Reshaping Edge-Assisted Visual SLAM by Embracing On-Chip Intelligence

Danyang Li, Yishujie Zhao, *Student Member, IEEE*, Jingao Xu, Shengkai Zhang, Longfei Shangguan, Qiang Ma, Xuan Ding, *Member, IEEE*, and Zheng Yang, *Fellow, IEEE*

***Abstract*—Edge-assisted visual SLAM plays a crucial role in enabling innovative mobile applications, such as autonomous swarm inspection, search-and-rescue, and smart logistics. Constrained by the computational capacities of lightweight mobile devices, current approaches delegate lightweight, time-sensitive tracking tasks to the mobile end while offloading resource-intensive, latency-tolerant map optimization tasks to the edge. However, our pilot study reveals several limitations of the tracking-optimization decoupled paradigm, stemming from the disruption of inter-dependencies between the two tasks. In this paper, we design and implement edgeSLAM2, an innovative system that reshapes the edge-assisted visual SLAM paradigm by tightly integrating tracking and partial-yet-crucial optimization on mobile. edgeSLAM2 harnesses the heterogeneous computing units offered by the commercial systems-on-chip (SoCs) to enhance the computational capacity of mobile devices, which in turn, allows edgeSLAM2 to design a suit of novel algorithms for map sync, optimization, and tracking that accommodate such architectural upgrade. By capitalizing on the full potential of on-chip intelligence, edgeSLAM2 supports both solitary and collaborative SLAM with accuracy and immediacy, underpinned by a cohesive software-hardware co-design. We deploy edgeSLAM2 on drones for industrial inspection. Comprehensive experiments in one of the world's largest oil fields over three months demonstrate its superior performance.**

***Index Terms*—Edge computing, visual SLAM, multi-agent collaboration, software and hardware co-design, drone-based applications.**

I. Introduction

VISUAL Simultaneous Localization and Mapping (SLAM) employs video streams to simultaneously construct a 3D environmental map and estimate the camera's pose [1], [2], [3], [4]. Its real-time functionality, particularly on mobile devices such as drones and robots, is pivotal for underpinning an array of intelligent device applications such as environmental perception, self-state estimation [5], [6], [7], [8], and capabilities like drone flight control, obstacle avoidance, and intelligent interaction [9], [10], [11], [12].

Manuscript received 8 March 2024; revised 21 May 2024; accepted 23 June 2024. Date of publication 8 July 2024; date of current version 5 November 2024. This work was supported in part by National Key Research Plan under Grant 2021YFB2900100, in part by NSFC under Grant 62372265, Grant 62202262, Grant 62302254, Grant 62272462, and Grant 62072272. Recommended for acceptance by W. Wang. *(Danyang Li and Yishujie Zhao are co-primary authors.) (Corresponding author: Jingao Xu.)*

Danyang Li, Yishujie Zhao, Jingao Xu, Qiang Ma, Xuan Ding, and Zheng Yang are with the School of Software and BNRist, Tsinghua University, Beijing 100084, China (e-mail: lidanyang1919@gmail.com; chillpill995@gmail.com; xujingao13@gmail.com; thumaq@mail.tsinghua.edu.cn; dingx04@gmail.com; hmilyyz@gmail.com).

Shengkai Zhang is with the School of Information Engineering, Wuhan University of Technology, Wuhan 430062, China (e-mail: shengkai@whut.edu.cn).

Longfei Shangguan is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260 USA (e-mail: longfei@pitt.edu).

Digital Object Identifier 10.1109/TMC.2024.3424452

Visual SLAM's computational intensity impedes efficient and accurate operations on lightweight devices such as Micro Aerial Vehicles (MAV) and smartphones [5], [6]. To enhance system accuracy and efficiency on mobile, current practice resorts to *edge computing* and design a *front-end Tracking with back-end Optimization* edge-assisted architecture. Within this setup, mobile devices focus on lightweight, time-sensitive *tracking* tasks for pose estimation and map generation. Meanwhile, the resource-intensive tasks of local and global map *optimization* are offloaded to edge servers.

