International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 15, Number 2, pages 85–101, July 2025

Enhancing Robustness and Accuracy in Edge-Assisted Visual SLAM Implementation

Chenzhang Xia
Kyushu University, Graduate School of
Information Science and Electrical Engineering,
Fukuoka, Japan

Yuan Wang
Kyushu University, Graduate School of
Information Science and Electrical Engineering,
Fukuoka, Japan

Koji Inoue
Kyushu University, Faculty of
Information Science and Electrical Engineering,
Fukuaka, Japan

Received: February 15, 2025
Revised: April 26, 2025
Accepted: May 29, 2025
Communicated by Michihiro Koibuchi

Abstract

With the increasing demand for spatial positioning on modern mobile devices, Simultaneous Localization and Mapping (SLAM), particularly camera-based Visual SLAM, has become essential for real-time perception and positioning by processing continuous image data. However, these algorithms often entail high memory and computational requirements, making it challenging to deploy them on mobile devices and run for extended periods. To address this issue, the edge-assisted SLAM architecture, which offloads computationally intensive tasks to edge servers, has been proposed. Despite its potential, existing solutions in this domain suffer from significant limitations in data synchronization and recovery capability, compromising both the robustness and accuracy of the system. In response to the identified limitations, we analyze the impact of the current data synchronization and relocalization recovery processes on system performance, and introduce a novel multithreaded tracking approach integrated with an efficient relocalization mechanism. We validated our approach in standard datasets, including the robustness of the system, tracking recovery capability, and localization accuracy. Experimental results demonstrate that our solution reduces tracking interruptions by up to 94.2%, significantly improves coverage, a vital robustness metric of the SLAM system, by up to 30.1%, and shortens relocalization recovery time by up to 35.2%. Furthermore, our approach improves the localization accuracy by 43.7% in translation scenarios and 36.8% in rotation scenarios.

Keywords: visual slam, edge computing, edge assisted slam, multithreaded system, thread refactoring, slam relocalization

[0]Part of this work was previously published in CANDAR 2024, focusing primarily on the parallel thread updates and keyframe backup mechanism

1 Introduction

With the rapid advancement of sensor technology and network connectivity, a growing number of applications leveraging spatial and environmental information are emerging on mobile devices. Mobile platforms have become a popular choice due to their convenience and portability. For these applications, precise spatial data acquisition and rapid processing are crucial, particularly for tasks such as real-time positioning and path planning. In addition, Augmented Reality (AR) applications also rely on real-time environmental data to interact with users [1]. Spatial perception has been a prominent research topic for years, with various modes tailored to different applications. These include location recognition, where an image (e.g., from a camera) is captured and matched to a previously known location; odometry, which tracks or estimates the path taken by a mobile device from a starting point; and localization, which involves determining the absolute position of a mobile device relative to known landmarks. Each method offers distinct advantages and trade-offs regarding computational complexity and practical applicability, making them suitable for a wide range of use cases. In this context, Visual Simultaneous Localization and Mapping (Visual SLAM) stands as the foundational and most effective algorithm for spatial positioning [2]. It collects environmental data through cameras, performs algorithmic processing for position estimation, and generates a map with environmental information. A typical Visual-SLAM algorithm consists of three primary tasks. First, as the mobile device moves, the algorithm performs inter-frame alignment by matching feature points and selects keyframes that capture significant environmental information. The second step involves local mapping, where the pose information calculated in the first step is optimized. The third step is loop closure, which enables the algorithm to recognize when the mobile device revisits a previously encountered location. This requires the algorithm to compare the current frame with all prior frames to identify a match. Visual-SLAM can accurately estimate the camera's pose and the movement path; however, as a resource-intensive algorithm, its memory and computational requirements increase over time. This growing demand for resources presents a significant challenge when deploying the algorithm on mobile devices; this usually results in an inability to process image data in real-time and low accuracy, especially when integrating it with other applications [3]. Some simplified versions of Visual SLAM minimize the complexity of the optimization algorithm to allow it to run on mobile devices, but this typically leads to a significant decrease in system accuracy [4], [5].

To address this challenge, increasing research efforts are focused on offloading computationally intensive tasks in Visual SLAM to external devices, thereby ensuring the system's stable operation [6]. Given the system's real-time performance requirement, integrating edge computing and distributing tasks to local devices near the data source presents a viable solution. In contrast to cloud computing, edge computing has lower transmission latency [7]. This approach, where complex computational tasks are offloaded to edge servers, is commonly referred to as edge-assisted Visual SLAM [8]. In the system, the mobile device estimates the position through image tracking, referred to as the Tracking module, while tasks such as map optimization are offloaded to the edge server. The optimized map is then returned to the mobile device, which synchronizes the data and updates the map. While this distributed architecture reduces the computational load on mobile devices, it also brings several challenges. Existing studies [9, 10] focus more on improving the transmission efficiency between mobile devices and edge servers, but they do not consider the impact of data synchronization delay or the system recovery capabilities on the mobile device. First, as the Tracking module constantly requires access to the map data, tracking interruptions occur during the local map update to prevent conflicts and maintain map consistency. Extended interruptions in tracking degrade the system's robustness and negatively impact its positioning accuracy. Second, when tracking is lost due to insufficient collected information, the system relies on the global map for relocalization. Since the global map is stored on the edge server, the communication delay between the mobile device and the edge server substantially increases the repositioning recovery time, which also worsens the pose estimation error. In response to these challenges, we base our approach on an advanced edge-assisted Visual SLAM system (i.e., Edge-SLAM [11]), where the data synchronization process of the mobile device is split into independent threads for parallel processing, and the relocalization process is offloaded to the edge server to enhance the system's robustness and accuracy. Our research makes the following contributions:

- We split the local map update in Edge-SLAM into independent threads for parallel processing, significantly enhancing the system's robustness. As a result, compared to Edge-SLAM, the tracking interruption time is reduced by approximately 94.2%, while coverage [12], a key robustness metric for SLAM systems, improves by up to 30.1%.
- We offload the system's relocalization process to the edge server, effectively reducing the time required for relocalization recovery and enhancing the system's accuracy. Compared to Edge-SLAM, we reduce the time required for system relocalization recovery by up to 35.2%.
- After evaluation using standard datasets, our solution demonstrates a significant improvement in system accuracy, with an enhancement of up to 43.7% in translation scenarios and 36.8% in rotation scenarios.
- We perform experimental evaluations on hardware consumption, including CPU and memory usage. The results show that our approach does not incur significant additional costs.

