Edge Assisted Mobile Semantic Visual SLAM

Hao Cao, Jingao Xu, Danyang Li, *Student Member, IEEE*, Longfei Shangguan, *Member, IEEE*, Yunhao Liu, *Fellow, IEEE*, and Zheng Yang, *Fellow, IEEE*

Abstract—Localization and navigation play a key role in many location-based services and have attracted numerous research efforts. In recent years, visual SLAM has been prevailing for autonomous driving. However, the ever-growing computation resources demanded by SLAM impede its applications to resource-constrained mobile devices. In this paper, we present the design, implementation, and evaluation of *edgeSLAM*, an edge-assisted real-time semantic visual SLAM service running on mobile devices. *edgeSLAM* leverages the state-of-the-art semantic segmentation algorithm to enhance localization and mapping accuracy, and speeds up the computation-intensive SLAM and semantic segmentation algorithms by computation offloading. The key innovations of *edgeSLAM* include an efficient computation offloading strategy, an opportunistic data sharing method, an adaptive task scheduling algorithm, and a multi-user support mechanism. We fully implement *edgeSLAM* and plan to open-source it. Extensive experiments are conducted under 3 datasets. The results show that *edgeSLAM* can run on mobile devices at 35fps and achieve 5cm localization accuracy from real-world experiments, outperforming existing solutions by more than 15%. We also demonstrate the usability of *edgeSLAM* through 2 case studies of pedestrian localization and robot navigation. To the best of our knowledge, *edgeSLAM* is the first edge-assisted real-time semantic visual SLAM for mobile devices.

Index Terms—Indoor localization, real-time, edge computing, semantic visual SLAM

1 Introduction

Indoor localization and navigation play a key role in many location-based services. However, due to the excessive signal attenuation and multi-path propagation [1], [2], [3], [4], the Global Positioning System (GPS) fails to achieve the desired accuracy and thus cannot be adopted for this purpose in most indoor scenarios. While the innovations on RF-based indoor localization techniques (e.g., Wi-Fi, RFID, and Bluetooth) are going full steam ahead [5], [6], [7], [8], [9], [10], few of these solutions are mature for real-world deployment, either because of the low accuracy or high infrastructure cost.

As another promising alternative, vision-based indoor localization techniques, in particular, visual simultaneous localization and mapping (visual SLAM), attracts more attention in recent years [11], [12]. Visual SLAM utilizes a sequence of images captured by a camera and inertial measurement unit (IMU) readings to build the map of the ambient environment and estimate the current location of the camera itself. Compared with RF-based solutions, visual SLAM achieves an order of magnitude higher localization accuracy (5cm) at the minimal infrastructure cost as camera and IMU units have become the standard components of mobile devices on today's market [13]. As research progresses, more SLAM-enabled applications arise for urban modeling[14], express delivery[15], and industrial inspection [16]. In those scenarios, mobile devices also have to perform visual SLAM in real-time for a better localization performance and scheduling of subsequent tasks.

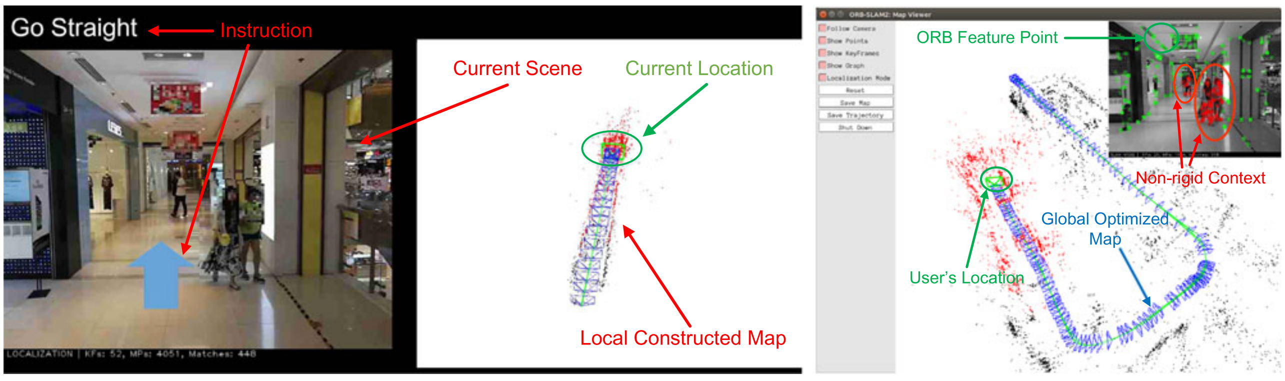
Although the evolving hardware and software of mobile devices (e.g., Samsung Galaxy S10 smartphone is equipped with three HD cameras) guarantee the image quality and IMU sensor precision for fine-grained visual SLAM, the computational resources on mobile devices, unfortunately, are still insufficient to meet the visual SLAM's ever-growing computation demand. For example, as we experimentally demonstrated, even ORB-SLAM2 [12], a lightweight, functionally limited visual SLAM system, still cannot work in real-time (i.e., $\leq$15 fps) on the latest smartphone (e.g., Samsung Galaxy S10 and Google Pixel 2). Straightforwardly applying visual SLAM to mobile scenarios, on the other hand, cannot achieve good performance because of highly dynamic environmental changes (e.g., walking customers in a shopping mall).

Recently, two new opportunities have arisen in the design of real-time visual SLAM on mobile devices:

1) The emerging paradigm of edge computing [17], [18], as well as advanced wireless technology (5G [19] and Wi-Fi 802.11ad standard [20]), is powerful for solving computation-intensive tasks locally and in real-time. It is thus possible to speed up the visual SLAM by offloading the workload to an edge server.
2) The evolving computer vision (CV) techniques (e.g., Mask-RCNN [21]) now can recognize objects in an image with very high accuracy. Therefore, it is

- *Hao Cao, Jingao Xu, Danyang Li, Yunhao Liu, and Zheng Yang are with the School of Software and BNRist, Tsinghua University, Beijing 100084, China. E-mail: {ihaocao, xujingao13, lidanyang1919, yunhaoliu, hmilyyz}@gmail.com.*
- *Longfei Shangguan is with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 USA. E-mail: shanggdlk@gmail.com.*

Manuscript received 16 November 2021; revised 13 August 2022; accepted 17 August 2022. Date of publication 24 August 2022; date of current version 3 November 2023.
This work was supported in part by National Key Research Plan under Grant 2018YFB2100300, and in part by NSFC under Grants 61832010, 61972131 and 61872081.
(Corresponding author: Zheng Yang.)
Digital Object Identifier no. 10.1109/TMC.2022.3201000



(a) (b)

Fig. 1. User interface of *edgeSLAM*. (a) Mobile client: Navigation instruction and local constructed map are displayed on mobile device.[1] (b) Edge server: Optimized global map and user's location are displayed.

possible to analyze semantic information from captured video and improve the SLAM performance.

However, realizing these intuitions is non-trivial and faces four significant challenges:

- *Task decomposition*. Simply transmitting all images to the edge server is not feasible since it introduces excessive bandwidth cost and transmission delays (details in Section 2.3). Partitioning both visual SLAM algorithm and semantic segmentation algorithm into task units is also non-trivial as their function units are tightly coupled. As shown in Fig. 2, an improper partition may result in redundant data storage, exchange, and most importantly, system delay, which in turn increases the algorithm latency.
- *Task cooperation*. The visual SLAM and semantic segmentation algorithms are generally regarded as two independent tasks without reusing intermediate results. However, to achieve both low latency and high accuracy, it is beneficial to share the intermediate results between these two algorithms so that redundant computations can be eliminated or minimized.
- *Task scheduling*. The computation resources on the mobile device and edge server is highly unbalanced. Meanwhile, the wireless link between these two parts also varies from time to time. Task scheduling strategy should be adaptive to the dynamics in computation resources and wireless link quality.
- *System scalability*. As the scenario scales, an edge server typically needs to serve multiple mobile clients. The expanded map data to be transferred, video frames to be inferred, and redundant map elements to be optimized inevitably hurt the system scalability.

In this paper, we present the design and implementation of *edgeSLAM*, a real-time *edge* assisted semantic visual *SLAM* service running on commercial mobile devices. *edgeSLAM* leverages the state-of-the-art semantic segmentation algorithm Mask-RCNN [21] to improve SLAM accuracy and speed up the SLAM and semantic segmentation algorithm by efficient computation offloading and data sharing, and adjust the offloading strategy automatically to adapt to the wireless link conditions, and make full use of the spatial relationship between clients to save the resources for enabling the multi-user support.

To find out the optimal task decomposition strategy, we take the operation time, memory overhead, and the transmission delay of each functional module into consideration and conduct extensive experiments to profile the performance of each module. We further analyze dependencies among these functional modules and determine the ”hourglass position” to decompose the visual SLAM and object detection algorithm.

To minimize the latency introduced by redundant data transmission, we leverage that the scenes in most consecutive frames are similar. For example, the same signboard tends to appear in multiple consecutive frames. *edgeSLAM* avoids per-frame object segmentation operation and reuses the previous result from the last object segmentation until a significant frame change is detected on mobile devices.

To accommodate dynamic link conditions, we design a probing-optimizing strategy that first probes the network conditions and then leverages such information to optimize our task scheduling mechanism.

To scale up the system, we design a set of algorithms to leverage the spatial relationship between the client maps and reuse the intermediate results on the edge server, thus archiving both real-time running of the clients and saving the computation and storage resources to enlarge the server capacity.