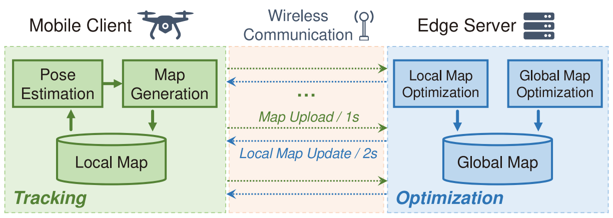
Such a *tracking-optimization* decoupled strategy not only alleviates resource pressures on mobile devices [13], [14], [15] but also allows edge servers to centralize and optimize visual maps from multiple agents, enhancing collaborative efforts for tasks like cooperative scheduling [16], [17], [18]. This edge-assisted paradigm underpins numerous applications, e.g., smart logistics [19], warehouse sorting [20], and industrial inspection [21].

While existing edge-assisted SLAM systems show promise, our deployment of these systems on drones for industrial inspections highlighted several drawbacks (Section II-B). We find due to the isolation of *tracking* and *optimization*, which are originally tightly intertwined in terms of data dependency, resource allocation, and thread management, the following challenges arise:

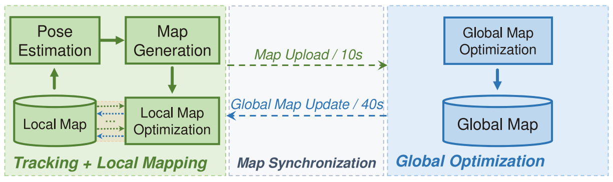
• *Map synchronization strains network bandwidth:* Real-time edge-mobile map synchronization facilitates mobile agents to access optimized local maps, crucial for maintaining tracking performance [13]. However, this synchronization, involving continuous streaming large volumes of map data (i.e., map points and keyframes) over wireless links, risks quickly saturating the constrained and congested wireless spectrum (Section II-B-C1).

• *Map update delay impairs localization accuracy:* The quality of local maps, which are crucial for the localization performance of mobile devices, hinges on prompt updates from edge-optimized maps [5]. However, the combined latency from on-network map data transmission and on-mobile map reconstruction results in delayed updates, potentially causing tracking drift (Section II-B-C2).

• *Map stitching deteriorates tracking determinism:* During map synchronization, mobile devices need to stitch received edge-optimized maps with their local ones. Typically, map



(a) Tracking-optimization decoupled architecture.



(b) Upgraded edge-assisted architecture.

Fig. 1. *Edge-assisted SLAM architecture comparison.* (a) The tracking-optimization decoupled architecture necessitates frequent data synchronization (i.e., map upload every 1 *s*, and local map update every 2 *s* [15]). (b) The upgraded architecture enables the tightly intertwined *tracking* and *local map optimization* to run concurrently on the mobile client, alleviating the map synchronization stress (i.e., upload every 10 s, update every 40 s on average).

updates and pose tracking are handled in separate threads [14], [15], employing locks to avoid read/write conflicts within the local map database. However, map stitching, can prolong the duration of map-locking, which results in unexpected tracking thread interruptions (Section II-B-C3).

Lessons Learned. The constrained computational capabilities of mobile devices necessitate positioning local map optimization at the edge in current *tracking-optimization* decoupled architectures (Fig. 1(a)). This leads to frequent mobile-edge map syncs, giving rise to the aforementioned challenges. To render edge-assisted SLAM more practical in challenging and network-limited environments, it's crucial to rethink the edge-assisted architecture – by integrating *tracking* and *local map optimization* on mobile (Fig. 1(b)), we can concurrently elevate system accuracy and efficiency, while minimizing resource overhead.

Recently, we find two opportunities to enhance the computing capability of lightweight mobile devices for such potential architectural upgrades. On the one hand, the software-hardware co-design paradigm has been widely adopted for mobile devices. Current innovations employ hardware resources (e.g., FPGA) to accelerate software algorithms and boost overall system efficiency. On the other hand, the proliferation of embedded SoCs (e.g., Xilinx Zynq [22], NVIDIA Tegra [23]) that offer heterogeneous arithmetic units are enabling software-hardware co-designs. These two trends propel *on-chip intelligence*, empowering lightweight mobile devices to handle intricate tasks [24], [25], [26].