This study is divided into five parts: introduction, research background (visual-SLAM, edge-assisted architecture, and system analysis), system introduction (design concepts and system mechanism), experimental evaluation, and conclusion.

2 Background

2.1 Visual SLAM Modules

The Visual SLAM system is mainly divided into three modules. The Tracking module is responsible for processing the information collected by the camera and estimating the camera position in real time; the Local Mapping and Loop Closing modules are responsible for optimizing the camera position information. The specific module descriptions are as follows:

- The Tracking module compares the frame passed into the system with the feature information of the previous frame or the previously stored feature information and estimates the current posture of the camera, including position and direction. Frames containing a large amount of new environmental information are selected as key frames by other modules for subsequent processing. If the matching feature information is insufficient, the system will determine that tracking has been lost [11], and it needs to rely on the relocalization algorithm to restore normal tracking. Long-term tracking loss will greatly reduce the system's accuracy and affect the user experience.
- The Local Mapping module will perform local Bundle Adjustment (BA) to optimize the posture information of all keyframes to improve the positioning accuracy of the system.
- The Loop Closing module is responsible for detecting whether the system returns to the visited location during operation. If so, the system will correct and optimize the accumulated positioning error.

Each module depends on the map, a shared data structure that stores the spatial information computed by the system, including camera poses and feature point information. Maps play a crucial role in Visual SLAM, as inaccurate map data can impede the system's long-term stability and performance [13]. Therefore, in Visual SLAM research for mobile devices, the emphasis is placed on reducing computational load while maintaining map accuracy [10, 14, 15].

2.2 Related Work of Edge-assisted Visual SLAM

Visual SLAM systems are increasingly being deployed on mobile devices. However, a significant challenge arises from the conflict between the high computational demands of SLAM and the limited processing power of mobile devices [2]. To address this issue, ORB-SLAM [3] and LSD-SLAM [4]

introduced a tracking-only mode to reduce computational load, significantly compromising positioning accuracy. The advent of edge-assisted architectures has emerged as a practical solution [15], offloading computational tasks from mobile devices to edge servers via local network hopping, thus alleviating the strain on mobile resources.

For the related research, CloudSLAM [16] offloads the Loop Closure module to the cloud, while only the Tracking and Local Mapping modules are executed on the mobile device. However, it does not significantly reduce the resource consumption on the mobile device. Based on ORB-SLAM2, a typical Visual SLAM system, the Edge-SLAM framework, proposed by Ben Ali A J et al., serves as a classic example of this edge-assisted approach. In this study, the Tracking module operates on the mobile device, while other modules are offloaded to the edge server. The mobile device retains only a portion of the map, while the complete map is stored on the edge server. The edge periodically sends optimized map data back to the mobile device, significantly reducing CPU and memory usage on the mobile device. Building upon Edge-SLAM, Sossalla et al. investigated the impact of data exchange frequency between mobile devices and edge servers on system performance and optimized the local map update mechanism [17]. PRE-SLAM further enhanced this framework by incorporating feature persistence filtering, which filters dynamic feature points to reduce the mobile device's workload, improving positioning accuracy. However, most existing studies on edge collaboration focus primarily on enhancing system performance during normal tracking, with limited attention given to the effects of data synchronization delays and the system's ability to recover after tracking failure.

On the other hand, relocalization is also a key focus in Visual SLAM, with various methods proposed. For example, a binary test classifier in [18] identifies feature correspondences for relocalization, but its high memory requirements limit performance in large-scale environments. In [19,20], a bag-of-words approach is used with a visibility map to find revisited locations. Straub et al. [21] match descriptors of the current frame with map points to estimate pose, though it is computationally expensive. To improve efficiency, Moteki et al. [22] propose an image-to-image or image-to-map approach based on geometric modeling between the current and target frames. The above relocalization strategies are implemented in systems like ORB-SLAM2, which struggles with tracking failures and requires high similarity between the new frame and the reconstructed map. System recovery is crucial for accuracy and robustness. Our research improves edge-assisted Visual SLAM by redesigning data synchronization and enhancing the relocalization mechanism to boost performance.

2.3 Analysis on Edge-SLAM

Our study is based on the typical edge-assisted visual SLAM architecture Edge-SLAM [11], which divides the representative visual SLAM system ORB-SLAM2 into a mobile device and an edge server. The Tracking module remains on the mobile device in this setting, while the Local Mapping and Loop Closure modules are offloaded to the edge server. Same as ORB-SLAM2, it uses the ORB feature algorithm (Oriented FAST and rotated BRIEF) [23] to extract features from image frames. It can remain stable in dynamic environments with a fast calculation speed and is suitable for real-time SLAM. In addition, it uses optical flow estimation to perform pose estimation. By analyzing the changes in pixel brightness in two consecutive frames of images, the direction and speed of movement of the midpoint in the image are calculated, thereby estimating the object's or camera's relative motion. After obtaining the initial pose, BA (Bundle Adjustment) [24] is used to optimize the estimated results. It has high accuracy and real-time performance, which is suitable for moving objects. The map part only keeps a small part of the map (called the local map) on the mobile device, which is required for tracking, while the entire map (called the global map) is stored on the edge server. Once a keyframe has been identified, the pertinent information is transmitted to the server and incorporated into the global map. The mobile device periodically receives the optimized location information from the edge server and synchronizes it with the local map. Compared with ORB-SLAM2, Edge-SLAM adds a local map update and modifies the relocalization procedure. By analyzing these two processes, we identified the key issues in the current system and proposed corresponding solutions.

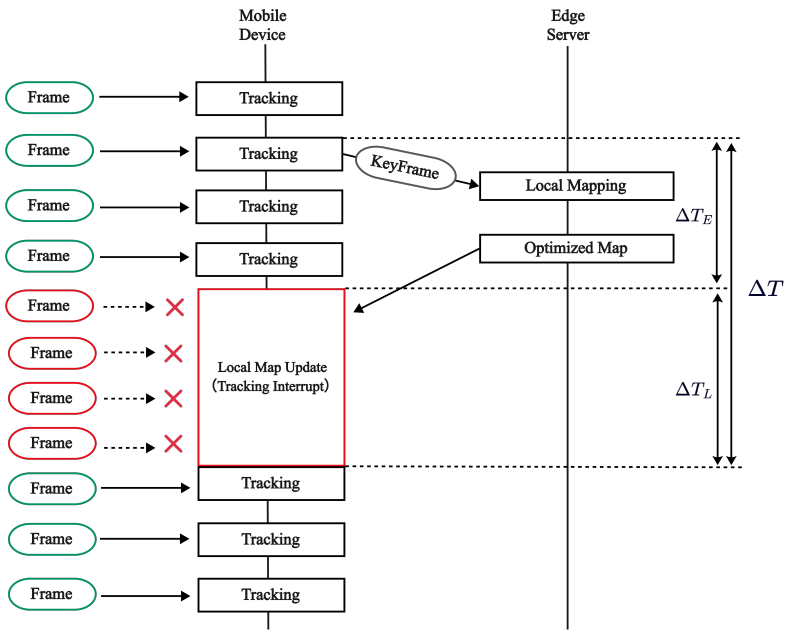

Figure 1: Data synchronization in Edge-SLAM system. Green represents frames processed by the system, red represents frames not processed by the system, and gray represents keyframes.