We fully prototype *edgeSLAM's* server on an Ubuntu edge server and the client on three different types of mobile devices, including an Nvidia Jetson TX2 development board, a Samsung Galaxy S10 and an Apple iPhone X. The interface of *edgeSLAM* is shown in Fig. 1. Comprehensive experiments are carried out under different network conditions, such as Wi-Fi 5G, Wi-Fi 2.4G, and Cellular 4G. We also examine *edgeSLAM* on two official datasets (TUM [23] and KITTI [24]) and a self-labeled dataset from three buildings. The results demonstrate that *edgeSLAM* can achieve average 35fps frame rate with 5cm localization accuracy and 2% relative mapping error in all scenarios, which outperforms existing systems by more than 10%. Our two case studies further demonstrate that *edgeSLAM* achieves outstanding performance in pedestrian localization task with

1. The design of user interface for mobile part adapted from our previous work *Pair-Navi* [22].

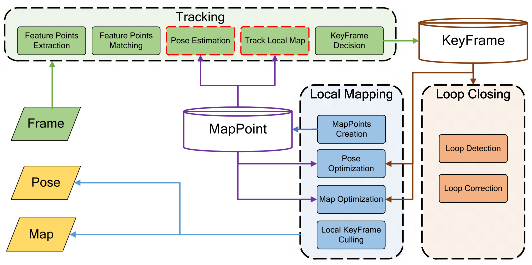


Fig. 2. Typical Visual SLAM [12] architecture.

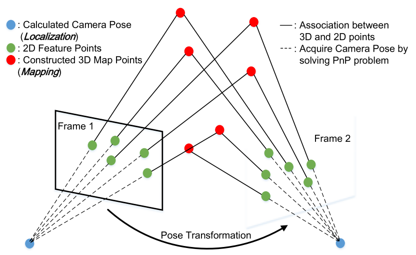

● : Calculated Camera Pose (Localization)
● : 2D Feature Points
● : Constructed 3D Map Points (Mapping)
— : Association between 3D and 2D points
--- : Acquire Camera Pose by solving PnP problem
Frame 1
Frame 2
Pose Transformation

Fig. 3. Illustration of Visual Odometry.

average 9.6cm accuracy and robot navigation task with 92.3% navigation success rate.

The key contributions are summarized as follows:

- We design and implement a complete edge-assisted real-time SLAM system on smartphones. To the best of our knowledge, this is the first time that *multi-user mobile semantic visual SLAM* can work in real-time on mobile devices.
- We measure each functional module's operation time and memory overhead in visual SLAM and semantic segmentation, and ascertain the optimal decoupling position and disassembly method to deeply fuse SLAM and semantic segmentation and determine task assignment between mobile and edge. To make the system both adaptive in fluctuated network conditions and fully scalable in multi-user cases, we adopt an adaptive method to adjust system parameters dynamically and leverage the spatial relationship between clients.
- We extensively evaluate the performance of *edge-SLAM* on under 3 datasets. The results show that *edgeSLAM* achieves satisfying results in all scenarios.
- We open-source *edgeSLAM* [1] to facilitate the community to develop multi-user gaming, collaborative drone mapping and other promising mobile applications based on real-time visual SLAM.

The rest of this paper is organized as follows. We present the background and related works, as well as introduce the motivation of our work in Section 2. Followed by an overview of *edgeSLAM* in Section 3. Key strategies and techniques are presented in Section 4, followed by the dynamic design of self-adaptation strategy in Section 4.4 and multi-user support mechanism in Section 5. Experiments setup, results and analysis are presented in Section 6. We finally conclude our work in Section 7.

2 Background and Motivation

2.1 Visual SLAM and Semantic Visual SLAM

Simultaneous Localization And Mapping (SLAM) consists of the concurrent construction of a map model and the estimation of the state of the robot moving within it. As the vision-based sensors get ubiquitous, the SLAM that leverages visual information has become the dominant method for robot localization and mapping [13].

In recent years, *ORB-SLAM* [25], the state-of-the-art monocular visual SLAM work has become a widely used system by both academic research and industrial applications for its modular design and high accuracy. Further, *ORB-SLAM* has become the representative design and base system of the Visual SLAM community. As the Fig. 2 depicts, the design of Visual SLAM represented by *ORB-SLAM* consists of three components:

- *Tracking* module estimates the coarse-grained pose of the shooting camera based on the consecutive video frames. When a new video frame arrives, the tracking module will extract its 2D feature points and associate them with 3D map points [2] in storage. As illustrated in Fig. 3, the 2D-to-3D feature points matching will give us a rough camera pose on the current frame.
- *Local Mapping* module then creates new 3D map points via triangulation between two consecutive frames [26]. An optimized camera pose can then be obtained by solving a Bundle Adjustment problem. This module repeatedly runs as the camera takes more photos, resulting in a trajectory of the camera pose, a map of the 3D landmarks, and the corresponding keyframes[3].
- *Loop Closing* module compares the features extracted from a video frame with keyframes. If a keyframe is found to be similar enough to the input video frame, the loop closing module will then fine-tune the current camera pose and optimize the map construction based on the matching result.

While the conventional point-clouds-based map representation remains the most widely used method for SLAM, higher-level representations, including objects and shapes, have been proposed with the development of computer vision. Semantic mapping consists in associating semantic concepts to geometric entities in a robot's surroundings[13]. DS-SLAM[27] employs SegNet[28] and OctoMap[29] to generate dense 3D semantic map. Further, Node-SLAM[30] achieves an object-level system with a shape inference algorithm. Recently, Kimera[31] and Kimera-Multi[32] proposed a novel representation that contains spatial concepts at different levels of abstraction. However, the computational requirements and resource demands of dense map building and object-level representation are considerably high and too demanding for resource-constrained mobile devices.

Semantic segmentation is also used to remove the influence of dynamic objects in addition to creating semantic

1. Code and data at https://github.com/MobiSense/edgeSLAM.

2. Map points represent discrete 3D landmarks in the global coordinate.

3. Keyframes are selected frames indicating poses and positions of the corresponding camera.

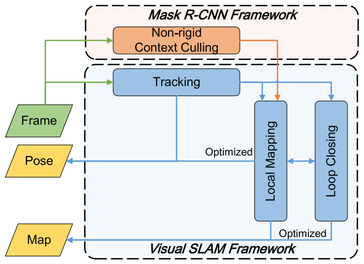


Fig. 4. Semantic Visual SLAM system architecture. Figure referred from [22].

maps. Recent studies [33], [34], [35] have attempted to incorporate semantics into visual SLAM to improve robustness in time-varying environments, such as moving objects and pedestrians. A prior work [36] uses R-CNN [37] to classify region proposals, which were subsequently refined by category-specific coarse mask predictions. As shown in Fig. 4, recent work [22] uses Non-rigid Context Culling (NRCC) with Mask R-CNN [21], the state-of-the-art work for semantic segmentation, to improve the localization and mapping accuracy, which extracts the temporal objects (such as pedestrians) on each video frame to minimize their negative effect on both localization and mapping.

2.2 Edge-Assisted Strategy

Continuous vision analysis or vision-based applications like VR/MR can be enabled by partially offloading computation-intensive tasks to cloud or edge cloud infrastructures. However, most of the recent works [38], [39], [40] leverage the edge-assisted strategy to only focus on the relative facile object detection task. While we are riveted to semantic visual SLAM, either segmentation or localization task is insuperable on mobile devices and demonstrated to be more complicated than detection task.

Edge-Assisted SLAM. With the popularity of the edge-assisted paradigm in academia and industry, SLAM has also been applied to the paradigm recently. Recent works [41], [42], [43], [44] propose edge-assisted SLAM systems. However, both systems are either 2D SLAM or laser-based SLAM, which need much fewer data transmission and are less complex than tightly coupled Visual SLAM. Edge-SLAM[45] enables mobile devices to run visual SLAM in real-time. However, the design principles behind it and our work diverge. Edge-SLAM leverages the assistance of the edge server to solve the growing complexity of the SLAM system that runs on computation-constrained mobile devices. Thus, Edge-SLAM's client only uses an enhanced VO, which runs *Tracking* and part of *Local Mapping*; the server handles the rest. Due to the lightweight design of the client, the map on the client is periodically replaced with the one that the server optimized, interrupting the tracking and mapping process and resulting in poor localization and mapping accuracy. In the case of *edgeSLAM*, the clients are more independent because of the *Local Mapping* module, which allows the client to locate and generate the local map without the help of the server. Furthermore, the map updating process is non-destructive, which keeps the clients' original maps and only synchronizes them with the server-optimized ones. The goal of *edgeSLAM* is to find an optimal decoupling and re-assignment approach for seamlessly splitting the entire SLAM system into client and server, which can serve as an example and reference for other SLAM systems interested in adopting the edge-assisted paradigm.

Cooperative SLAM. Cooperative SLAM refers to multiple clients working together to solve the SLAM problem, which has been explored in recent works. In most cases, a powerful server is used to collect, fuse, and optimize the maps created by the clients. This paradigm shares some similarities with the edge-assisted SLAM system, whereas edge-assisted SLAM concentrates more on real-time performance because of the vicinity of the edge server. Cooperative SLAM systems, on the other hand, focus on solving collaborative mapping and localization with higher accuracies and less time/bandwidth. [46], [47] proposed cooperative SLAM systems with clients that only use Visual Odometry (VO). Although VO is sufficient for short-term localization, it is insufficient for long-term mapping due to cumulative drifts. [48] recently presented a collaborative system that runs *Tracking* and Local Mapping on the client, and only maintains the latest map with limited size. Map fusion and optimization are performed on the server. This work focuses on the collaborative mapping task on multiple Unmanned Aerial Vehicles (UAVs), which involves communication, data management, and information sharing between clients. Similarly, a recent study [49] also presented a collaborative SLAM system, focusing on data transmission minimization, service quality, and global map management. However, *edgeSLAM* addresses the optimal splitting strategy of the Semantic SLAM system, with the goal of achieving a real-time SLAM system at 30 fps in a fluctuating network environment.