Our Work. Motivated by the above challenges and opportunities, we design and implement *edgeSLAM2*, a fresh edge-assisted visual SLAM system that re-imagines edge-assisted architecture by tightly integrating *tracking* and local map *optimization* on mobile. edgeSLAM2 harnesses the heterogeneous computing units offered by the Zynq UltraScale+ MPSoCs [22] to enhance the computational capacity of mobile devices, and on this basis, accommodates such architectural upgrade through software-hardware co-design. As illustrated in Table I, a

TABLE I
EDGE-ASSISTED SLAM SYSTEM COMPARISON

System	Bandwidth(MB/s)	Accuracy(cm)	Latency(ms)
SwarmMap [13]	1.35	13.2 ± 5.3	32.2 ± 6.4
Edge-SLAM [14]	2.99	17.9 ± 5.9	34.3 ± 8.0
edgeSLAM [15]	4.49	11.3 ± 5.1	37.2 ± 10.9
edgeSLAM2	**0.27**	$\mathbf{7.6 \pm 2.9}$	$\mathbf{23.7 \pm 1.2}$

The bold values highlight the fact that our approach, by all metrics, outperforms the comparison work.

comparison with existing practices in oil-field scenarios showcases the effectiveness of edgeSLAM2. Overall, edgeSLAM2 excels in three aspects:

- On the Architecture front, we redesign task allocation between mobile and edge. Different from current practice, the *Local Map Optimization* module, which is tightly coupled with *Tracking*, is loaded to mobile devices. We further extract a lightweight *loop detection* module from the *global map optimization* and relocate it to mobile to decrease the triggering frequency and data volume for map synchronization, further enhancing efficiency (Section III).
- On the Algorithm front, we propose a new mobile-edge map synchronization solution compatible with the upgraded architecture. We first introduce *Event-Responsive Map Synchronization*, optimizing the timing and frequency of map synchronization under the new paradigm (Section IV-B). Then, we propose *Observation Consistency based Map Streamlining*, minimizing the transmission payload by selectively compressing the necessary map elements (Section IV-C).
- On the Implementation front, We fully utilize heterogeneous computing units to enable mobile tasks to run in real time (Section IV-A). Particularly, we propose a *Delay Deterministic Tracking* approach, leveraging FPGA and resource isolation strategy to accelerate critical modules in tracking and prevent its thread from being interrupted, to alleviate the tracking delay bottleneck (Section IV-D).

We deploy edgeSLAM2 on a drone testbed, and further integrate it into ArduPilot [27], a widely-used drone flight controller, for automated industrial inspections. Comprehensive experiments are carried out in one of the world's largest oil fields over three months, covering a variety of scenarios including warehouses, oil-producing areas, and factories, collecting 188 trajectories with 182,267 frames (Section V-A). We compare edgeSLAM2 with three state-of-the-art (SOTA) edge-assisted SLAM systems, SwarmMap [13], edgeSLAM [15], and Edge-SLAM [14]. Evaluation results show that edgeSLAM2 achieves an average bandwidth consumption of 0.27 MB/s, a localization accuracy of 7.6 cm, and a tracking delay of 23.7 ms. This performance surpasses competing methods by achieving an 80% reduction in bandwidth consumption, a 32% improvement in accuracy, and a 26% reduction in tracking delay (Section V-B). Furthermore, by embracing the upgrade edge-assisted paradigm, edgeSLAM2 inherently prevents scalability issues associated with expanding agent counts in multi-agent collaborative mapping tasks (Section V-F).