Table 1: The execution time proportion of each process in Edge-SLAM system

System	***Process***	***Proportion***
Edge-SLAM	Feature Detection	26.9%
	Local Map Update	**20.4%**
	Pose Prediction	35.1%
	Keyframe Creation	17.6%

2.3.1 Local Map Update

In Edge-SLAM, the local map update is an integral part of the Tracking module and is executed sequentially. Before processing each incoming frame, the system checks if an update to the map is necessary. When a data synchronization update is required, the system pauses the Tracking module, clears the existing map, and loads the updated map sent from the edge server. The detailed data synchronization process is illustrated in Figure 1. The total data synchronization time ΔT can be divided into edge data processing time ΔT_E and local map update time ΔT_L, which can be summarized as (1):

$$\Delta T = \Delta T_E + \Delta T_L \tag{1}$$

Since Edge-SLAM temporarily stops receiving frames during the local map update, the Tracking module experiences an interruption, with the duration of this interruption equal to the local map update time, denoted as ΔT_L. Once the update is complete, the Tracking module resumes operation. However, prolonged interruptions in tracking prevent the system from continuously providing location estimates, significantly compromising system robustness and leading to a loss of positioning accuracy and stability [25]. We evaluated the proportion of execution time consumed by the system's main processes, as shown in Table 1. These processes mainly include feature detection (extracting feature points from image frames), local map update(updating the local map based on data received from the edge server), pose prediction (estimating the current pose by analyzing spatial changes in feature points), and keyframe creation (selecting frames with significant changes as keyframes).

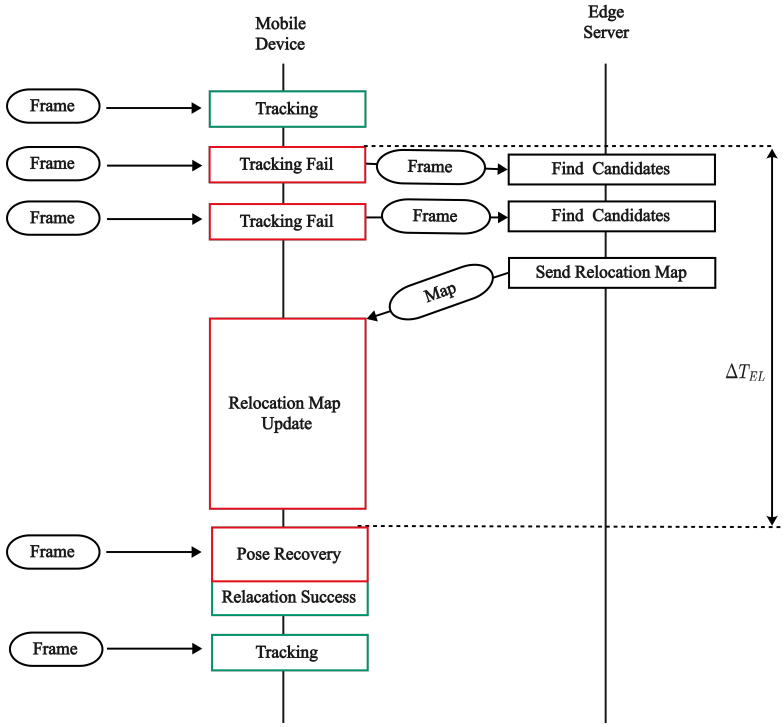

Figure 2: Relocalization mechanism in Edge-SLAM system. Green means the system is in a normal tracking state, and red means the system is in a relocalization state.

We find that local map update, i.e., ΔT_L in Figure 1, accounts for 20.4% of the total execution time, which is a result of the introduction of the edge-distributed architecture. Therefore, we should reduce the local map update time ΔT_L. To this end, we process the entire local map update as an independent thread in parallel and propose a keyframe backup mechanism to ensure the localization accuracy of the system.

2.3.2 Relocalization Recovery

Edge-SLAM tracking has two operating states. The system stays in normal tracking mode when the feature points between frames are sufficiently matched. However, it enters relocalization mode if tracking is lost due to insufficient feature point matching caused by fast movement, sharp turns, or other factors. In this mode, the system attempts to optimize the position and resume normal tracking. To achieve this, the system needs to find suitable candidate keyframes in the map to perform pose recovery. Since the global map is on the edge server and the mobile device only saves part of the map information, it is necessary to send a request to the edge server, and the required set of candidate keyframes is transmitted back as a relocalization map at a fixed interval, as shown in Figure 2. We can find that the time from the mobile device sending the first relocalization request to receiving the relocalization data from the edge server and completing the local map update is ΔT_{EL}, which is an additional delay for the relocalization process. Because the relocalization map is large, it takes more time to transmit data and update the map. Based on the standard TUM datasets [26] that are suitable for evaluating the relocalization recovery performance of the system, we compared the time required for relocalization recovery of ORB-SLAM2 and Edge-SLAM; we found that Edge-SLAM takes much more time to recover from the tracking loss status. The extended relocalization state seriously affects the accuracy of the system; we should avoid these additional delays to shorten the relocalization duration. To this end, we redesigned the relocalization method and adopted an efficient mechanism to complete relocalization recovery.

