag{5}$$

where x_i is the $K \times D$ feature vector, x_i^+ and x_i^- represent the feature vectors of the positive and negative samples that absolutely match and absolutely do not match the image, respectively. m denotes the margin, and $d(\cdot, \cdot)$ represents the distance metric between two feature vectors.

Perspective-N-Point and Line (PNPL) Method: The Perspective-N-Point and Line (PNPL) method involves solving a system of equations to estimate the camera pose from a set of 3D points and their corresponding 2D projections, as well as a set of 3D lines and their corresponding 2D line segments in the image. Given n 3D-2D point correspondences, the equations can be written as:

$$[u_i, v_i, 1] = K[R|t][X_i, Y_i, Z_i, 1]^T \tag{6}$$

where (u_i, v_i) are the coordinates of the ith point on the image, (X_i, Y_i, Z_i) are the corresponding 3D point coordinates in the world frame, and K is the camera intrinsic matrix. R and $\mathbf{t}$ represent the rotation and translation of the camera's relative pose.

The 3D-2D line correspondences' equations are as:

$$\begin{aligned} L_i &= \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top[\mathbf{t}]_\times \\ 0 & \mathbf{R}^\top \end{bmatrix} \begin{bmatrix} \mathbf{n} \\ \mathbf{d} \end{bmatrix} \\ \mathbf{l}_i &= [l_1, l_2, l_3]^\top = \mathbf{K}_L \mathbf{n}_i \end{aligned} \tag{7}$$

where $\mathbf{K}_L = \begin{bmatrix} f_y & 0 & 0 \\ 0 & f_x & 0 \\ -f_y c_x & -f_x c_y & f_x f_y \end{bmatrix}$ denotes the line projection matrix and L is the 3D line, l_i are the image coordinates of the ith 2D line.

We aim to estimate R and $\mathbf{t}$ that minimize the error between the observed 2D points, lines and the points, lines projected from 3D. We use RANSAC method[43] to remove outliers. Then, we adopt the least squares method[44] to minimize both reprojections of the points and lines to estimate relative pose.

5.3 Cached map fusion and global pose optimization

Using the CSG results of the agents, the keyframes are then added to the global pose graph for map fusion. Each keyframe serves as a vertex in the global pose graph, connected to other vertices through two types of edges as shown in Fig. 4.

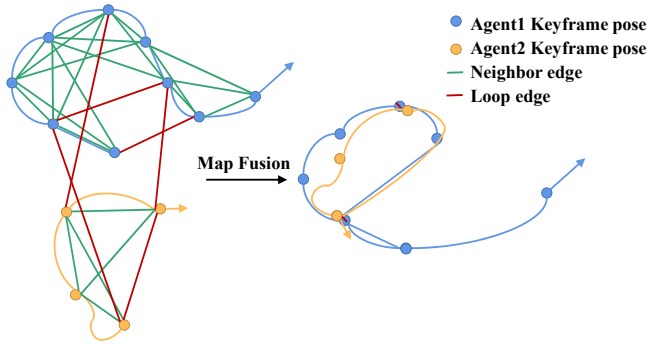

Figure 4: Neighbor edges and loop edges of pose graph.

(1)**Neighboring edges:** One keyframe is connected to several previous keyframes via neighboring edges. These neighboring edges represent the relative transformation between two keyframes and are directly obtained from the same agent's CSG.

(2)**Loop closure edges:** When a keyframe is detected as a loop closure frame, it is connected to some previous keyframes through loop closure edges in the pose graph, indicating that there exists a loop closure between these two keyframes. The value of the loop closure edge is calculated by the loop detection algorithm.

The residual between the keyframe i and the keyframe j is then defined as:

$$r_{i,j} = \begin{bmatrix} R_i^{-1}(p_j^w - p_i^w) - \hat{p}_{ij}^i \\ q_j \otimes q_i^{-1} - \hat{q}_{ij}^i \end{bmatrix} \tag{8}$$

where the positions and orientations are available in CSG. The optimization problem with the neighboring edges and the loop closure edges can then be formulated as the following optimization problem:

$$\min_{p,q}\{\sum_{(i,j)\in S}\|r_{i,j}\|_2+\sum_{(i,j)\in L}\|r_{i,j}\|_2\} \tag{9}$$

where S is the set of neighboring edges, and L is the set of loop closure edges. We do not use any robust kernel function for the consecutive edges, as these edges are extracted from CSG, which already been processed by sufficient outlier rejection mechanism.