The key contributions are summarized as follows:

1) We design and implement edgeSLAM2, an innovative system that reshapes the edge-assisted visual SLAM paradigm by fully embracing on-chip intelligence.
2) We advance the core technologies of mapping, localization, and synchronization foundational to the new architecture through software and hardware co-design. Consequently, edgeSLAM2 empowers mobile devices to achieve accurate localization and mapping in real-time, even in network-challenged environments.
3) We leverage the efficient map-sharing capabilities of the upgrade architecture to enhance the map synchronization and compression mechanisms for multi-agent collaborative mapping, potentially serving as a foundation for multi-agent collaboration tasks under the new generation of edge-assisted SLAM paradigm.
4) We implement edgeSLAM2 and deploy it on industrial inspection drones. Our three-month pilot study in oil fields demonstrates that edgeSLAM2 makes a great process towards fortifying edge-assisted visual SLAM into a fully practical system for wide deployment.

II. Background and Motivation

A. Edge-Assisted Visual SLAM Systems

An edge-assisted visual SLAM system can be abstracted as mobile, edge, and network, three layers [13]. The current practices with tracking-optimization decoupled architecture can be summarized as below.

Front-End Tracking on Mobile. A mobile device receives video stream input, extracts feature points from each frame, and estimates the camera pose (i.e., *pose tracking*) by correlating these features with a pre-constructed local map (i.e., a set of 3D map points and keyframes).[1] Moreover, new map points are created and added to the local map (i.e., map tracking), aiding the following tracking process.

Back-End Optimization on Edge. On an edge server, a global map is maintained and persistently fine-tuned through both *local* and *global map optimization*. Specifically, local Bundle Adjustment (*local BA* [28]) is employed to optimize the uploaded local map, enhancing the accuracy of map point locations and keyframe poses. Simultaneously, *loop closing* [2] combined with global BA is leveraged to globally optimize the overall map and trajectory.

Map Synchronization Through Network. Newly generated map points and keyframes on mobile devices are uploaded to the edge server for either local or global optimization. The optimized map, in turn, is transmitted back to the mobile device for refining the local map.

B. Limitations of Current Practice

We conduct a field-study within the oil-fields, and re-implement three SOTA, open-source edge-assisted SLAM systems, SwarmMap [13], Edge-SLAM [14], and edgeSLAM [15], on an inspection drone. Through the evaluation of their performance using 232 trajectories (setup detailed in Section V), we identified three major issues stemming from the isolation of *Tracking* and *Local Map Optimization*.

C1: Excessive Bandwidth Consumption. Within edge settings, maintaining high-quality *Tracking* requires frequent map synchronization. In practice, the mobile client persistently uploads detected keyframes and newly generated map points to the edge. The edge server, in response, promptly optimizes the local map and returns the updates. This process implies frequent and intensive data transmission. For instance, SwarmMap executes updates every two seconds, which includes dozens of keyframes along with their associated map points [13].

We quantify the bandwidth usage of three related systems in diverse scenarios, depicted in Fig. 2(a). The data volume for map sync, even with a single mobile device, is considerable. This demand escalates in multi-agent collaborative scenarios, further outstripping the available network bandwidth. Although SwarmMap reduce map volume through designing compact map representations, it faces limitations as the map scale and complexity increase.

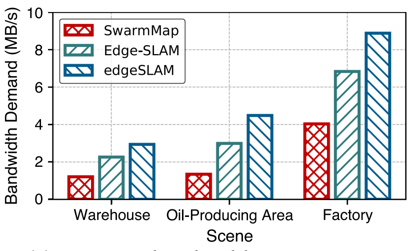
C2: Considerable Localization Error. In the *Tracking* module, the mobile device location is estimated by referencing a pre-constructed local map. Errors in map construction can lead to drift in localization, and conversely, inaccurate device location can adversely affect map incremental construction. Therefore, timely updates of the local map are crucial to prevent the bidirectional negative feedback between map offset and location drift. However, in existing architectures, local map updates are inevitably subject to delays caused by network transmission and map stitching processes.

To validate our analysis, we investigated the direct impact of map update delays on localization accuracy of Edge-SLAM, as demonstrated in Fig. 2(b). While the accuracy degradation due to map update delays in the warehouse is inconspicuous, as network quality degrades and map scale grows, Edge-SLAM experiences significantly longer map update times, leading to a considerable reduction in localization accuracy.