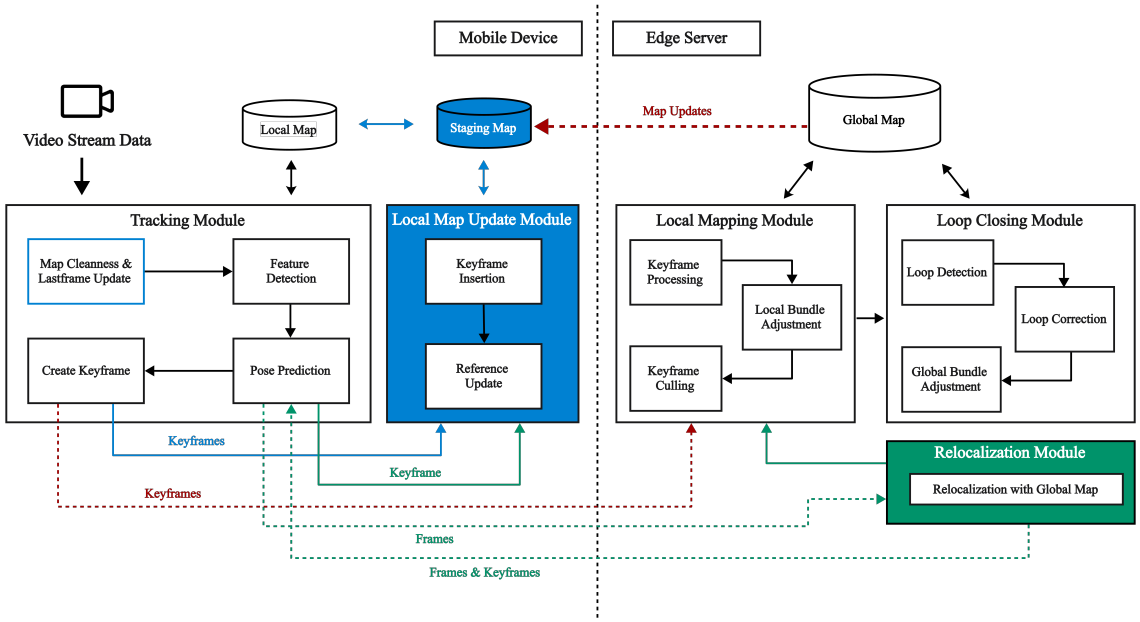

Figure 3: Architecture of the Enhanced Edge-Assisted Visual SLAM System

3 Enhancing Robustness and Accuracy in Edge-Assisted Visual SLAM Implementation

3.1 System Design

Our work consists of two parts: local map update optimization and relocalization improvement. The system architecture in Figure 3 shows these parts in blue and green, respectively. First, to shorten the tracking interruption time caused by local map updates, we separate the update process into a single module, namely the Local Map Update module shown in Figure 3, which is parallel to the Tracking module. The red dotted line is the data interaction between the mobile device and the edge server. The parallel approach can significantly shorten the system interruption time caused by local map updates, but there are also some challenges. Since the Tracking module is tightly coupled with the local map, it is challenging to decouple, and the current system architecture utilizes a single local map, which prevents the parallel processing of the Tracking module and Local Map Update module. Furthermore, even if the local map update is separated from the Tracking module, the frame information processed by the Tracking module will still be lost during the update procedure. To address these challenges, we add a map storage structure dedicated to local map updates and split the process into more detailed parts to ensure efficient data synchronization. When the mobile device receives map information from the edge server, the Local Map Update module starts synchronization. During this period, the Tracking module is not interrupted and continues to use the local original map for tracking. To prevent the loss of keyframe information during the local map update, we also back up the keyframe information generated during this period and send it to the Local Map Update module to merge it with the new map to improve the system's positioning accuracy.

Then, to complete the relocalization recovery work more efficiently, we move the relocalization algorithm to the edge to ensure that it can be completed as quickly as possible, that is, the Relocalization module on the edge server side shown in Figure 3. A dedicated communication channel transmits relocation data to minimize delays and prevent network congestion. Upon successful relocalization on the server, the optimized pose and reference keyframe are returned to the mobile device.

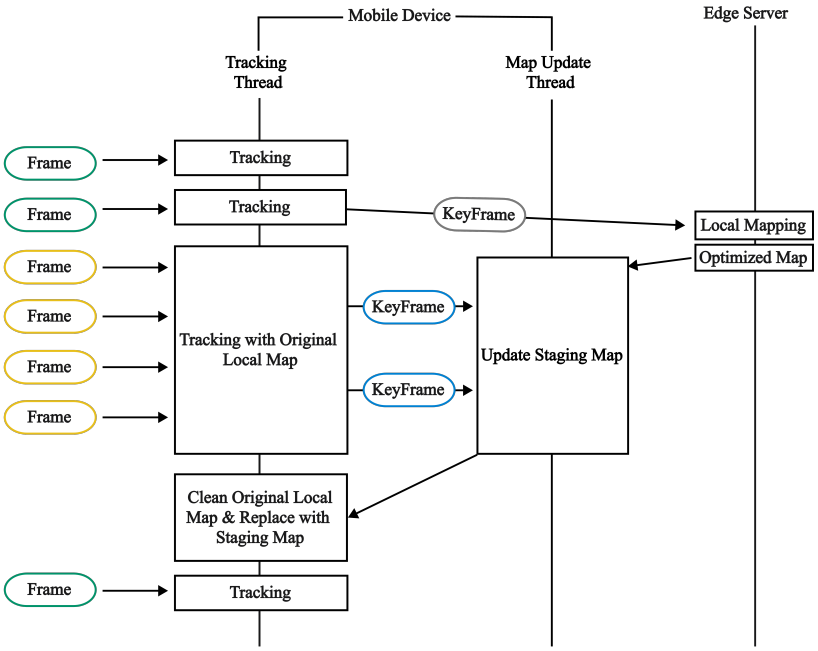

Figure 4: Data parallel synchronization with a keyframe backup mechanism. Frames marked in yellow are lost when the map is updating, and blue are keyframes added to the Staging Map.

Table 2: The execution time proportion of each function in the local map update

Process	***Functions***	***Proportion***
Local Map Update	Map Cleanness	7.03%
	Keyframe Insertion	**87.72%**
	Reference Update	0.16%
	Lastframe Update	5.09%

3.2 Thread Splitting and Keyframe Backup

To enable parallel processing of local map updates, we decouple the update process from the Tracking module by splitting it into smaller, independent functions. The specific functions and their execution time ratios are shown in Table 2. The local map update consists of four main functions: Map Cleanness, which cleans up the original local map; Keyframe Insertion, which stores new keyframes along with environment information; and Reference Update and Lastframe Update, which updates the position information of the reference keyframe and the last frame, respectively. Our analysis shows that Keyframe Insertion is the most time-consuming part of the update process. Based on the analysis, we separate Keyframe Insertion and Reference Update into a dedicated Local Map Update module, running in parallel with the Tracking module. Meanwhile, Map Cleanness and Last Frame Update functions stay in the Tracking module to prevent thread conflicts. In addition to threading separation, we introduce a Staging Map, a dedicated storage unit for local map updates. This ensures that the local map update does not interfere with other system operations. The Local Map Update module performs two main tasks: it stores map data from the edge server into the staging map, inserts keyframe information, and updates reference data for the Tracking module, allowing uninterrupted tracking. Once the new map is ready, the Tracking module updates the last frame's position based on the staging map and cleans up the original map. The staging map is used for tracking, while the original map becomes the new staging map. Since Lastframe Update cannot