2.3 Latency and Accuracy Analysis

Running semantic visual SLAM on mobile devices is challenging due to its extensive computation overhead. For example, the classical visual SLAM algorithm ORB-SLAM [25] works at a rate of only 10fps when running on the Samsung Galaxy S10 smartphone, which is far lower than the real-time requirement ($\geq$ 30 fps [22], [40]). Beyond the real-time issues, the lower processing frame rate actually reveals two shortcomings: (i) the efficiency of the SLAM task processing could not match the camera sampling rate, which eventually results in a misalignment between the displayed image and the localization result; and (ii) the SLAM tasks occupy a large amount of computing resources on mobile devices, hindering the devices from performing upper-layer tasks that could need additional computing power as SLAM is typically a service to identify the location or recognize a place. This frame rate will drop sharply when the semantic segmentation algorithm (e.g., Mask-RCNN) runs parallel to SLAM on mobile devices. Offloading the entire computation to the powerful edge server, on the other hand, may cause considerable latency. To better understand this transmission latency and its impact on the localization and object segmentation performance, we conduct experiments detailed below.

Latency Analysis. We model the end-to-end latency (from capturing a video frame until we obtain a camera pose) of a semantic visual SLAM solution as follows:

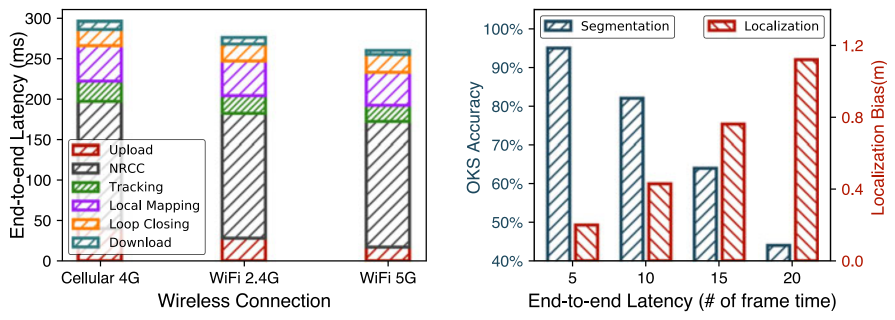


Fig. 5. Accuracy and Latency Analysis. (a) Localization and segmentation accuracy with respect to different end-to-end latency. (b) End-to-end latency with different wireless connection link.

$$
\begin{aligned}
t_{\mathrm{e2e}} &= t_{\mathrm{upload}} + t_{\mathrm{infer}} + t_{\mathrm{download}} \\
t_{\mathrm{infer}} &= \max(t_{\mathrm{NRCC}}, t_{\mathrm{T}}) + t_{\mathrm{LM}} + t_{\mathrm{LC}}
\end{aligned}
\quad (1)
$$

where t_{upload} and t_{download} represents the delay for uploading an image to an edge server and downloading the result to the mobile device, respectively. t_{infer} represents the delay of running semantic visual SLAM on an edge server, which consists of the time consumed for semantic segmentation t_{NRCC}, tracking t_{T}, local mapping t_{LM}, and loop closing module t_{LC}, respectively.

We measure the latency of each functional module in various wireless link connections and show the result in Fig. 5a. From the result, we can see that offloading the entire computation to an edge server introduces significant latency (280ms on Cellular 4G link, 270ms on Wi-Fi 2.4G link, and 260ms on Wi-Fi 5G link, respectively). In other words, the offloading-all mechanism only gets 4fps over all three wireless links. While SLAM and semantic segmentation can be further accelerated by leveraging a more powerful edge server, the uploading and downloading latency, however, still takes 48ms, which sets an upper bound of the achievable frame rate (20fps when ignoring the inference latency).

Accuracy Analysis. We further conduct an experiment to understand the impact of different delays on localization accuracy and semantic segmentation accuracy. In the experiments, we record the 3-D position of the camera reported by the visual SLAM algorithm and calculate the distance between the inference result and the ground-truth (measured by a laser ranger) using the bias Euler-distance metric. We also use the Object Keypoint Similarity (OKS) [50] metric to measure the accuracy of keypoints in each group in the object keypoint segmentation task.

Fig. 5b shows the localization and semantic segmentation accuracy as a function of end-to-end delay. We observe that the localization error maintains a very low level when the end-to-end latency is low (i.e., five frames delay). The localization error then reaches 1m as we increase the end-to-end delay to 20 frames (about 666ms). The segmentation accuracy shows a similar trend: it decreases from over 90% accuracy to less than 15% accuracy as we increase the end-to-end delay from 5 frames to 20 frames. On the one hand, the decreased segmentation accuracy impairs the effectiveness of the non-rigid context removal, causing degraded overall localization accuracy. On the other hand, the client can not get the timely optimized map due to increased latency, so the localization errors accumulate over time. This result demonstrates that the end-to-end latency has a significant impact on both two algorithms' performance.

3 System Architecture

To overcome these limitations, we propose *edgeSLAM*, a semantic visual SLAM system for mobile platforms, achieving both accuracy and real-time simultaneously with the help of edge computation resources. To reduce the latency caused by offloading the semantic SLAM task, *edgeSLAM* decouples the integrated visual SLAM process into two separate pipelines. At a high level, as shown in Fig. 6, these two parts are connected through a wireless link: *Mobile tracking and Local Mapping* parts are on the mobile devices (smartphone for people or development board for robots), and *Edge Optimization and Segmentation* part is on an edge side. The former pipeline tracks the pose of a camera and constructs a local map simultaneously. Parallelly, the latter pipeline segments image frames at the pixel-level, optimizes the roughly estimated pose, and further maintains the remote maps. (To distinguish the maps on client and server, the maps on the server are remote maps, and the maps on clients are local maps.) Communications between two sides occur when keyframes are selected by the mobile pipeline. The selected keyframes and their related map points, i.e. the newly constructed local map, are uploaded to the edge side. The server pipeline begins when it receives the local map from a client. It first saves the map on the server and then tries to merge the remote map into the global map, in which the maps are from all clients and in the same coordinate system. Next, it checks the similarity between the remote map and the global map, and eliminates redundancy to exploit memory usage. Lastly, the server optimizes the remote map by geometric constraints and sends the optimized map and segmentation result back to the client. The client corrects the camera pose and the local map, and displays the map in the current scene when receiving optimized results. Also, the segmentation result is used for semantic mask transfer among camera frames. The design of *edgeSLAM* is to run the separated pipelines in parallel. The client keeps tracking and constructing the local map while the edge server optimizes the maps the client uploaded. In this way, the client can continuously track and display the accurate camera pose and constructed map in real-time.

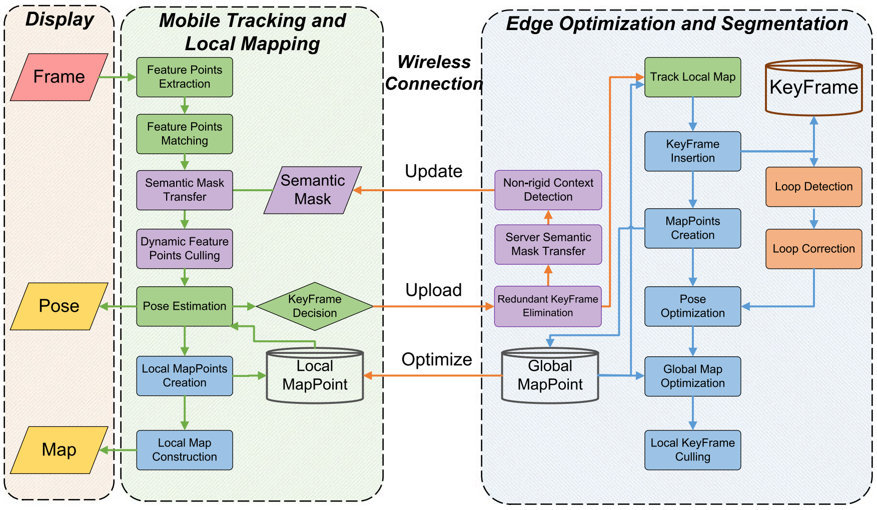


Fig. 6. *edgeSLAM* architecture.

In a nutshell, *edgeSLAM* is a real-time mobile semantic visual SLAM system. The elaborate design of *edgeSLAM* lies in fourfold:

- We decouple the whole visual SLAM into fine-grained modules and re-assign these modules between the mobile device and the edge server, guaranteeing the mobile pipeline to run in real-time.
- We design a *Parallel Local Tracking and Global Optimization* workflow. The time-consuming and complex optimization procedure is hidden by mobile local tracking and mapping pipeline, meanwhile ensuring the accuracy.
- We adopt a *Semantic Mask Transfer Strategy*. The pixel-level semantic information of a keyframe can be transferred to related non-keyframes, which dramatically reduces the inference latency and fulfill semantic segmentation tasks on mobile devices.
- We design a *Multi-user Support* mechanism that enables the semantic mask to reuse between clients and eliminates the map redundancy to exploit the memory usage, allowing the edge server to support more clients at a time.

In the following sections, we will present the details of these strategies.

4 Real-Time Mobile Semantic Visual SLAM

4.1 Decoupling and Task Re-Assignment of Visual SLAM

In ORB-SLAM, pose estimation and tracking in *Tracking* module, as well as optimization-related parts in *Local Mapping* and *Loop Closing* modules contribute most of the computation latency. On mobile devices, both pose estimation and map points creation require more than 33.3ms, which makes visual SLAM insuperable to run at $>$30fps. Moreover, they all rely on map point and keyframe databases for global optimization. These facts are exemplified in Fig. 7, which breaks down the execution latency of the ORB-SLAM task.