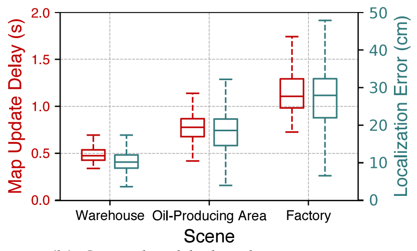
C3: Highly Dynamic Tracking Delay. The *Tracking* and *Local Map Optimization* modules share map resources and employ map-locking to avoid memory access conflicts between these two parallel threads. As such, when updating the map, *Tracking* must momentarily pause, waiting for the release of resources, and vice versa. However, unlike the direct access to map elements facilitated by local updates (e.g., through pointer access), edge map stitching necessitates an extensive search for each keyframe and map point to find all their references. This process prolongs the duration of shared map resource occupancy, subsequently stalling the real-time operation of *Tracking*.

We evaluate the tracking delay of three related systems over a continuous period of time, as shown in Fig. 2(c). Although all systems can maintain an average frame rate of up to 30fps, resource contention induced by map stitching profoundly impacts the tracking performance. Specifically, around the 280th frame, the tracking delay for all systems surged beyond 70 ms (e.g., $<$15 Hz). Additionally, inevitable procedures such

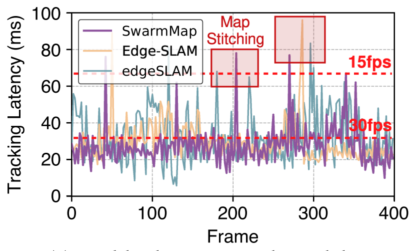
[1]Keyframes are a subset of frames that capture essential data like camera position, map point observations, and the visibility relationships with other keyframes.



(a) Excessive bandwidth consumption



(b) Considerable localization error



(c) Highly dynamic tracking delay

Fig. 2. *Limitations of current practice observed in our field study.* (a) As synchronization frequency and map scale increase, each agent requires over 4 MB/s bandwidth in factory scenarios, $> 2\times$ that in simpler warehouse scenarios. (b) Current edge-assisted SLAM systems [14] suffer notable map update delays, leading to substantial localization errors (i.e., exceeding 1.5 s and 40 cm in the factory). (c) Resource contention arising from map stitching, which occurs at certain frequencies, induces significant spikes in tracking delay.

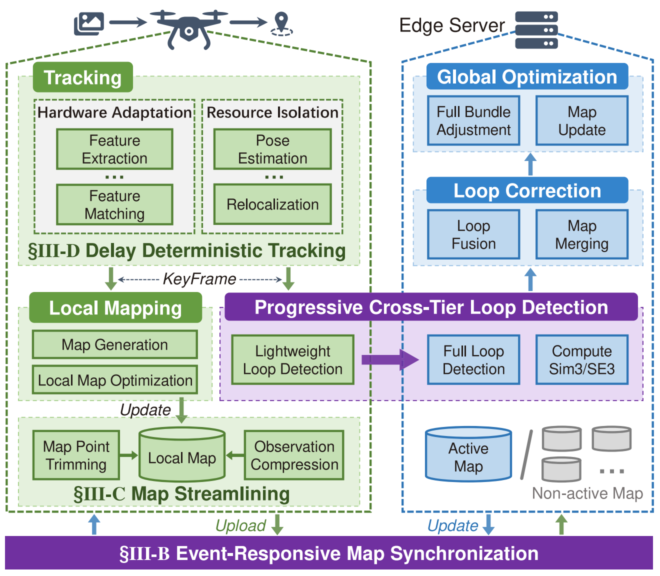


Fig. 3. edgeSLAM2 overview.

as map generation, serialization, and transmission chip away at the limited computational resources intensifying the uncertainty of tracking delay.

III. System Overview

edgeSLAM2 upgrades the tracking-optimization decoupled architecture (Fig. 1(a)) by transferring the *local map optimization* from edge to mobile, as shown in Fig. 1(b). edgeSLAM2's architecture is outlined in Fig. 3. From a top perspective, edgeSLAM2 shares the similar system abstraction of mobile, edge, and network layers, and is built upon the latest ORB-SLAM3 [2]. We delve into the specific workflow in this re-imagined edge-assisted visual SLAM architecture, and summarize the novel functional modules designed to support architectural upgrade across three layers.