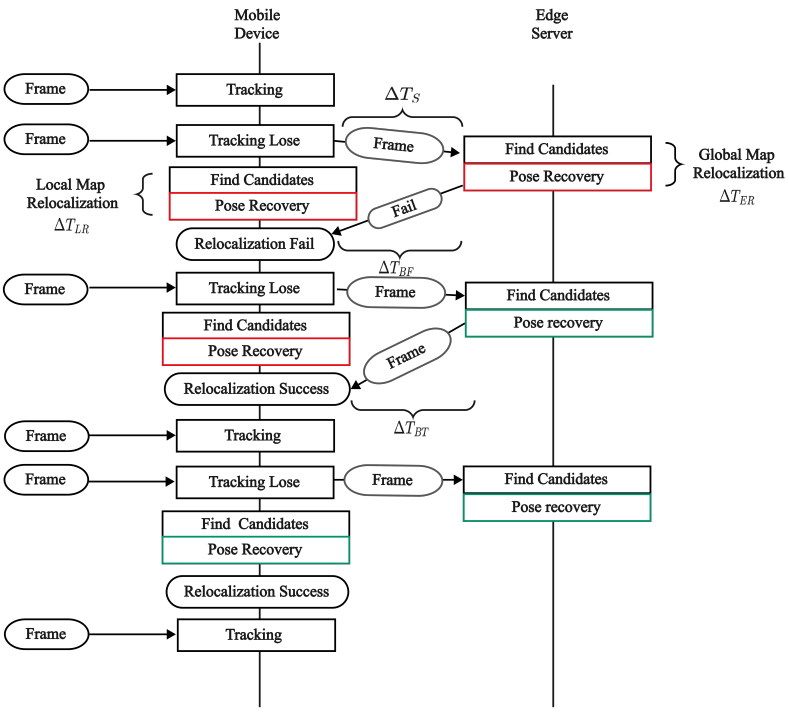

Figure 5: Proposed relocalization mechanism. Green means recovery is successful, and red means recovery fails.

run in parallel, the interruption time due to local map updates is now limited to Map Cleanup and Lastframe Update, which account for just 12% of the original interruption time.

After separating the local map update, we find that a lot of keyframe information is still lost during the update. As shown in Fig. 4, the updated map misses the yellow keyframes marked. Since these recent keyframes are critical for calculating position information, losing them significantly reduces the localization accuracy. Based on the separation of local map update threads, we design a keyframe backup mechanism to ensure the updated map contains more keyframe information. We redesign the functional dependencies to ensure that the new mechanism operates independently from the Local Map Update module. The selected keyframe information is inserted into the new map during the local map update and synchronized accordingly. The keyframes marked in blue in Figure 4 are inserted into the Staging Map that is currently being updated.

3.3 Relocalization Mechanism Improvement

To improve the efficiency of relocalization, we migrate the relocalization algorithm (including candidate frame selection and pose recovery) from the mobile device to the edge server. As shown in Figure 5, when relocalization occurs, the mobile device serializes the current frame and transmits it to the edge server as a string, with a data size of about 250kb. The server then performs relocalization using its global map. If the relocalization fails, a fixed-size flag string (75KB) is returned to notify the mobile device. If successful, the server returns the relocalized frame along with its reference keyframe; both serialized into larger strings totaling approximately 800KB. This remains significantly smaller than the complete relocalization map required by the Edge-SLAM mechanism, which typically exceeds 5MB in size. At the same time, the mobile device also attempts local relocalization using its local map. If the local relocalization succeeds before receiving the server result, it immediately resumes normal tracking, thereby reducing idle waiting time. In addition, since the relocalization map is no longer transmitted to the mobile device, the system's memory consumption on the mobile device is also reduced. Another key design element is the use of dedicated communication

Table 3: Relocalization Execution Process Evaluation

Process	ΔT_{LR}	ΔT_{ER}	$\Delta T_S + \Delta T_{BF}$	$\Delta T_S + \Delta T_{BT}$
Execution time(ms)	55	11	47	168

channels to prevent network congestion. Following the architectural principles of Edge-SLAM, we assign separate TCP-based ports to each type of operation between the mobile device and the edge server (i.e., local map updates, keyframe creation, and the novel relocalization process). In addition, in order to ensure relocalization efficiency, the local map update and keyframe creation channels are suspended during the relocalization phase. This approach effectively eliminates conflicts caused by concurrent transmissions, ensures uninterrupted data flow, and maintains the responsiveness of mobile devices throughout the relocation process with minimal additional latency.

To verify the practicality of the mechanism, we measure the execution and transmission delays under a typical home network (see Section 4.1 for details). As shown in Figure 5, ΔT_{LR} represents the time required for mobile-side local map relocalization, and ΔT_{ER} represents the time required for server-side global map relocalization. The whole process includes the selection of candidate nodes and posture recovery. ΔT_S represents the data transmission time from the client to the server, while ΔT_{BF} and ΔT_{BT} represent the response time in the case of relocalization failure and success, respectively. The empirical results (see Table 3) demonstrate that server-side global map relocalization runs significantly faster than local map relocalization due to the server's greater computational power when global map relocalization fails (i.e., ΔT_S plus ΔT_{BF} in Table 3), the overall latency increases only slightly (by about 5%) owing to the extremely small size of the returned data. When relocalization succeeds (i.e., ΔT_S plus ΔT_{BT} in Table 3), the transmission time increases (up to 168 ms) due to the larger volume of response data, occasionally causing minor frame drops. Nevertheless, because the mobile device immediately exits the relocalization state upon receiving the result from the server, this transient delay does not have a lasting impact on system performance. Furthermore, since the transmitted data is currently uncompressed, there remains room for further optimization of transmission efficiency.

4 Evaluation

4.1 Experimental Environment

We built the following experimental environment to evaluate the improvements brought by our proposed optimization scheme. The edge server comprises a MacBook Pro with an Apple M1 Pro chip running Ubuntu 22.04 LTS, featuring an 8-core CPU and 32 GB of RAM. The mobile device, with limited hardware resources, is a Raspberry Pi 4 running Ubuntu 22.04 LTS, equipped with a 4-core CPU and 4 GB of RAM. In our experimental setup, both the mobile device and the edge server were connected to the same local Wi-Fi network (IEEE 802.11ac), simulating a typical home environment. This configuration mirrors a practical edge computing scenario, such as those found in smart homes or indoor robotics applications. Data transmission between the two devices was facilitated using the Transmission Control Protocol (TCP), ensuring reliable and orderly communication over the local area network (LAN). Using the standard network diagnostic tools, iperf3 and ping, we measured an average bandwidth of approximately 50 Mbps, an average latency of less than 10 milliseconds, and observed no packet loss. It is worth noting that this evaluation was conducted under a single-device setup and does not account for potential network congestion that may occur when multiple mobile clients simultaneously access the edge server.