The *local map drift* refers to the drift in coordinates from the CSG to the global map. It is detected by the offset in Eq(10) between the earliest loop closure frame in CSG and the global map.

$$\begin{aligned} R_w &= R_{q2r}(\hat{q}_{first})R_{q2r}(q_{first})^T \\ \mathbf{t}_w &= \hat{p}_{first} - R_w p_{first} \end{aligned} \tag{10}$$

where $\{q_{first}, p_{first}\}$ is the pose of the firstly detected keyframe in CSG and $\{\hat{q}_{first}, \hat{p}_{first}\}$ is the pose of the corresponding loop closure keyframe in the global keyframe set.

6 EXPERIMENTAL RESULTS

In this section, we first describe the configuration of our experiments in appendix A. Thereafter, we discuss the performance of ColSLAM in terms of accuracy, robustnes, and scalability.

6.1 Accuracy

We firstly compare the accuracy performances. EuRoc dataset[11] is used in the accuracy evaluation. We evaluate the root mean squared error (RMSE) of the absolute trajectory error (ATE) among VINS-MONO[6], IDLL[17], RAMA-SLAM[26], COVINS[23] and ColSLAM. The results are shown in Table1. In each experiment, two sequences, as shown in the first column in Table1, are used to evaluate the SLAM system. In the single-agent architecture, the agent runs on each sequence once. In the multi-agent collaborative SLAM (COVINS, RAMA-SLAM and ours), two agents run on the two sequences simultaneously. The average trajectory errors (ATEs) obtained from the localization results of two sequences provide a reliable measure of the accuracy. As shown in the Table1, we firstly note that the single-agent results of ColSLAM outperform the performances of IDLL. This shows the effectiveness of our loop closure detection method and sliding window relocalization method combining point and line features. We highlight the error decreasing rate from IDLL to single-agent ColSLAM in the 9th column.

Then we see the ATEs of multi-agent ColSLAM are significantly lower than that of VINS-MONO, IDLL, COVINS, RAMA-SLAM, and its single-agent version. This is because the single-agent SLAM can only query historical keyframes to add loop closure constraints, while multi-agent collaboration adds more loop closure edges as the multiple agents open the maps and accumulate keyframes more quickly. We highlight the error decreasing rate from single-agent ColSLAM to multiple-agent SLAM in the last column. The average error decreasing is around 20%. The multiple agent collaboration can effectively reduce the ATEs. What's more, COVINS and RAMA-SLAM work pretty much the same, suggesting that it's just that RAMA-SLAM is designed to be more lightweight and suitable for mobile phones. Only RAMA-SLAM and ColSLAM run on mobile phones, and ColSLAM is the best of all collaborative SLAM systems.

6.2 Robustness

To evaluate the robustness performance, we have specifically constructed a dataset for assessing the collaborative mapping capabilities of multiple mobile phones in a weak-textured laboratory environment with long, straight corridors. A total of 12 trajectories have been collected in https://ieee-dataport.org/documents/idls-inlab.

We evaluate the performances of different algorithms on these set of challenging trajectories. The ATEs of different algorithms on different sequences are evaluated and are shown in Table2. ColSLAM provides much better performances than that of point-based methods (VINS-Mobile, RAMA-SLAM) on pantry trajectory with dim light and the corridor trajectory, indicating that our method is much more robust by using the line features. VINS-Mobile and RAMA-SLAM fail to generate maps in these trajectories. A visualization example of the sequence No. 06 is shown in Fig. 5.