- *On the Mobile layer,* edgeSLAM2 applies a software-hardware co-design on a heterogeneous computing chip (Section IV-A). It integrates FPGA-adapted essential algorithms and resource isolation for *delay deterministic tracking* (Section IV-D), while retaining most generic computing resources for *local map optimization*. Additionally, an initial loop recognition is conducted by a *lightweight loop detection* module.
- *On the Edge layer,* edgeSLAM2 undertakes an complete loop verification upon the arrival of a loop detection signal from the mobile client, subsequently initiating *loop correction* and *global optimization*. Moreover, in the multi-agent collaborative mapping, edgeSLAM2 carries out map stitching when overlaps in maps from various agents are identified.
- *On the Network layer,* edgeSLAM2 refrains from frequent uploads of newly generated maps. Instead, it implements an *event-responsive map synchronization* strategy (Section IV-B) to refine the timing and frequency of synchronization. To further enhance efficiency, an *observation consistency based map streamlining* approach (Section IV-C) is employed for effective map compression before synchronization.

IV. Software and Hardware Co-Design

In this section, we first introduce the on-chip design of edgeSLAM2 supporting the paradigm shift (Section IV-A), covering both the on-chip architecture and workflow. Next, we propose a map synchronization strategy (Section IV-B) and a map streamlining method (Section IV-C) that work together to improve map synchronization efficiency in the upgrade edge-assisted SLAM paradigm. Finally, we explain how the on-chip hardware and software resources are utilized to ensure the latency determinism of the critical tracking module (Section IV-D).

A. edgeSLAM2's On-Chip Architecture

We utilize the most recent iteration of the commercially available Zynq UltraScale+ MPSoC (hereafter referred to as MPSoC), a heterogeneous computing platform pioneered by Xilinx [22], to implement the mobile side of edgeSLAM2 via software-hardware co-design. We initially provide a succinct overview of the MPSoC platform, following which we delineate the on-chip architecture of edgeSLAM2.

1) Zynq Platform Primer: Fig. 4 depicts the hierarchical computational resources offered by the MPSoC. As illustrated, the MPSoC is comprised of two modules: a Processing System (PS), purposed for software development, and User-Programmable Logic (PL), intended for hardware design. The