For evaluation, we utilize four widely used TUM benchmark datasets [27]: fr2_desk_with_person ("Desk"), fr2_pioneer_slam2 ("Pioneer2"), fr2_pioneer_slam3 ("Pioneer3"), fr2_pioneer_360 ("FP_360"). These datasets collectively represent a range of motion patterns and environmental complexities for evaluating system robustness and relocalization performance. Each sequence contains synchronized color and depth images: the RGB images are stored in PNG format at a resolution of 640×480 pixels

Table 4: System Robustness Evaluation

Dataset	***System***	***Average Interruption(ms)***	***Coverage(%)***
Desk	Edge-SLAM	473.6	86.9
	Multithread	53.2	97.2
	Relocate	446.3	91.6
	Multithread-RL	**51.4**	**98.2**
Pioneer2	Edge-SLAM	1613.7	64.5
	Multithread	96.8	93.2
	Relocate	910.4	88.3
	Multithread-RL	**93.2**	**94.6**
Pioneer3	Edge-SLAM	1167.5	86.7
	Multithread	74.4	97.4
	Relocate	745.5	90.3
	Multithread-RL	**75.1**	**97.5**

with 8-bit per channel(RGB), while the depth images are 16-bit single-channel PNGs at the same resolution. The "Desk" sequence involves limited camera motion with dynamic foreground interference (e.g., a moving person), typically without causing tracking failure, and is thus suitable for assessing performance under mild dynamic disturbances. "Pioneer2" features structured indoor navigation without forming a closed loop, providing a balanced scenario for evaluating localization stability in moderately complex environments. "Pioneer3" includes longer trajectories and transitions across visually diverse regions, making it suitable for testing relocalization capabilities in large-scale and semantically varied scenes. In contrast, "FP_360" focuses on pure rotational motion with repeated 360-degree turns, offering a challenging case for evaluating the system's robustness to large angular changes and loop closure detection. These sequences constitute a comprehensive benchmark suite for systematically evaluating SLAM performance across diverse and realistic operational scenarios. To evaluate the robustness of the system, we used the rosbag playback method [28], with a fixed playback rate of 30 frames per second for continuous frame streaming. This consistent rate simulated real-time operation. If the mobile device loses tracking due to the local map updates, subsequent frames are discarded. Our evaluation primarily covers four aspects: system robustness, relocalization recovery time, system accuracy, and CPU and memory usage. For comparison, ORB-SLAM2 was excluded from the robustness evaluation because it does not include the local map update. However, for all other metrics, we compare our proposed method with ORB-SLAM2 and Edge-SLAM.

4.2 Robustness Improvement

First, we evaluate the system's robustness in our proposed solution. The coverage C_R, which is related to the total number of frames N_{TF} and the number of frames processed by the system N_{PF}, is a critical metric for the robustness of SLAM systems [10]. Here, N_{TF} represents the total number of frames in the dataset, while N_{PF} denotes the number of frames successfully processed by the system after running through the entire dataset. This relationship can be expressed as shown in equation (2):

$$C_R = \frac{N_{PF}}{N_{TF}} \tag{2}$$

In our evaluation, "Multithread" refers to our parallel thread update scheme, while "Relocate" represents the relocation optimization scheme. "Multithread-RL" denotes the combined scheme integrating both approaches. The evaluation results are shown in Table 4. Both of our proposed solutions enhance the robustness of the system, with the parallel local map update solution showing particularly significant improvements across all three datasets. For the relatively simple "Desk" dataset, where no tracking loss occurred, the integrated method reduced the average tracking interruption time caused by local map updates from 473.6 ms to 51.4 ms, while trajectory coverage

Table 5: Relocalization Recovery Time Evaluation

Dataset	***System***	***Time used for Relocation(s)***
Pioneer2	ORB-SLAM2	67.61
	Edge-SLAM	79.35
	Multithread	72.39
	Relocate	69.52
	Multithread-RL	**67.67**
Pioneer3	ORB-SLAM2	14.25
	Edge-SLAM	21.86
	Multithread	17.71
	Relocate	14.06
	Multithread-RL	**14.17**
FP_360	ORB-SLAM2	35.49
	Edge-SLAM	45.82
	Multithread	42.06
	Relocate	39.13
	Multithread-RL	**40.23**

increased from 86.9% to 98.2%. In more complex datasets such as "Pioneer2" and "Pioneer3," tracking failures frequently triggered the transfer of keyframes to the mobile device for local map updates, leading to severe tracking interruptions in Edge-SLAM. In contrast, our relocalization mechanism effectively mitigated this issue by suspending local map updates during periods of tracking loss. For example, on the "Pioneer2" dataset, the average interruption time during relocalization was reduced from 1613.7 ms to 910.4 ms, representing a reduction of up to 43.6%. Our parallel local map update mechanism achieved even greater robustness improvements. The integration of both optimization strategies results in a significant 30.1% increase in trajectory coverage on the "Pioneer2" dataset. Similarly, we also observed substantial improvements in the "Pioneer3" dataset. These results demonstrate the effectiveness of our proposed mechanisms in enhancing system robustness across varying levels of scene complexity.

4.3 Relocalization Recovery Evaluation

We adopt relocalization recovery time as an evaluation metric to assess the system's ability to regain tracking after a failure. This metric is commonly used in SLAM evaluations to measure the efficiency of relocalization processes [26, 29]. It represents the duration required for the system to successfully resume normal tracking after tracking is lost. In our experiments, we measured the total recovery time over the entire runtime of each dataset. The results are presented in Table 5. Compared with Edge-SLAM, our proposed relocalization mechanism significantly reduces the recovery time and closes the performance gap with ORB-SLAM2. On the more complex "Pioneer3" dataset, for example, the recovery time was reduced from 21.86 s to 14.06 s—a 35.7% improvement. Moreover, our relocalization mechanism integrates well with the parallel thread-based local map update strategy, maintaining consistently strong relocalization performance across different system configurations.