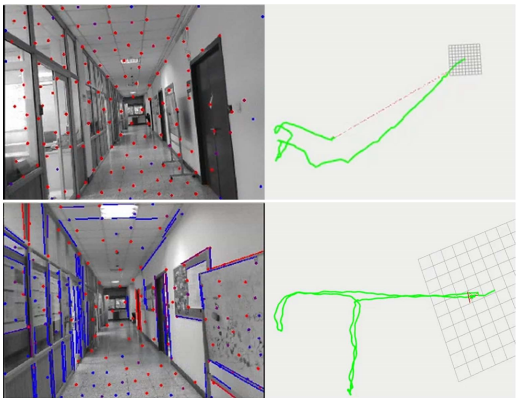

Figure 5: Visualization result of sequence-06. It can be seen that the VIO results diverse in RAMA-SLAM, while ColSLAM provides a reliable localization result.

6.3 Scalability

Next, we evaluate scalability performances. We excluded trajectories that RAMA-SLAM could not handle, so that 10 sequences are used. We compare the memory usage when there are 2, 5, and 10 agents are running RAMA-SLAM and ColSLAM respectively. The memory usage of the server is shown in Fig. 6. It can be seen that ColSLAM method has obviously lower memory usage. The memory usages of RAMA-SLAM and ColSLAM both increase with the number of agents, but the memery usage of ColSLAM increases much slowly than that of RAMA-SLAM. We also compare the mapping accuracy for different numbers of agents. ColSLAM provides much better mapping accuracy using the same number of agents. We also find that in the testing environment, five agents running ColSLAM are sufficient to achieve good map accuracy. The results of five agent collaboration are shown in Fig. 8.

Table 1: Accuracy performance of ColSLAM

Sequence	VINS-MONO	IDLL	COVINS		RAMA-SLAM		ColSLAM			
			single-agent	multi-agent	single-agent	multi-agent	single-agent		multi-agent	
MH_01&MH_02	0.059	0.058	0.061	0.089	0.060	0.058	0.051	↓*13.73%*	**0.041**	↓*24.39%*
MH_01&MH_03	0.069	0.055	0.061	0.055	0.061	0.065	0.052	↓*5.77%*	**0.042**	↓*23.81%*
MH_01&MH_05	0.139	0.137	0.186	0.174	0.168	0.167	0.136	↓*0.74%*	**0.123**	↓*10.57%*
MH_02&MH_03	0.077	0.071	0.071	0.061	0.066	0.069	0.065	↓*9.23%*	**0.044**	↓*47.73%*
MH_03&MH_05	0.171	0.152	0.181	0.162	0.194	0.200	0.148	↓*2.70%*	**0.131**	↓*12.98%*
MH_04&MH_05	0.111	0.093	0.106	0.101	0.110	0.099	0.090	↓*3.33%*	**0.081**	↓*11.11%*
V1_01&V1_02	0.051	0.046	0.048	0.046	0.040	0.051	0.046	*0*	**0.034**	↓*35.29%*
V1_01&V1_03	0.125	0.112	0.071	0.044	0.083	0.092	0.073	↓*53.42%*	**0.022**	↓*231.82%*
V2_01&V2_02	0.087	0.074	0.082	0.071	0.074	0.079	0.069	↓*7.25%*	**0.049**	↓*40.82%*
V2_01&V2_03	0.321	0.143	0.144	0.121	0.221	0.215	0.122	↓*17.21%*	**0.113**	↓*7.96%*

Table 2: Robust performance of ColSLAM

Seq No.	Environment	VINS Mobile	RAMA SLAM	ColSLAM
01	Lab	0.060	0.040	0.039
02		0.073	0.041	0.038
03		0.046	0.023	0.022
04	Pantry(dim light)	failed	failed	0.167
05	Corridor	failed	failed	0.254
06	(blank/glass wall)	10.6	10.6	0.135
07	Lab+Corridor	9.62	1.24	0.156
08	(blank/glass wall)	9.18	1.74	0.231
09	Mail room(similar feature)	3.52	0.130	0.111
10	Mail room+Hall	0.440	0.220	0.130
11		0.317	0.240	0.123
12	Corridor(doors)+Hall	0.287	0.215	0.113