We consider that the decoupling and task re-assignment method should fulfill three principles: First, the mobile client should run in real-time, which means the total latency of its modules is no more than 33.3ms. Second, functional modules that frequently interact with the keyframe or map point database for further optimization should be executed on the edge server. Breaking the data dependency will also lead to high data transmission latency. Last but not least, same as the original visual SLAM, both localization (tracking camera pose) and mapping (creating map points) tasks must run on mobile devices because the user/robot needs the guide from the real-time localization result.

The task assignment strategy in *edgeSLAM* fulfills the above principles. As shown in Fig. 6, the mobile client estimates the relative rough camera pose from feature points extraction and matching modules. In the design of *edgeSLAM*, the client can localize itself in the local map and simultaneously construct the map of the surrounding environment in the short term. As for the long term, localization and accumulative error correction are done by the edge server. To achieve this, the client maintains a local map point database, which is used to estimate the camera pose and generate new map points. In order to keep the tracking latency low, *edgeSLAM* only keeps the latest 10% map points of the server's remote map points database. Meanwhile, the edge server executes the time-consuming and resource-aware optimization procedures, including optimizing pose and map, maintaining remote map point and keyframe databases, and storing the global databases from multiple clients. After receiving a keyframe, the edge side sends the optimized pose and updated map points positions back to

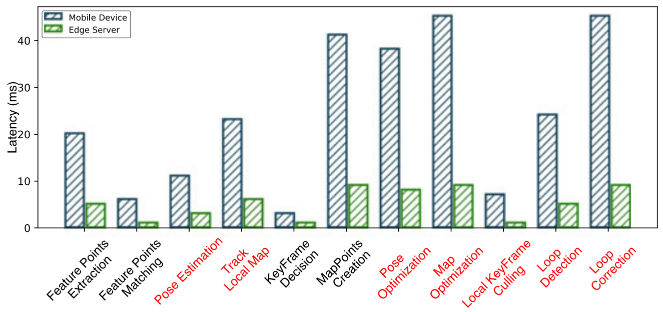


Fig. 7. Operation latency of each function unit in visual SLAM.

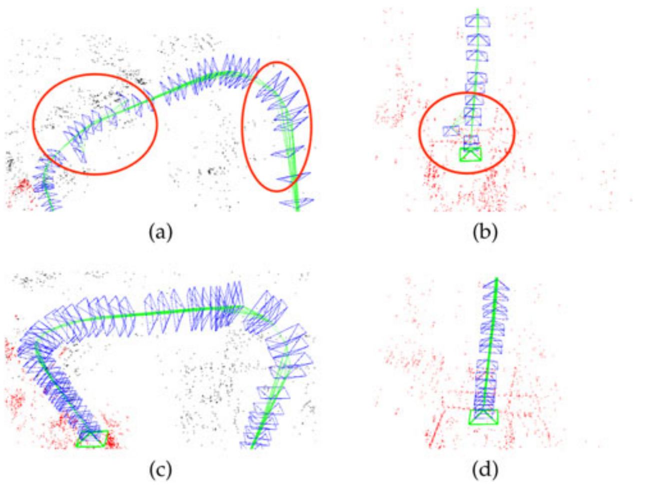


Fig. 8. Illustrations of optimization of the local constructed map. A complex local constructed map (a) before optimization on the mobile device, in which the camera poses of some frames in the red circle are deviated from the original path, and (c) after optimization, in which the camera poses are correct and smooth. Similarly, (b) and (d) show a simple trajectory on the mobile device before and after optimization, respectively.

the mobile side; and the mobile side accordingly calibrates the accumulative tracking and mapping error.

In the ORB-SLAM system, when the client fails to estimate the current camera pose, the client will attempt to recover the camera pose by using the keyframe database. This procedure is referred to as *Relocalization*. In *edgeSLAM*, we design a hierarchical relocalization mechanism that enables the client to recover the camera pose from both the local and remote keyframe database. The client first searches the local keyframe database, which only contains part of the keyframes. If the client can find a keyframe by which the pose can be estimated, it will utilize the keyframe to calculate the camera pose. Otherwise, it sends the current camera frame to the edge server. On receiving the frame, the edge server searches it in the global keyframe database containing all the keyframes. If the server finds a keyframe that can relocalize the current frame, the keyframe and its surrounding keyframes will be sent back to the client, helping the client to estimate its pose. Otherwise, the client will reset the generated map and restart the tracking and mapping processes.

4.2 Parallel Local Tracking and Global Optimization

The server needs to send the optimized remote map to the client. However, the data volume of the entire map is enormous, which dramatically increases the data serialization time and transmission latency. We further design a mechanism to reduce the data volume, as illustrated in Fig. 8. *edgeSLAM* employs an incremental updating strategy, which only sends the map points and keyframes, i.e. map elements, modified during this updating period. *edgeSLAM* tracks the changes of the map elements rather than checking the modification status of the map elements individually, thus saving the computation and processing time. Precisely, the map point and keyframe databases are monitored. Once a map element is modified (creation, deletion, and position/pose changed), it will be inserted into a set structure. This data structure guarantees the uniqueness of each element, which means the element will not be recorded twice. At the end of each updating period, the server sends the set containing the modified elements to the client. *edgeSLAM* also applies this strategy to the client so that maps between the client and server can be synchronized periodically.

By doing so, *edgeSLAM* incrementally optimizes the client map, meanwhile decreasing the transmission latency and serialization/de-serialization time.

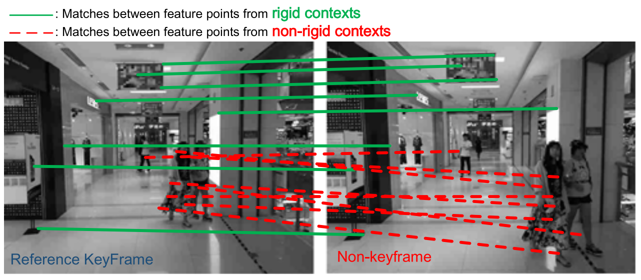

: Matches between feature points from rigid contexts
: Matches between feature points from non-rigid contexts
Reference KeyFrame
Non-keyframe

Fig. 9. Illustration of semantic mask transfer strategy.

4.3 Semantic Mask Transfer Strategy

Same as many recent works, we adapt Mask R-CNN for NRCC. Mask R-CNN is a famous instance segmentation framework. It aims to separate different instances in an image via a segmentation mask for each instance. By applying the semantic mask, the SLAM system can benefit from getting rid of the influence of the dynamic objects and thus gets a higher accuracy. However, the cost of high accuracy is time-consuming and resource-aware computations. As shown in Fig. 5a, it results in more than 150ms latency to infer a video frame, which is the bottleneck of semantic visual SLAM.

In *edgeSLAM*, we design a semantic mask transfer strategy, which can obtain the segmentation information on mobile in real-time. More specifically, as illustrated in Fig. 9, only keyframes are uploaded to the edge server. The semantic masks of keyframes are computed on the server and sent to the client. As for non-keyframe, the pixel-level semantic information can be transferred from the semantic mask of its nearest keyframe. The rationale behind the transfer strategy is: First, the feature points matched between two consecutive frames are more likely to belong to the same object category according to the matching algorithm (DBoW2) in ORB-SLAM. Second, in a typical walking scenario with a speed of 1m/s, a user (or robot) moves less than 4cm during 33.3ms (30fps). Therefore, the changes of the scene between two consecutive frames are not significant in most situations. So the semantic mask of the selected keyframe can be transferred to its near remaining keyframes. Once the mobile client receives a semantic mask, the semantic information of the related remaining keyframes will be updated. Notice that the hypothesis of the module is the accurate matching of the feature points on the dynamic objects. Thus, the semantic mask is supposed to be generated from the segmentation-based method like Mask R-CNN instead of the detection-based method like YoloV4 [51]. The *Semantic Mask Transfer Strategy* helps to alleviate the workload of the edge server and enables the mobile client a higher accuracy.

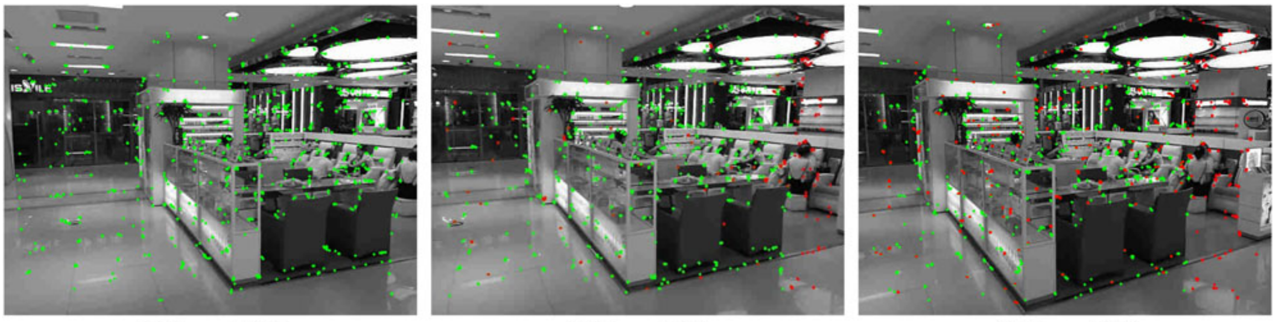


Fig. 10. Illustration of keyframe selection strategy. (a) A reference Keyframe. (b) A non-keyframe, more than 90% of the extracted feature points are matched with (a) previous Keyframe. (c) A new keyframe, whose associating rate with (a) is less than 80%.

4.4 Self-Adaptation Strategy Design

We find several parameters available for adjustment in the keyframe selection function for better adaptation to diverse environments. By adjusting these parameters dynamically, *edgeSLAM* achieves relatively optimal results under various computing power restrictions and network conditions.