4.4 System Accuracy Optimization

We also evaluate the system's localization accuracy using two widely adopted metrics: Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) [10]. ATE is commonly used to assess the overall accuracy of visual SLAM systems, while RPE is particularly effective in quantifying drift. RPE is divided into the components of translational (RPE.t) and rotational (RPE.r). We utilize the official evaluation scripts and compute the Root Mean Square Error (RMSE) for ATE and RPE.

Table 6: System Accuracy Evaluation

Dataset	*System*	*ATE(cm)*	*RPE.t (cm)*	*RPE.r (deg)*
Pioneer2	ORB-SLAM2	6.17	10.25	1.87
	Edge-SLAM	7.37	11.63	2.17
	Multithread	5.43	8.65	1.69
	Relocate	5.13	9.06	1.89
	Multithread-RL	**4.69**	**7.51**	**1.56**
Pioneer3	ORB-SLAM2	5.56	12.21	2.99
	Edge-SLAM	5.79	10.66	2.53
	Multithread	4.87	8.84	2.04
	Relocate	5.05	8.89	2.12
	Multithread-RL	**4.82**	**8.33**	**1.89**
FP_360	ORB-SLAM2	2.24	4.71	1.77
	Edge-SLAM	2.23	4.78	1.84
	Multithread	1.96	3.97	1.53
	Relocate	2.21	3.77	1.71
	Multithread-RL	**1.33**	**2.69**	**1.19**

We conducted 20 runs on three standard TUM datasets to ensure statistical reliability. The "Desk" dataset was excluded from this evaluation because no tracking loss occurred, making it unsuitable for observing the effectiveness of the proposed relocalization mechanism. The experimental results are summarized in Table 6. Compared with Edge-SLAM, both of our proposed strategies significantly reduce localization errors across all three datasets, especially in complex scenes or those involving large rotational motions. For instance, on the "FP_360" dataset, our parallel local map update scheme reduced the rotational error (RPE.r) from 1.84° to 1.19°, a 35.3% improvement. This aligns with our expectations, as the system preserves critical keyframe information during rotation. In addition, the relocalization mechanism further enhanced localization accuracy, particularly in ATE and RPE.t, by ensuring better tracking recovery. However, since the system suspends frame processing during local map updates, the RPE.r remains relatively higher. When both strategies are integrated, the system substantially improves localization accuracy. Specifically, on the "FP_360" dataset, ATE is reduced from 2.24 cm to 1.33 cm (a 40.4% decrease), and RPE.t is reduced from 4.71 cm to 2.69 cm (a 43.7% decrease), compared to Edge-SLAM.

4.5 CPU and Memory Consumption Analysis

To evaluate the CPU and memory consumption on mobile device, we observe the CPU and memory usage during system operation. We compare our integrated solution with Edge-slam and orb-slam2, respectively. Figure 6 illustrates the comparison of CPU usage. The straight lines in the figure represent the average CPU usage on the mobile device throughout the SLAM process. As shown in Figure 6, due to the addition of parallel threads, the CPU usage of our proposed system is slightly higher than that of Edge-SLAM, with the average CPU usage increasing from 53.9% (Edge-SLAM) to 60.7% (proposed system). However, compared to ORB-SLAM2, the CPU usage of our system remains significantly lower, with a 30.4% difference. Figure 7 presents the comparison of memory usage. As we can see, the memory usage of ORB-SLAM2 increases continuously over time, with the system occupying more memory as it runs. In contrast, both our proposed system and Edge-SLAM show very stable memory usage with minimal difference between them, as the staging map contains only a small number of keyframes. Moreover, our proposed relocalization mechanism no longer requires receiving the relocalization map, further reducing memory consumption during the process.

In conclusion, our solution provides notable improvements in robustness and localization accuracy despite the slight increase in CPU usage. Furthermore, we observe that all Visual SLAM systems

in the experiment suffer a significant decline in processing efficiency over extended periods. This performance degradation is primarily due to insufficient swap memory, which decreases as the system operates.

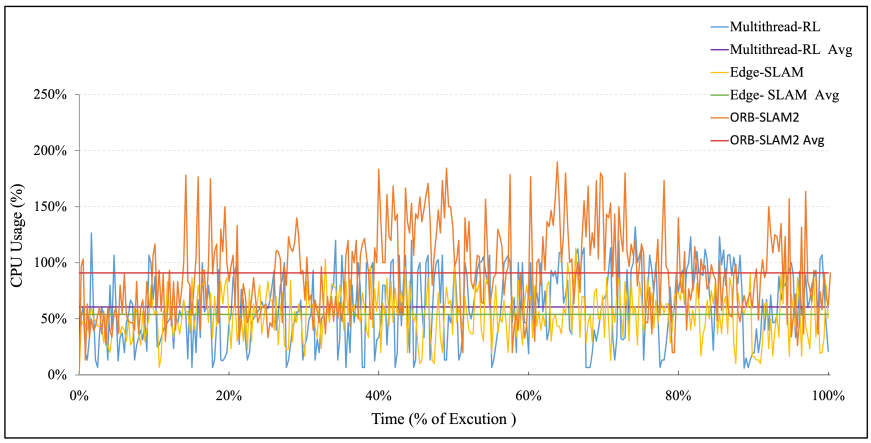

Figure 6: Comparison of CPU Consumption in Visual SLAM System

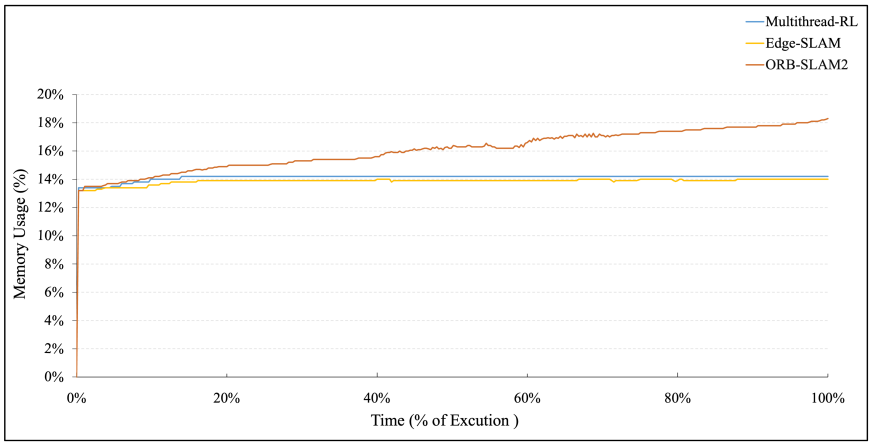

Figure 7: Comparison of Memory Consumption in Visual SLAM System

5 Conclusion

For edge-assisted Visual SLAM, rapid data synchronization between the mobile device and the edge server presents a significant challenge. Additionally, the distributed architecture also affects the system's ability to recover after a tracking loss; this weakens the system's robustness and impacts its accuracy. To address this, we separated the data synchronization process on the mobile device into an independent thread and processed it in parallel while backing up keyframe information to compensate for the accuracy loss during synchronization. Furthermore, we proposed an efficient relocalization mechanism that can help the system restore tracking more effectively. Overall, our approach reduces the tracking interruption time caused by local data synchronization on the mobile device by up to 94.2%, improves coverage by up to 30.1%, reduces the time required for system relocalization recovery by up to 35.2%, and increases the system's localization accuracy to 43.7% and 36.8% in translation and rotation scenarios, respectively.