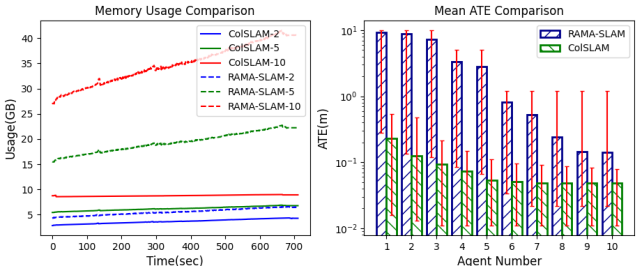

Figure 6: (a) Memory usage of RAMA-SLAM and ColSLAM. (b) ATE distribution of RAMA-SLAM and ColSLAM.

We also test the agent location updating latency when there are 5 agents. For ColSLAM, we measure each agent's *ColSLAM-pose correction latency*, i.e. the delay of the corrections returned by the cached map updating, *ColSLAM-map correction latency*, i.e., the delay of the corrections returned by the global mapping, and we compare them with *RAMA-SLAM pose correction latency*. The results are depicted in Fig. 7. We observe ColSLAM-pose correction latency is within 30ms, which is similar to the camera's inter-frame interval, which is nearly real-time. Although the *ColSLAM-map correction latency* is larger, it does not need to be responded in real-time.

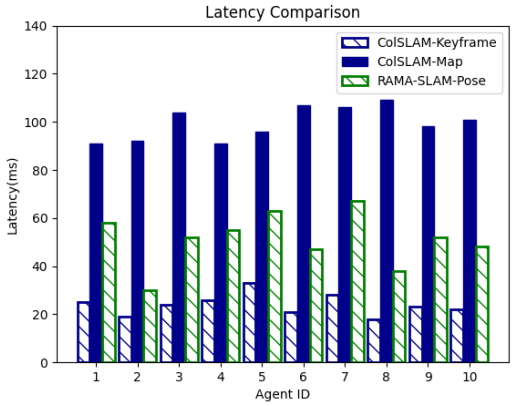

Figure 7: Latency of ColSLAM

7 CONCLUSIONS

In this paper, we propose an innovative collaborative SLAM system, ColSLAM, which can be used to map large-scale indoor environments in real-time. The map caching scheme, one of the key features of ColSLAM, allows for the utilization of measurement information shared by multiple agents, resulting in a more comprehensive and accurate global map. The joint optimization method for point and line features in VIO improves the robustness of ColSLAM in weak-texture environments, while the collaborative loop detection mechanism enhances the accuracy of of ColSLAM through NetVLAD loop detection and PNPL relative pose calculation. Empirical evaluations based on the EuRoc dataset and our own dataset confirm the effectiveness of the proposed system, with significant improvements in accuracy, robustness, and scalability. We believe that the proposed system has the potential to thoroughly advance the fields of robotics, autonomous systems, and augmented reality. In the future, we plan to integrate more mobile phone sensors (e.g., GPS and barometer) to extend the system for large-scale collaborative map building in joint indoor and outdoor environments.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China Grant No. 61972404, 12071478; Public Computing Cloud, Renmin University of China; Blockchain Laboratory, Metaverse Research Center, Renmin University of China.

REFERENCES

[1] Xiuquan Qiao, Pei Ren, Schahram Dustdar, Ling Liu, Huadong Ma, and Junliang Chen. Web ar: A promising future for mobile augmented reality—state of the art, challenges, and insights. *Proceedings of the IEEE*, 107(4):651–666, 2019.

[2] Danping Zou, Ping Tan, and Wenxian Yu. Collaborative visual slam for multiple agents: A brief survey. *Virtual Reality & Intelligent Hardware*, 1(5):461–482, 2019.

[3] Ming Ouyang, Xuesong Shi, Yujie Wang, Yuxin Tian, Yingzhe Shen, Dawei Wang, Peng Wang, and Zhiqiang Cao. A collaborative visual slam framework for service robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8679–8685. IEEE, 2021.