Oriented FAST and rotated BRIEF (ORB) feature points [52] are commonly adopted in the ORB-SLAM system for their sufficient matching accuracy and low computation overhead. The ORB feature employs the Gaussian Pyramid to ensure the robustness of the feature matching, including scale and rotation invariance. An image pyramid is a multiscale representation of an image, where each level contains the downsampled version of the previous level. By detecting key points in each level, the ORB feature can achieve scale invariance.

Different Gaussian Pyramid layers (n_l, nlevels) lead to diverse ranges of key points search. With more layers, more computing power consumption is also needed. Thus, the basis of the keyframe selection is to adjust the related parameters when the end device has limited computing power.

Specifically, there are four principles of the keyframe selection in *edgeSLAM's* design, which must be met when generating a new keyframe. The following are the four principles:

1) More than $f_{\min}$ (mMinFrames, 20 by default) frames have passed from the last keyframe selection on the mobile device, or the number of keyframes in the map is less than $f_{\min}$ frames.
2) Less than $f_{\max}$ (mMaxFrames, 60 by default) frames have passed from the last optimization for local constructed map.
3) The ratio of extracted feature points in the current frame compared with the previous keyframe should be at least α (0.4 by default).
4) The ratio of matched feature points in the current frame compared with the previous keyframe cannot exceed β (0.9 by default).

Condition 1 ensures good optimization, and condition 3 provides good tracking. Both the conditions enhance the quality of selected keyframes. Conditions 2 and 4 reduce the redundancy of keyframes and ensure their uniqueness and representativeness.

Considering the principles above, *edgeSLAM* can appropriately increase $f_{\max}$, $f_{\min}$ and α or reduce β under poor network conditions. An illustration of the keyframe selection strategy is shown in Fig. 10.

Apart from the variables we mentioned above, the non-maximum suppression parameter n_m in the Mask R-CNN framework is also an essential parameter, which makes a trade-off between segmentation accuracy and latency. We will also adjust it to adapt to different requirements in diverse environments.

Similar to DeepDecision [39], the *Self-adaptation Strategy* takes environment measurements (network bandwidth B and network latency L) as inputs and defines the network condition N as

$$N = L + \frac{M}{B}, \tag{2}$$

where M is the data size for each frame (1280 $\times$ 720 pixels in *edgeSLAM*). Note that the arguments L and B reflect the network condition (latency, bandwidth, and congestion), making it feasible to adapt to the network fluctuation caused by the multiple agents and suitable for multi-user cases.

Furthermore, *edgeSLAM* employs the optimization strategy tuple $(f_{\min}, f_{\max}, \alpha, \beta, n_l, n_m)$. Specifically, there are 5 configuration tuples in *edgeSLAM* to adapt to various network conditions, which can be seen in Table 1.

The rationale of the self-adaptation strategy is to adjust the data transmission of the newly generated keyframes along with the NRCC traffic by controlling the keyframe generation rate. Moreover, the strategy also adjusts the computation costs by changing the ORB Pyramid levels. Take the challenging poor network conditions as an example, the client increases the keyframe generation interval so that fewer keyframe data will be transmitted to the edge server and more frames between adjacent keyframes apply the *Semantic Mask Transfer Strategy* to reuse the semantic mask of the last keyframe. Additionally, with fewer levels in the ORB Pyramid, there is a lower demand for computation (i.e., feature extraction, matching, and the pose tracking

TABLE 1
Different Configuration Tuples in Self-Adaptation Strategy

Configuration#	$f_{\min}$	$f_{\max}$	α	β	n_l	n_m	N (ms)
1	10	40	0.3	0.9	8	0.8	(0, 10]
2	20	50	0.3	0.9	6	0.8	(10, 15]
3	20	60	0.4	0.9	6	0.8	(15, 35]
4	30	70	0.5	0.8	4	0.7	(35, 50]
5	40	80	0.5	0.7	4	0.6	(50, 300)

based on them) on mobile devices. Eventually, the client could still achieve real-time performance at the expense of slightly degraded system accuracies (demonstrated in Section 6.5).

5 Multi-User Support

Multi-user SLAM has been widely used in many scenarios, such as fire rescue, logistics sorting, and indoor navigation. In such cases, multiple clients need assistance from the edge server, which accounts for a large partition of computing resources. However, the resources on the edge server are limited, and the server may easily be overloaded when serving multiple clients.

At a high level, there are three challenges in the multi-user support problem:

- The maps from different clients are of various coordinate systems, which means the map data are isolated spatially. On the one hand, there are overlaps and redundancy in the global map. On the other hand, the extra information brought by various clients cannot be used for cross-validating the accuracy of the maps and not to mention further optimizing them. Thus, this isolation harms both the mapping accuracy and memory utilization rate.
- As the number of clients grows, the edge server needs to handle more semantic mask calculation. The mask calculation latency increases accordingly, which prevents the real-time running of the mobile clients and harms the localization accuracy.
- The total map size scales rapidly with the growing number of clients and the system's running time, exceeding the memory limit. We also find that the trajectories of the clients running at close range may easily overlap, which along with the revisit of the same place, produce severe data redundancy, prohibiting the scalability of the system. The memory usage becomes the bottleneck of the scalability issue.

To solve the three challenges, we find that keyframes are similar when the clients share the same mapping area and can be utilized with a global map. Thus, we design a mechanism that optimizes the redundant computation and memory storage to enable the edge server to support more clients.

5.1 Map Merging

Most of the trajectories are overlapped and can be merged into a global map. The map merging is done by the co-visibility of the keyframes. If there are at least 8 pairs of the matched feature points, the relative transform of the two keyframes can be computed, and with the original poses of the keyframes, the transform between their coordinate systems can be derived. Then, all the keyframes and map points in this map can be expressed in another coordinate system. In *edgeSLAM's* design, the map merging task is treated as a loop closure event, which is implemented by the *Loop Closing* module. If there are two maps merged, *edgeSLAM* marks the coordinate system of the first map as the reference coordinate system. *edgeSLAM* also stores the transforms between all maps and the reference map. By doing this, all maps that overlap with other map(s) share the same coordinate system, and thus all keyframes and map points are utilized in the global coordinate system.

5.2 Server Semantic Mask Transfer

As described in Section 4.3, NRCC is a time-consuming and resource-aware process. Parallel requests from multiple clients exacerbate the resource usage, causing a longer average response time and decreasing the total number of clients the server can support. Thus, *edgeSLAM* applies the aforementioned *Semantic Mask Transfer Strategy* on the server. Specifically, it compares the newly received keyframe X with those in the server's global keyframe database by the fast matching algorithm(DBoW2). If there is a keyframe Y more than N%(70%, in our case) similar with the keyframe X, the semantic mask of the keyframe X is transferred from the mask of keyframe Y. Otherwise, the semantic mask of the keyframe X is computed as usual. Reusing the previous results reduces the computation time and computing resources, yet the server with the same configuration can support more clients.

5.3 Redundant Keyframe Elimination

Multiple clients may often visit the same place, which travels and maps the same area several times, causing redundancy and increasing memory usage. Thus, we design a mechanism to eliminate redundancy. *edgeSLAM* maintains a K-D tree of the keyframe positions. The K-D tree accelerates the searching of the nearby keyframes by its pose. Once the server receives a keyframe X, it first searches its nearest keyframe Y. Then a similarity check based on feature points is performed between keyframe X and Y, in case the two keyframes are only in the same position but with different poses. If the match ratio is above 70%, *edgeSLAM* project the keyframe X's feature points to the keyframe Y's coordinate and merge the transformed feature points with keyframe Y's. Lastly, keyframe X removes its original data and only keeps a reference to the keyframe Y, becoming an alias of the keyframe Y. In this way, all accesses on keyframe X are forwarded to keyframe Y, and all modifications to keyframe Y are also reflected on keyframe X. By doing this, *edgeSLAM* decreases the redundancy of the global map and thus enhances the workload of the edge server.

6 Experiments and Evaluation

6.1 Implementation

The implementation of *edgeSLAM* follows the system workflow in Fig. 6, and is developed on *ORB-SLAM* [12], the state-of-the-art visual SLAM system. Specifically, we adopt a Server-Client structure by splitting and reusing the *ORB-SLAM* modules.

Client. We implement the client part of *edgeSLAM* on mobile devices (Nvidia Jetson TX2, Apple iPad Mini 5, iPhone X, and Galaxy S10) mostly reusing the *Tracking* and *Local Mapping* modules of *ORB-SLAM*. To begin with, we continuously capture video frames by a camera on devices using OpenCV [53] API and JetPack Camera API and feed it to *Mobile Tracking and Local Mapping* modules.

`ORBextractor()` function extracts ORB feature points. Meanwhile, CUDA (only on the development board) accelerates the feature extracting process [54]. This process is followed by `featurePointsMatching()` function to calculate associations between extracted feature points. Furthermore, `featureCulling()` takes over the procedure, which leverages semantic mask to ensure the rigidity of the environment. Keyframe decision function `isKeyFrame()` will be finally performed according to principles described in Section 4.4. It determines whether the newly created frame is a keyframe. If one keyframe is selected, it is uploaded to the edge server. Meanwhile, `poseEstimation()` and `localMapGeneration()` estimates the coarse-grained camera pose and construct a map. Once client receives a map slice sent from edge server, `Update()` function optimizes pose and map. As for the *Self-Adaption Strategy*, we use *ping*, a network performance evaluation tool using ICMP protocol, to get the network latency and *iperf*, the bandwidth measurement tool, to get the data link bandwidth.