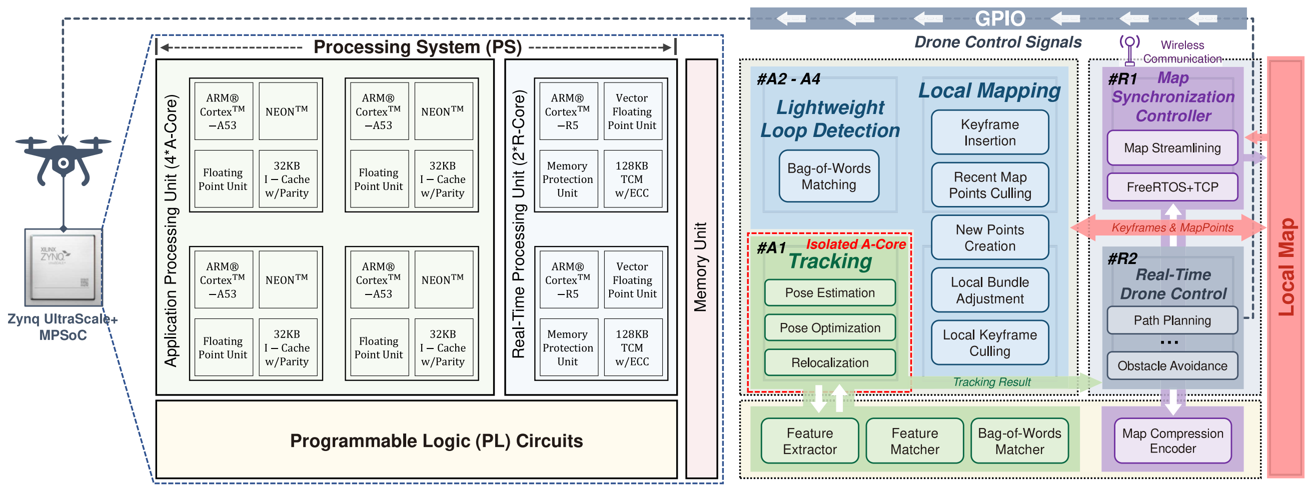


Fig. 4. *edgeSLAM2's on-chip architecture*. The left side exhibits the Zynq UltraScale+ MPSoC, while on the right, the functional modules of edgeSLAM2 are spatially mapped one-to-one with the heterogeneous computing cores of the MPSoC.

PS is equipped with a Cortex-A53 quad-core processor (4*A-Core) and a Cortex-R5 dual-core processor (2*R-Core). Typically, the Linux OS (e.g., PetaLinux, Debian) is employed for centralizing the four A-Cores, whereas a Real-Time Operating System (RTOS) is used for scheduling the two R-Cores, designed specifically for real-time applications. On the other hand, the PL offers programmable logic blocks, advanced digital signal processing (DSP), and other resources specifically designed for hardware development and customization.

2) Architecture: The on-chip architecture of edgeSLAM2, as depicted in Fig. 4, is deftly integrated with the computational capabilities of MPSoC. First, *Tracking* is accomplished through the collaboration of PS and PL: the PL is responsible for executing repetitive and parallelizable modules such as feature extraction and matching, while the dedicated #A1 serves as the host, performing control and optimization tasks.

edgeSLAM2 allocates most of the general-purpose computational resources to *Local Mapping* on #A2-A4 in PS, a process that is resource-intensive involving both map generation and local map optimization. Furthermore, *Lightweight Loop Detection*, a non-latency-sensitive task, shares #A2-A4 with *Local Mapping* under the management of the OS.

We employ a map synchronization controller on #R1. This controller primarily serves two functions: leveraging the map compression encoder provided by the PL for real-time map compression (Section IV-C), and executing map transmission based on the specified synchronization strategy (Section IV-B).

Lastly, the #R2 hosts the real-time drone control module, receives pose information from the *Tracking* module (#A1), plans the flight path, and transmits control signals through the General Purpose Input/Output (GPIO).

3) Workflow: The workflow of edgeSLAM2, demonstrated in Fig. 5, aligns with a typical SLAM pipeline and emphasizes both parallelism and pipelining. Upon receipt of an input (e.g., N_{th} frame), the *Tracking* module employs a specific hardware accelerator (PL) through an Advanced eXtensible Interface (AXI) for feature extraction and matching, followed by pose

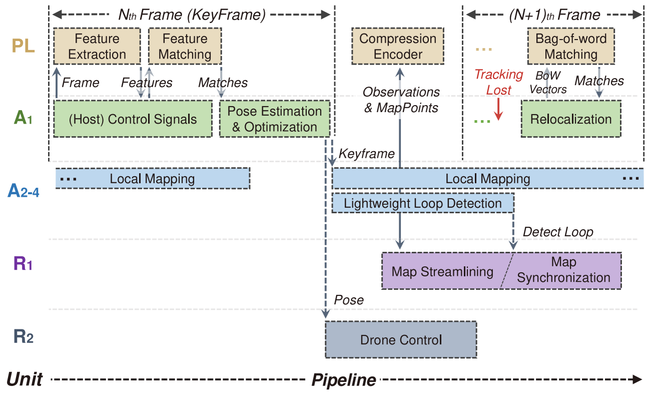


Fig. 5. A typical on-chip workflow of edgeSLAM2.

estimation and optimization on #A1. The estimated pose information is then passed to the flight control module (#R2) for downstream tasks. If the N_{th} frame is selected as a keyframe, two operations run concurrently: (i) *Local Mapping* on #A2-A4 for local map generation and optimization; (ii) *Lightweight Loop Detection* for initial loop identification. Upon loop detection, map streamlining on #R1, in conjunction with the compression encoder (PL), compresses the yet-to-be-synchronized map. This streamlined map is then dispatched to the edge server for further verification and global optimization. Additionally, in the case of *Tracking Lost*, the *Relocalization* module on #A1 utilizes the Bag-of-Words matching module (PL) to recalibrate the current location.