[4] Pierre-Yves Lajoie, Benjamin Ramtoula, Fang Wu, and Giovanni Beltrame. Towards collaborative simultaneous localization and mapping: a survey of the current research landscape. *arXiv preprint arXiv:2108.08325*, 2021.

[5] Abhishek Gupta and Xavier Fernando. Simultaneous localization and mapping (slam) and data fusion in unmanned aerial vehicles: Recent advances and challenges. *Drones*, 6(4):85, 2022.

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

[7] Lingqiu Jin, He Zhang, and Cang Ye. Camera intrinsic parameters estimation by visual–inertial odometry for a mobile phone with application to assisted navigation. *IEEE/ASME Transactions on Mechatronics*, 25(4):1803–1811, 2020.

[8] Jannis Möller, Benjamin Meyer, and Martin Eisemann. Porting a visual inertial slam algorithm to android devices. 2019.

[9] Zhongli Wang, Yan Shen, Baigen Cai, and Muhammad Tariq Saleem. A brief review on loop closure detection with 3d point cloud. In *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 929–934. IEEE, 2019.

[10] Saba Arshad and Gon-Woo Kim. Role of deep learning in loop closure detection for visual and lidar slam: A survey. *Sensors*, 21(4):1243, 2021.

[11] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[12] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3565–3572. IEEE, 2007.

[13] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. Openvins: A research platform for visual-inertial estimation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4666–4672. IEEE, 2020.

[14] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[15] Yijia He, Ji Zhao, Yue Guo, Wenhao He, and Kui Yuan. Pl-vio: Tightly-coupled monocular visual–inertial odometry using point and line features. *Sensors*, 18(4):1159, 2018.

[16] Qiang Fu, Jialong Wang, Hongshan Yu, Islam Ali, Feng Guo, Yijia He, and Hong Zhang. Pl-vins: Real-time monocular visual-inertial slam with point and line features. *arXiv preprint arXiv:2009.07462*, 2020.

[17] Yongcai Wang Shuo Wang Xuewei Bai Deying Li Wanting Li, Yu Shao. IDLL: Inverse Depth Line based Visual Localization in Challenging Environments. *arXiv e-prints*.

[18] Robert Castle, Georg Klein, and David W Murray. Video-rate localization in multiple maps for wearable augmented reality. In *2008 12th IEEE International Symposium on Wearable Computers*, pages 15–22. IEEE, 2008.

[19] Danping Zou and Ping Tan. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):354–366, 2012.

[20] Jacob M Perron, Rui Huang, Jack Thomas, Lingkang Zhang, Ping Tan, and Richard T Vaughan. Orbiting a moving target with multi-robot collaborative visual slam. In *Proceedings of the Workshop on Multi-View Geometry in Robotics (MVIGRO), Rome, Italy*, pages 1339–1344, 2015.

[21] Luis Riazuelo, Javier Civera, and JM Martınez Montiel. C^2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[22] Marco Karrer, Patrik Schmuck, and Margarita Chli. Cvi-slam—collaborative visual-inertial slam. *IEEE Robotics and Automation Letters*, 3(4):2762–2769, 2018.

[23] Patrik Schmuck and Margarita Chli. Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics*, 36(4):763–781, 2019.

[24] Patrik Schmuck, Thomas Ziegler, Marco Karrer, Jonathan Perraudin, and Margarita Chli. Covins: Visual-inertial slam for centralized collaboration. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 171–176, 2021.

[25] Arindam Saha, Bibhas Chandra Dhara, Saiyed Umer, Ahmad Ali AlZubi, Jazem Mutared Alanazi, and Kulakov Yurii. Corb2i-slam: An adaptive collaborative visual-inertial slam for multiple robots. *Electronics*, 11(18):2814, 2022.

[26] Jialing Liu, Kaiqi Chen, Ruyu Liu, Yanhong Yang, Zhenhua Wang, and Jianhua Zhang. Robust and accurate multi-agent slam with efficient communication for smart mobiles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2782–2788. IEEE, 2022.