Server. We implement the server part of *edgeSLAM* on the edge server, and reuse the visualization with some user interface modifications via OpenCV, and most of *ORB-SLAM* modules including *Tracking*, *Local Mapping*, *Loop Closing* and so on. The server is equipped with Intel(R) Xeon(R) CPU E5-2620v4 of 2.10GHz main frequency and 256G RAM, running the Ubuntu 16.04 operating system. For Mask R-CNN, we use TITAN V GPU with CUDA version 9.1.85. We apply our Mask R-CNN models with the ResNet-FPN-50 backbone, and the network parameters are pretrained on COCO image Dateset [55]. The Mask R-CNN code is implemented in python-3.6.5 with pytorch-0.4.0 [56].

Remote Data Interaction. The Remote Procedure Call (RPC) refers to inter-process communication (IPC). However, different processes have distinct address spaces. The RPC model helps to deal with the communication between the client and server. The client calls a function with its original argument(s), and the function is invoked automatically on the server. It hides application protocol details, argument parsing, message passing, and network communication process, acting as an intermediary conveying function calling information. The RPC model bonds the client and the server together, which is perfectly suitable for the divided structure of *edgeSLAM*. By applying a widely-used RPC framework gRPC [57] developed by Google, *edgeSLAM* achieves efficient data transmission and remote function invocation.

edgeSLAM leverages the boost serialization library [58] which is used to reconstitute an equivalent structure in another program context. Specifically, the library serializes C\raise.22ex+\raise.22ex+ objects into binary form, which is convenient to transfer over the network, and also de-serializes the binary data into C\raise.22ex+\raise.22ex + objects. Moreover, the strategy helps to encode the structure to binary stream and embed it in Protobuf [59] package as the RPC arguments, thus, saving network bandwidth and accelerating the data transmission rate.

6.2 Experiment Setup

We have performed an extensive experimental validation of our system in two standard SLAM datasets: TUM [23] and KITTI [24]. The TUM benchmark is an excellent dataset to evaluate the accuracy of camera localization as it provides several sequences with accurate ground truth obtained with an external motion capture system. The odometry benchmark from the KITTI dataset contains 11 sequences from a car driving around a residential area with accurate ground truth from GPS and a Velodyne laser scanner. The dataset is exceedingly challenging for monocular vision due to fast rotations and areas with a lot of foliage, making data association more difficult.

We evaluate the general localization accuracy of *edgeSLAM*, in 16 hand-held indoor sequences of the TUM RGB-Monocular benchmark. Moreover, in 5 sequences from the KITTI dataset, we evaluate the tracking accuracy of camera pose and efficiency of the constructed map optimization. We have carried out all experiments with an Nvidia Jetson TX2 development as a mobile device and a desktop computer as an edge server. The configurations are shown above. In our experiments, we feed the image from datasets at 30 fps into the mobile device. *edgeSLAM* runs in real-time and processes the images precisely at the frame rate they acquired.

Furthermore, to extensively evaluate the performance of the edge-assisted design of *edgeSLAM*, we compare *edgeSLAM* with *ORB-SLAM* and *Mask-SLAM*. The former is the original baseline of our *edgeSLAM* without edge-assisted method and NRCC. The latter is the system used in recent works [22], [60]. They execute NRCC for each video clip and fuse the semantic information with ORB-SLAM. However, neither of these comparative methods can work on mobile devices in real-time. Thus, the entire processing tasks of the comparative systems are fully offloaded to the edge server once the mobile device captures a frame, i.e., the client acts as a data collector sending the frames to the server, and the edge server feeds the frames to the whole SLAM system. After the system outputs the device's location, the server sends the result back to the client and the client display the received location. Besides, we also compare *edgeSLAM* with *Edge-SLAM* [45], the latest edge-assisted SLAM system. Different from *edgeSLAM*, Edge-SLAM moves *Local Mapping* module to the edge server and only keeps the *Tracking* module and fixed size of the map on the mobile device.

6.3 Performance Comparison

6.3.1 Localization Accuracy in TUM Dataset

We first examine the localization accuracy of *edgeSLAM*. Fig. 11 depicts the performance of the proposed *edgeSLAM* as well as three other comparative systems in indoor localization scenarios. As shown, the 95th percentile localization accuracy of *edgeSLAM*, Edge-SLAM, ORB-SLAM, and Mask-SLAM is 1.97cm, 2.92cm, 3.66cm, and 6.22cm, respectively. *edgeSLAM* outperforms the other three approaches by more than 30%. The delightful result comes from the low end-to-end latency, which is ensured in *edgeSLAM* by the edge-assisted method and task assignment strategy. Compared with Edge-SLAM, our system achieves higher localization accuracy benefiting from the NRCC module, which gets the system rid of the influence of the dynamic objects. Further, we keep *Local Mapping* on the client so that the map is generated and maintained on the mobile device, which

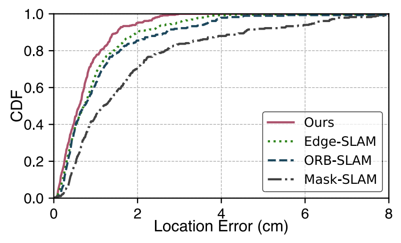


Fig. 11. Localization accuracy on TUM dataset.

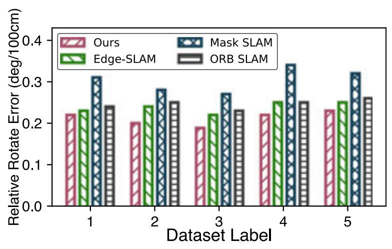


Fig. 12. Tracking accuracy on KITTI dataset.

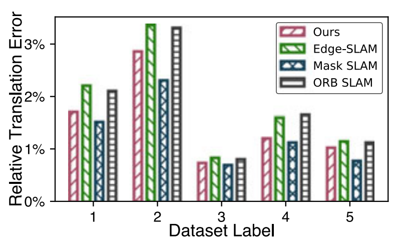


Fig. 13. Mapping accuracy on KITTI dataset.

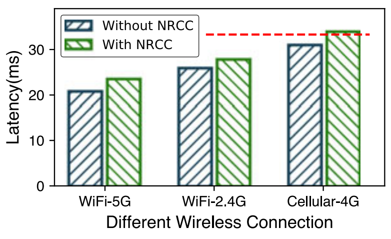


Fig. 14. End-to-end latency comparison.

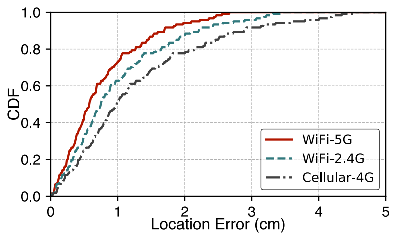


Fig. 15. Localization accuracy comparison.

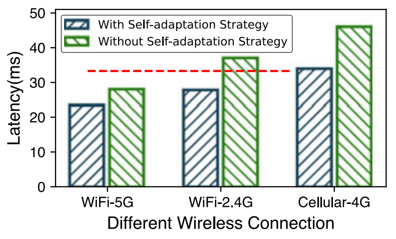


Fig. 16. Impact of self-adaptation on latency.

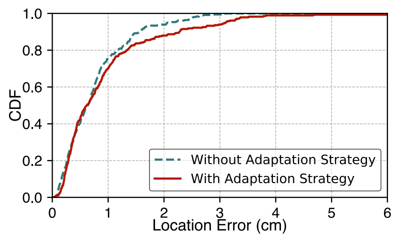


Fig. 17. Impact of self-adaptation on accuracy.

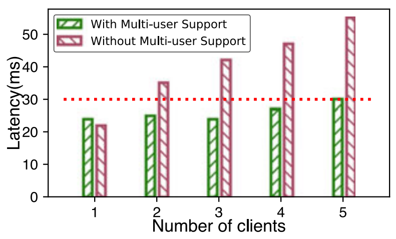


Fig. 18. Impact of multi-user support on latency.

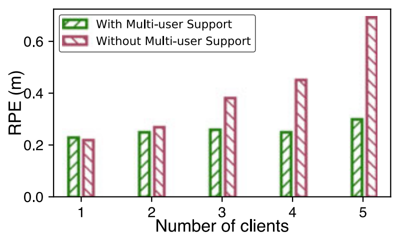


Fig. 19. Impact of multi-user support on RPE.

enables stable tracking and fast relocalization once the mobile device is lost.

6.3.2 *Pose Tracking Accuracy in KITTI Dataset*

We further examine the camera pose tracking accuracy of *edgeSLAM* in KITTI camera rotation dataset. We calculate the Relative Rotation Error (RRE, an essential evaluation indicator in KITTI dataset, means the average relative rotational error among the whole trace) of the tracking result compared with the ground truth. The performance of *edgeSLAM* as well as three comparative methods are depicted in Fig. 12. The tracking accumulative bias of *edgeSLAM* is within 0.22 degrees for one-meter length traces, which outperforms Edge-SLAM, ORB-SLAM and Mask-SLAM by 4.0%, 5.1% and 17.6%. The KITTI dataset is a challenging benchmark for monocular vision, as it contains a lot of dynamic objects like moving cars. Thus, systems like ORB-SLAM and Edge-SLAM without *NRCC* can be easily lost in the dynamic environment. Although Mask-SLAM utilizes NRCC, the high end-to-end latency causes the severe localization errors which can be avoided by the low-latency design of *edgeSLAM*.

6.3.3 *Mapping Precision in KITTI Dataset*

Finally, we evaluate the mapping precision of *edgeSLAM* in KITTI Dataset. We calculate the Relative Translation Error (RTE, another fundamental evaluation indicator in KITTI dataset, means the average relative trajectory error among the whole trace) of the constructed map compared with ground truth. The experiment compared the optimized global map stored in the edge server in *edgeSLAM* with maps constructed by the other three systems. As shown in Fig. 13, in all five image sequences, *edgeSLAM* outperforms Edge-SLAM by more than 13% and ORB-SLAM by more than 12%. As for Mask-SLAM, the offloading-all strategy achieves the highest mapping precision, which is 0.5cm for one-meter length traces with the powerful edge server. *edgeSLAM* works on the computation-constrained mobile device and enables real-time SLAM, rendering the 10% performance gap with Mask-SLAM, demonstrating that *edgeSLAM* achieves competitive performance.