By evaluating the system's CPU and memory usage, we found that the exchange memory usage in the edge-assisted Visual SLAM architecture is quite high, and insufficient exchange memory space

negatively affects the system's execution efficiency. There is also room for further improving data exchange efficiency between the mobile device and the edge server. In future work, we plan to conduct a more in-depth analysis and evaluation of these issues and aim to further reduce resource consumption and the mobile device's end-to-end load through hardware and software optimizations.

Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Number JP22H05000, JST CREST JPMJCR23A5, and JSPS ASPIRE JPMJAP2411.

References

[1] Charalambos Theodorou, Vladan Velisavljevic, Vladimir Dyo, and Fredi Nonyelu. Visual slam algorithms and their application for ar, mapping, localization and wayfinding. *Array*, 15:100–222, 2022.

[2] Andréa Macario Barros, Maugan Michel, Yoann Moline, Gwenolé Corre, and Frédérick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 11(1):24, 2022.

[3] Jiansheng Peng, Yaru Hou, Hengming Xu, and Taotao Li. Dynamic visual slam and mec technologies for b5g: a comprehensive review. *EURASIP Journal on Wireless Communications and Networking*, 2022(1):98, 2022.

[4] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[5] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849, 2014.

[6] Sebastian Eger, Rastin Pries, and Eckehard Steinbach. valuation of different task distributions for edge cloud-based collaborative visual slam. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020.

[7] Peter Sossalla, Johannes Hofer, Christian Vielhaus, Justus Rischke, and Frank HP Fitzek. Offloading visual slam processing to the edge: An energy perspective. In *2023 International Conference on Information Networking (ICOIN)*, pages 39–44, 2023.

[8] Jingao Xu, Zheng Yang, Yunhao Liu, and Hao Cao. Edge assisted mobile visual slam, 2024.

[9] Peter Sossalla, Johannes Hofer, Justus Rischke, Christian Vielhaus, Giang T Nguyen, Martin Reisslein, and Frank HP Fitzek. Dynnetslam: Dynamic visual slam network offloading. *IEEE Access*, 10:116014–116030, 2022.

[10] Timothy Chase, Ali J Ben Ali, Steven Y Ko, and Karthik Dantu. Pre-slam: Persistence reasoning in edge-assisted visual slam. In *2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 458–466, 2022.

[11] Ali J Ben Ali, Marziye Kouroshli, Sofiya Semenova, Zakieh Sadat Hashemifar, Steven Y Ko, and Karthik Dantu. Edge-slam: Edge-assisted visual simultaneous localization and mapping. *ACM Transactions on Embedded Computing Systems*, 22(1):1–31, 2022.

[12] Shuai Liu, Dongye Liu, Gautam Srivastava, Dawid Poap, and Marcin Woźniak. Overview and methods of correlation filter algorithms in object tracking. *Complex & Intelligent Systems*, 7:1895–1917, 2021.

[13] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[14] Swarnava Dey and Arijit Mukherjee. Robotic slam: a review from fog computing and mobile edge computing perspective. In *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services*, pages 153–158, 2016.

[15] Victor Kathan Sarker, J Peña Queralta, Tuan Nguyen Gia, Hannu Tenhunen, and Tomi Westerlund. Offloading slam for indoor mobile robots with edge-fog-cloud computing. In *2019 1st international conference on advances in science, engineering and robotics technology (ICASERT)*, pages 1–6, 2019.

[16] Kwame-Lante Wright, Ashiwan Sivakumar, Peter Steenkiste, Bo Yu, and Fan Bai. Cloudslam: Edge offloading of stateful vehicular applications. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 139–151, 2020.

[17] Peter Sossalla, Johannes Hofer, Justus Rischke, Johannes Busch, Giang T Nguyen, Martin Reisslein, and Frank HP Fitzek. Optimizing edge slam: Judicious parameter settings and parallelized map updates. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 1954–1959, 2020.

[18] Brian Williams, Georg Klein, and Ian Reid. Automatic relocalization and loop closing for real-time monocular slam. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1699–1712, 2011.

[19] Hauke Strasdat, Andrew J Davison, JM Martìnez Montiel, and Kurt Konolige. Double window optimisation for constant time visual slam. In *2011 international conference on computer vision*, pages 2352–2359, 2011.

[20] Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–853, 2014.

[21] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[22] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[23] Jeremy Straub, Sebastian Hilsenbeck, Georg Schroth, Robert Huitl, Andreas Möller, and Eckehard Steinbach. Fast relocalization for visual odometry using binary features. In *2013 IEEE International Conference on Image Processing*, pages 2548–2552, 2013.

[24] Atsunori Moteki, Nobuyasu Yamaguchi, Ayu Karasudani, and Toshiyuki Yoshitake. Fast and accurate relocalization for keyframe-based slam using geometric model selection. In *2016 IEEE Virtual Reality (VR)*, pages 235–236, 2016.

[25] Johannes Hofer, Peter Sossalla, Justus Rischke, Christian L Vielhaus, Martin Reisslein, and Frank HP Fitzek. Circular frame buffer to enhance map synchronization in edge assisted slam. In *ICC 2023-IEEE International Conference on Communications*, pages 210–215, 2023.

[26] Pengfei Zhang, Huaimin Wang, and Bo Ding. Using collaborative sharing on cloud for fast relocalization in keyframe-based slam. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 291–296, 2017.

[27] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580, 2012.

[28] Gang Peng, Tin Lun Lam, Chunxu Hu, Yu Yao, Jintao Liu, and Fan Yang. Connecting the robot to ros. In *Introduction to Intelligent Robot System Design: Application Development with ROS*, pages 41–137, 2023.

[29] Zongqian Zhan, Wenjie Jian, Yihui Li, and Yang Yue. A slam map restoration algorithm based on submaps and an undirected connected graph. *IEEE Access*, 9:12657–12674, 2021.