[27] Jialing Liu, Ruyu Liu, Kaiqi Chen, Jianhua Zhang, and Dongyan Guo. Collaborative visual inertial slam for multiple smart phones. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11553–11559. IEEE, 2021.

[28] Jingao Xu, Hao Cao, Zheng Yang, Longfei Shangguan, Jialin Zhang, Xiaowu He, and Yunhao Liu. {SwarmMap}: Scaling up real-time collaborative visual {SLAM} at the edge. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 977–993, 2022.

[29] Titus Cieslewski, Siddharth Choudhary, and Davide Scaramuzza. Data-efficient decentralized visual slam. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2466–2473. IEEE, 2018.

[30] Pierre-Yves Lajoie, Benjamin Ramtoula, Yun Chang, Luca Carlone, and Giovanni Beltrame. Door-slam: Distributed, online, and outlier resilient slam for robotic teams. *IEEE Robotics and Automation Letters*, 5(2):1656–1663, 2020.

[31] Siddharth Choudhary, Luca Carlone, Carlos Nieto, John Rogers, Henrik I Christensen, and Frank Dellaert. Distributed trajectory estimation with privacy and communication constraints: a two-stage distributed gauss-seidel approach. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5261–5268. IEEE, 2016.

[32] Siddharth Choudhary, Luca Carlone, Carlos Nieto, John Rogers, Henrik I Christensen, and Frank Dellaert. Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models. *The International Journal of Robotics Research*, 36(12):1286–1311, 2017.

[33] Yun Chang, Yulun Tian, Jonathan P How, and Luca Carlone. Kimera-multi: a system for distributed multi-robot metric-semantic simultaneous localization and mapping. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11210–11218. IEEE, 2021.

[34] Shuo Wang, Yongcai Wang, Deying Li, and Qianchuan Zhao. Distributed relative localization algorithms for multi-robot networks: A survey. *Sensors*, 23(5), 2023.

[35] Jianbo Shi et al. Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600. IEEE, 1994.

[36] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[37] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2008.

[38] Lilian Zhang and Reinhard Koch. An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency. *Journal of Visual Communication and Image Representation*, 24(7):794–805, 2013.

[39] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library.* " O'Reilly Media, Inc.", 2016.

[40] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[41] Shaojie Shen, Nathan Michael, and Vijay Kumar. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft mavs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5303–5310. IEEE, 2015.

[42] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016.

[43] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[44] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[45] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2930–2937, 2013.

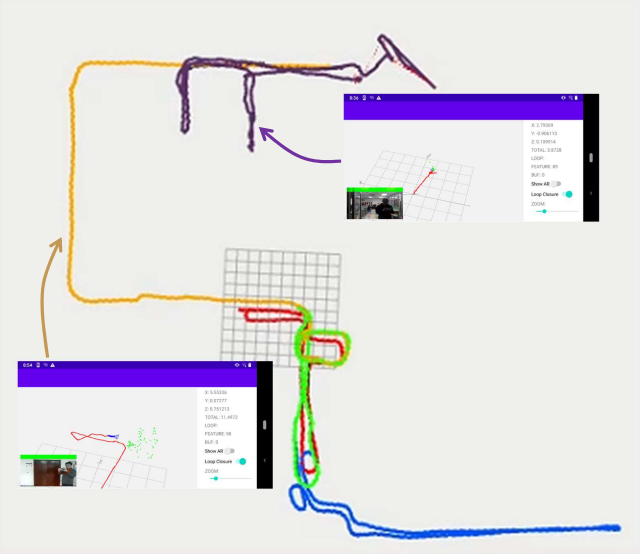

Figure 8: The visualization of the collaborate SLAM results on the server.