In summary, real-time *edgeSLAM* achieves enhanced localization accuracy and competitive mapping performance with state-of-the-art online and offline frameworks. The grateful performance comes from the efficient design of the edge-assisted method and the adaption of the semantic mask transfer strategy.

6.4 End-to-End Latency

Our system achieves an end-to-end latency within 33.3ms inter-frame time at 30fps to ensure a smooth localization and mapping experience. To validate this, we run *edgeSLAM* under different network connections and calculate the average end-to-end latency for each frame in Fig. 14. The red dashed line in the figure is the 33.3ms deadline

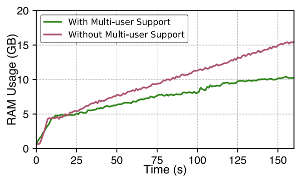


Fig. 20. Impact of multi-user support on memory usage.

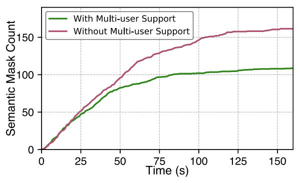


Fig. 21. Impact of multi-user support on semantic mask calculation count.

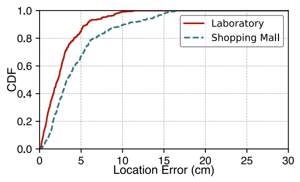


Fig. 22. Pedestrian tracking accuracy.

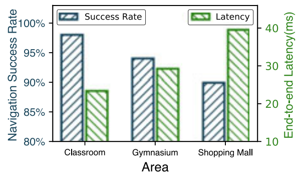


Fig. 23. Robot navigation success rate.

for 30fps SLAM devices. We can find that our system can finish the entire task within 33.3ms under the majority of wireless connection conditions (34.1ms under 4G wireless connection with NRCC). Moreover, the average frame latency increase merely $<$ 4ms if NRCC is involved in *edgeSLAM*, which is the bottleneck in most systems as aforementioned.

Furthermore, we evaluate the robustness of *edgeSLAM* under different network conditions. As shown in Fig. 15, *edgeSLAM* can reach an average accuracy of 0.7cm, 0.9cm, and 1.3cm under different wireless connections, respectively. The drift of accuracy influenced by network conditions is within 0.5cm.

6.5 Impact of Self Adaptation Strategy

To demonstrate the effectiveness of the designed self-adaptation strategy. We evaluate the average frame latency and system localization accuracy with and without leveraging the strategy. If the self-adaptation strategy is disabled, *edgeSLAM* uploads each frame once the mobile client receives the optimization result of the previous one. As shown in Fig. 16, after bringing in the adaptation strategy, the average frame latency decreases by more than 5ms, especially $>$10ms under Celluar-4G wireless connection. The result reflects that the designed self-adaptation strategy is vital in ensuring *edgeSLAM* runs in real-time, especially under unfavorable network conditions. Fig. 17 shows the performance of *edgeSLAM* with and without adaptation strategy. As seen, the average localization accuracies are 1.03cm and 0.93m, respectively. The precision difference is within 10%. As mentioned in Section 4.4, the self-adaptation strategy achieves real-time performance with an acceptable precision loss of 10%, the above results show that the leverage of the self-adaptation strategy effectively decreases the end-to-end latency while yielding similar accuracies.

6.6 Impact of Multi-User Support

To prove the effectiveness of the *Multi-user Support* module, we perform extensive experiments on our system with and without applying the module. Without the *Multi-user Support* module, the edge server computes every semantic mask of the received keyframes and completely stores the Keyframes. The experiments are conducted with the EuRoC MAV Machine Hall dataset [61], which contains 5 sequences in the same area. Therefore, the maps can be fully merged. In the multi-user case, the clients run different dataset sequences simultaneously, respectively. In the single client case, the client runs the dataset sequences sequentially, and the latency and accuracies are averaged.

As shown in Fig. 18, after enabling this module, *edgeSLAM* achieves real-time SLAM in all cases, and the average latency decreases by 10 more milliseconds, especially when the client number reaches 5. It is worth noting that there are some more operations in the multi-user module in the one-client case, such as checking similarity and server semantic mask transfer, which may bring about more processing time. So the average latency is slightly larger when this module is enabled. As suggested in Fig. 19, the RPE(Relative Pose Error) of *edgeSLAM* with multi-user support is lower in most cases except the one-client case. As aforementioned, there are more procedures in the multi-user module so that the error is a bit larger.

To better illustrate the memory consumption trends, the dataset sequence input of the experiment with 5 clients is sequential. The memory consumption is shown in Fig. 20. It shows that when supporting 5 clients, *edgeSLAM* with *Multi-user Support* module consumes 30% less memory. Moreover, from 100s on, the memory consumption is becoming stable, which means the area is fully explored and mapped.

We also evaluate the impact of the *Server Semantic Transfer Strategy*. Semantic mask count refers to the number of semantic masks that are calculated by the edge server. With the help of the strategy, semantic mask can be transferred on the server so that the computation resources are saved. So the less the count is, the more effective the strategy is. Fig. 21 indicates that the *edgeSLAM* with *Multi-user Support* costs 30% less semantic mask calculation, which means the strategy significantly decreases the computation resources in the multi-user scenario.

In short, the *Multi-user Support* module is a crucial component in the *edgeSLAM*. It leverages the maps' spatial relationship, reuses the intermediate semantic masks, and eliminates the system's redundancy, thus extending its workload.

6.7 Case Study

6.7.1 Pedestrian Localization and Tracking

We conducted experiments in a laboratory and 1st floor of a shopping mall. These two areas have different floor layouts, diverse wireless environments, and distinct user behavior patterns. In particular, the crowded shopping mall is the most dynamic. While there are a reasonable number of users in the laboratory most of the time.

Setup. In this case, the client side of *edgeSLAM* is implemented on an iOS platform (Apple iPhone X). The server we use is a Lenovo IdeaPad-Y700 with i7-6700HQ CPU of 2.6GHz main frequency and 16G RAM, running the Ubuntu 16.04 operating system. For Mask R-CNN, the GPU we used is TITAN V with CUDA version 9.1.85.

Ground Truth Acquisition. To obtain the ground truth, which is the accurate location of pedestrians, we recruited a volunteer to watch surveillance videos, artificially differentiate and track pedestrians, and manually mark their locations on the 2D indoor map.

Localization Result Analysis. Fig. 22 shows the performance of *edgeSLAM* for pedestrian localization in different areas. As seen, *edgeSLAM* yields an average accuracy of 7.6cm in the laboratory and 9.9cm in the shopping mall. The corresponding 95th percentile location errors in these two buildings are 9.9cm, 11.4cm, respectively. The result shows that *edgeSLAM* can locate a pedestrian at fine-grained in real-time (40fps in this case study), outperforming state-of-the-art RF-based and vision-based localization systems. Moreover, *edgeSLAM* yields similar performance (accuracy difference $< 20\%$) regardless of the environmental difference because of the semantic mask transfer strategy.

6.7.2 Robot Mapping and Navigation

We conducted extensive experiments in an office building, a gymnasium and the 1st-3rd floors of a shopping mall, whose sizes are about $400m^2$, $1{,}000m^2$ and $4{,}000m^2$, respectively. Overall, we design 17 navigation paths, including 4 short paths ($\leq 100m$), 6 medium paths ($100m - 200m$) and 7 long paths ($\geq 200m$), covering all the main pathways of the testing areas.

Setup. In this test scenario, the client side of *edgeSLAM* is a robot implemented with an Nvidia Jetson TX2. The edge server is mentioned above.

Evaluation Metrics. Similar to some existing works like *Travi-Navi* and *Pair-Navi*, we set checkpoints at turns, escalators, and some landmarks on each trajectory. In total, we set 274 checkpoints for the 21 navigation paths. *Navigation success rate* is defined as the rate of successful arrival at each checkpoint and departure from the checkpoint within a radius of $2m$ is seemed as immediate tracking failure.

Navigation Performance. The performance of *edgeSLAM* is depicted in Fig. 23. The average navigation success rates (with NRCC) by *edgeSLAM* in the classroom building, gymnasium, and shopping mall are 97%, 94% and 90%, respectively.

Additionally, the wireless connections are different in three areas. In the classroom building, the client is connected to an edge server under Wi-Fi 5G link, while in the gymnasium and shopping mall are Wi-Fi 2.4G and Cellular 4G, respectively. The average end-to-end latency for each frame is also shown in Fig. 23. The result demonstrates that in the classroom and gymnasium, *edgeSLAM* can run in real-time ($>$ 30fps) meanwhile about 25fps in the shopping mall. The rationale behind that is the network suffers severe fluctuation in crowded shopping malls.

7 Conclusion

In this work, we propose *edgeSLAM*, a semantic visual SLAM system for mobile devices, achieving both accuracy and real-time simultaneously in multi-user scenarios with the help of edge computation resources. The core technology of our design lies in: 1, the decomposition of computation modules of SLAM and semantic segmentation; 2, avoidance of redundant computation by reuse the intermediate results on the server; 3, the adaption of system parameters to various conditions of network bandwidth and latency. 4, full use of the spatial relationship between clients to eliminate the map redundancy and save the computational resources, expanding the system capacity. We fully implement *edgeSLAM* and extensively evaluate the performance on three datasets under different network conditions. The results show that *edgeSLAM* achieves satisfying results in all scenarios. Being truly real-time, *edgeSLAM* sheds light on practical localization and mapping for mobile users and robots.

Acknowledgments

Hao Cao and Jingao Xu are co-primary authors (alphabetic order).

References

[1] C. Wu, J. Xu, Z. Yang, N. D. Lane, and Z. Yin, "Gain without pain: Accurate WiFi-based localization with fingerprint spatial gradient," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, 2017, pp. 1–19.
[2] J. Wang and D. Katabi, "Dude, where's my card?," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 51–62.
[3] Z. Yang, C. Wu, and Y. Liu, "Locating in fingerprint space," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 269–280.
[4] C. Wu, Z. Yang, and Y. Liu, "Smartphones based crowdsourcing for indoor localization," *IEEE Trans. Mobile Comput.*, vol. 14, no. 2, pp. 444–457, Feb. 2015.
[5] J. Xu et al., "iVR: Integrated vision and radio localization with zero human effort," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, 2019, pp. 1–22.
[6] J. Xu et al., "Embracing spatial awareness for reliable WiFi-based indoor location systems," in *Proc. IEEE 15th Int. Conf. Mobile Ad Hoc Sensor Syst.*, 2018, pp. 281–289.
[7] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu, "Tagoram," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 237–248.
[8] L. Shangguan, Z. Yang, A. X. Liu, Z. Zhou, and Y. Liu, "STPP: Spatial-temporal phase profiling-based method for relative RFID tag localization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 596–609, Feb. 2017.
[9] C. Wu, Z. Yang, and C. Xiao, "Automatic radio map adaptation for indoor localization using smartphones," *IEEE Trans. Mobile Comput.*, vol. 17, no. 3, pp. 517–528, Mar. 2018.
[10] L. Li, P. Xie, and J. Wang, "RainbowLight," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 807–809.
[11] H. Abdelnasser et al., "SemanticSLAM: Using environment landmarks for unsupervised indoor localization," *IEEE Trans. Mobile Comput.*, vol. 15, no. 7, pp. 1770–1782, Jul. 2016.
[12] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
[13] C. Cadena et al., "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.
[14] M. Panzarino, "Apple is rebuilding Maps from the ground up," *Tech Crunch*, 2018. [Online]. Available: https://www.thurrott.com/apple/162150/apple-rebuilding-maps-ground
[15] Amazon Warehouse with Robots, 2019. [Online]. Available: https://www.wired.com/story/amazon-warehouse-robots/
[16] "DJI Industrial Solutions," 2020. [Online]. Available: https://www.dji.com/cn/products/industrial
[17] X. Zhang, Z. Yang, Y. Liu, and S. Tang, "On reliable task assignment for spatial crowdsourcing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 174–186, Jan.–Mar. 2019.
[18] L. Cheng and J. Wang, "ViTrack: Efficient tracking on the edge for commodity video surveillance systems," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1052–1060.
[19] J. G. Andrews et al., "What will 5G be?," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.
[20] I. C. S. L. S. Committee et al., "IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11^*, 2007.

[21] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.

[22] E. Dong, J. Xu, C. Wu, Y. Liu, and Z. Yang, "Pair-navi: Peer-to-peer indoor navigation with mobile visual SLAM," in *Proc. Conf. ComputerComput. Commun.*, 2019, pp. 1189–1197.

[23] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 573–580.

[24] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.

[25] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[26] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge, U.K.: Cambridge Univ. Press, Mar. 2004.

[27] C. Yu et al., "DS-SLAM: A semantic visual SLAM towards dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1168–1174.

[28] V. Badrinarayanan, A. Handa, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," 2015, *arXiv:1505.07293*.

[29] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[30] E. Sucar, K. Wada, and A. Davison, "NodeSLAM: Neural object descriptors for multi-view shape reconstruction," in *Proc. Int. Conf. 3D Vis.*, 2020, pp. 949–958.

[31] A. Rosinol et al., "Kimera: From SLAM to spatial perception with 3D dynamic scene graphs," *Int. J. Robot. Res.*, vol. 40, no. 12/14, pp. 1510–1546, 2021.

[32] Y. Chang, Y. Tian, J. P. How, and L. Carlone, "Kimera-multi: A system for distributed multi-robot metric-semantic simultaneous localization and mapping," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 11210–11218.

[33] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic SLAM," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1722–1729.

[34] J. Civera, D. Gálvez-López, L. Riazuelo, J. D. Tardós, and J. M. M. Montiel, "Towards semantic SLAM using a monocular camera," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 1277–1284.

[35] Y. Liu and J. Miura, "RDMO-SLAM: Real-time visual slam for dynamic environments using semantic label prediction with optical flow," *IEEE Access*, vol. 9, pp. 106 981–106 997, 2021.

[36] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 297–312.

[37] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.

[38] Z. Chen et al., "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–14.

[39] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE Conf. Comput. Commun.*, 2018.

[40] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–19.

[41] V. K. Sarker, J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Offloading SLAM for indoor mobile robots with edge-fog-cloud computing," in *Proc. 1st Int. Conf. Adv. Sci., Eng. Robot. Technol.*, 2019, pp. 1–6.

[42] J.-Q. Li et al., "A novel edge-enabled SLAM solution using projected depth image information," *Neural Comput. Appl.*, vol. 32, no. 19, pp. 15 369–15 381, 2020.

[43] P. Huang, L. Zeng, K. Luo, J. Guo, Z. Zhou, and X. Chen, "ColaSLAM: Real-time multi-robot collaborative laser SLAM via edge computing," in *Proc. IEEE/CIC Int. Conf. Commun. China*, 2021, pp. 242–247.

[44] X. Cui, C. Lu, and J. Wang, "3D semantic map construction using improved ORB-SLAM2 for mobile robot in edge computing environment," *IEEE Access*, vol. 8, pp. 67179–67191, 2020.

[45] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-SLAM: Edge-assisted visual simultaneous localization and mapping," in *Proc. 18th Int. Conf. Mobile Syst., Appl., Serv.*, 2020, pp. 325–337.

[46] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular SLAM with multiple micro aerial vehicles," in *Proc. IEEE/RSJ Int. Conf. Robots*, 2013, pp. 3962–3970.

[47] L. Riazuelo, J. Civera, and J. Montiel, "C2TAM: A Cloud framework for cooperative tracking and mapping," *Robot. Auton. Syst.*, vol. 62, no. 4, pp. 401–413, Apr. 2014.

[48] P. Schmuck and M. Chli, "CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *J. Field Robot.*, vol. 36, no. 4, pp. 763–781, 2019.

[49] J. Xu et al., "{SwarmMap}: Scaling up real-time collaborative visual {SLAM} at the edge," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 977–993.

[50] "Object keypoint similarity (OKS)," 2017. [Online]. Available: http://cocodataset.org/#keypoints-eval

[51] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[52] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, 2011, pp. 2564–2571.

[53] G. Bradski, "The OpenCV Library," *Dr. Dobb's J. Softw. Tools*, vol. 120, pp. 122–125, 2000.

[54] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Reading, MA, USA: Addison-Wesley Professional, 2010.

[55] T.-Y. Lin et al., "Microsoft coco: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[56] "Pytorch mask r-CNN," 2017. [Online]. Available: https://github.com/multimodallearning/pytorch-mask-rcnn

[57] "Grpc," 2017. [Online]. Available: https://grpc.io/

[58] "Boost serialization," 2015. [Online]. Available: https://www.boost.org/doc/libs/1_70_0/libs/serialization/doc/index.html

[59] "Protocol buffers," 2015. [Online]. Available: https://developers.google.com/protocol-buffers/

[60] I. A. Barsan, P. Liu, M. Pollefeys, and A. Geiger, "Robust dense mapping for large-scale dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 7510–7517.

[61] M. Burri et al., "The EuRoC micro aerial vehicle datasets," *Int. J. Robot. Res.*, vol. 35, no. 10, pp. 1157–1163, 2016.

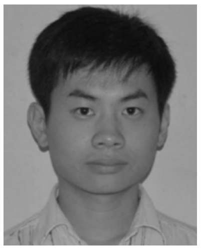
Hao Cao received the BE degree from the College of Intelligence and Computing, Tianjin University, in 2019. He is currently working toward the PhD degree with the School of Software, Tsinghua University. His research interests include Internet of Things and mobile computing.

Jingao Xu received the BE and PhD degrees from the School of Software, Tsinghua University, in 2017 and 2022, respectively. He is now a postdoc research fellow with the School of Software, Tsinghua University. His research interests include Internet of Things and mobile computing.

Danyang Li (Student Member, IEEE) received the BE degree from the School of Software, Yanshan University, in 2019, and the MS degree from the School of Software, Tsinghua University, in 2022. He is currently working toward the PhD degree from the School of Software, Tsinghua University. His research interests include Internet of Things and mobile computing.

Longfei Shangguan (Member, IEEE) received the PhD degree from the Hong Kong University of Science and Technology. He is currently an assistant professor with the University of Pittsburgh. His research interests include all aspects of IoT systems: from building novel IoT applications, solving security issues all the way down to optimizing the network stack, and designing low-power IoT hardware.

Yunhao Liu (Fellow, IEEE) received the BS degree from Automation Department, Tsinghua University, the MA degree from Beijing Foreign Studies University, China, and the MS and a PhD degrees in computer science and engineering from Michigan State University, USA. He is now the dean of the Global Innovation Exchange Institute, Tsinghua University.



Zheng Yang (Fellow, IEEE) received the BE degree in computer science from Tsinghua University, Beijing, China, in 2006, and the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong, in 2010. He is currently an associate professor with Tsinghua University. His main research interests include Internet of Things and mobile computing. He is the PI of National Natural Science Fund for Excellent Young Scientist. He was the recipient of the State Natural Science Award (second class).