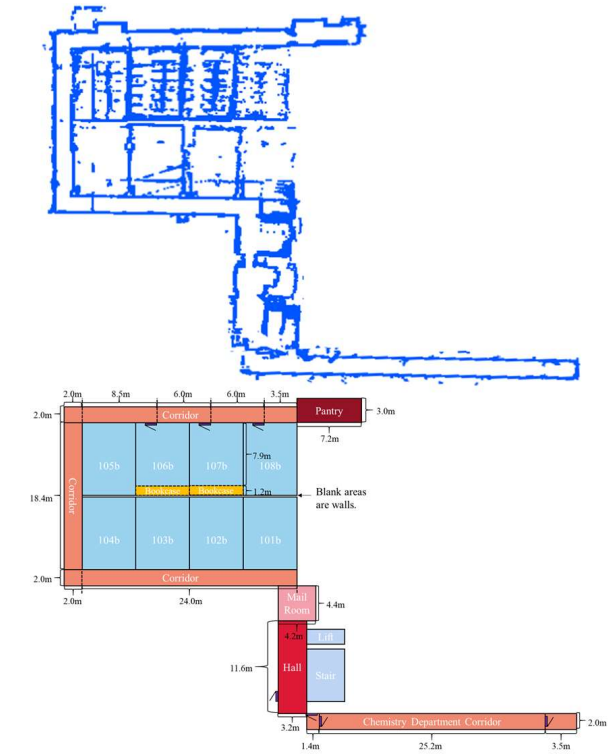

Figure 9: The top figure is the feature map, and the bottom figure is the ground truth of the scene.

A EXPERIMENT CONFIGURATION

Compared methods: CoLSLAM is compared with VINS-Mono[6], IDLL[17], COVINS and RAMA-SLAM[26]. The former two are single agent SLAM methods run on a PC, COVINS[23] is a collaborative SLAM system where the agents run on PCs, and RAMA-SLAM is the state-of-the-art collaborative SLAM system where the agents run on mobile phones and the server runs on a Server. Meanwhile VINS-Mono has a version that supports running on mobile phones, VINS-Mobile.

Device and network settings: The device settings of the PC, the phones, and the Server are compared in Table3. In ColSLAM and RAMA-SLAM, agents communicate with the server through wireless network. The network bandwidth is measured to be around 95Mbps, which can well support 200 point features and 100 line features to be communicated with low latency.

Table 3: Hardware Configuration of Devices

Type	Platform	Configuration
Server	Ubuntu 18.04	i7-11700, 2.50GHz CPU, 64GB RAM
Phones	Android 10	Tianji 700, 2.20GHz CPU, 6GB RAM
PC	Ubuntu 18.04	i7-11700, 2.50GHz CPU, 12GB RAM

Datasets for NetVLAD training: Two datasets are used to train the NetVLAD based loop detection model, i.e., Pittsburgh 250k[42] and 7Scenes[45]. The number of point features extracted on each image is 200, and the number of line features is 100. For the outdoor Pittsburgh 250k dataset, positive loops are images within $10m$ from the query image, and negative samples are images farther than $25m$ from the query image. For the indoor 7Scenes dataset, positive loops are the images within $0.1m$ from the query image and negative samples are the images more than $0.25m$ from the query image. The margin of training is set to 0.1m.

B VISUALIZATION RESULT OF COLSLAM

Server-side display of trajectory results and screenshots of the mobile phone run as shown in Fig.8. The feature map created on the server side is shown in Fig.9.

C CONTRIBUTIONS OF PL-NETVLAD

The ability to accurately detect loop closures and relocalize the map is crucial for successful collaborative SLAM and map fusion. We evaluate the contributions of PL-NetVLAD, by comparing the results with that using only DBoW2 and point-based VLAD. The results are presented in Table 4.

Table 4: Loop detection

Method	Tokyo 24/7			EuRoc
	Recall@1	Recall@5	Recall@10	Recall<0.1m
DBOW2	44.7	58.3	69.1	46.0
NetVLAD	64.4	78.4	81.6	65.7
PL-NetVLAD	**80.2**	**83.5**	**90.5**	**88.2**

We use the Tokyo 24/7 and EuRoc datasets to evaluate the performances of PL-NetVLAD. We focus on the recall of loop detection within 0.1m in the EuRoc dataset. Our proposed method improves over other mainstream methods by nearly 10 percentage points. Furthermore, we conducted experiments on the outdoor Tokyo 24/7 dataset. PL-NetVLAD improves over other loop closure detection methods by more than 20 percentage points.