B. Event-Responsive Map Synchronization

To determine the optimal timing for map synchronization, we monitor and respond to specific events within the map. This approach aims to: (i) ensure timely execution of loop detection and global optimization, maintaining the quality of maps; (ii) facilitate the swift sharing of locally constructed maps with

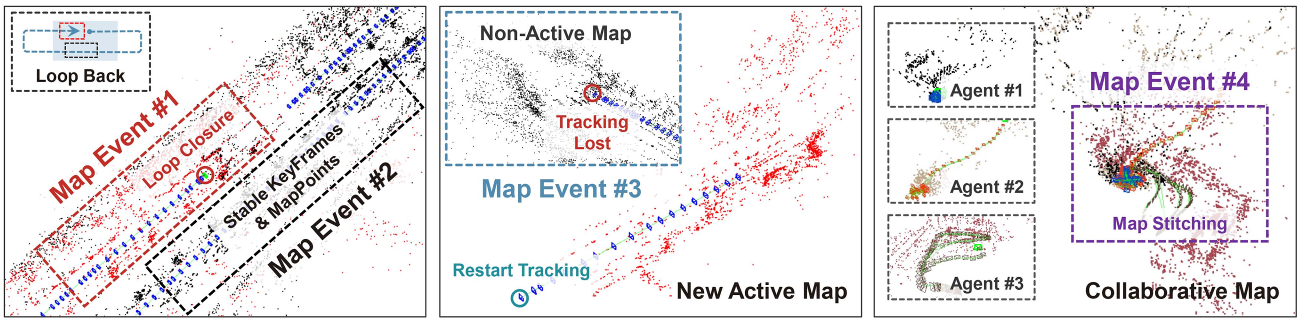


Fig. 6. *Event-Responsive Map Synchronization.* In these scenarios, Events #1 and #2 occur within loop closure, while Event #3 is triggered by *Tracking Lost*, resulting from swift shifts in perspective. Event #4 is set off after the merging of local maps from three distinct agents at the edge. For Events #1-3, red dots represent active map points detected by the current frame and co-visible keyframes, whereas black dots signal optimized, stable map points that are out of observation. In Event #4, map points are color-coded in three different hues to distinguish the local maps contributed by each agent.

other clients via centralized edge server, promoting cooperative tasks; and (iii) minimize redundant data transfers, maximizing synchronization efficiency. Motivated by these objectives, we focus on four specific map events from both mobile and edge side, and suitable for multi-agent collaborative scenarios. We showcase typical examples of four map events in Fig. 6.

Event #1: Progressive Cross-Tier Loop Detection. A lightweight loop detection module is utilized on the mobile client for initial identification of potential loops, with subsequent resource-intensive processes (i.e., loop correction, global optimization) are offloaded to the edge server.

Specifically, we use Bag-of-Words vector matching (i.e., the first step in full loop-closure [2]) for initial keyframe comparison within the local map, pinpointing similar keyframes that may indicate a loop. Upon loop detection (Event #1), the mobile device retrieves and immediately uploads the unsynchronized segment of the current local map. When the edge server receives this map segment along with the loop closure signal, the subsequent steps vary based on the loop type. For an intra-map loop closure (i.e., revisiting a location within the current active map), loop correction and global optimization are swiftly initiated. Conversely, with an inter-map loop closure (i.e., identifying overlap between the current active and a non-active map), the maps are merged prior to global optimization. Following the completion of optimization, the updated map is synchronized back to the mobile device, thereby replacing the local map.

Loop-closure not only directly rectifies the local map offset and localization drift but also simultaneously optimizes global mapping in multi-agent collaboration. Therefore, this map-event is assigned the highest priority in the Map Synchronization Controller (#R1).

Event #2: Map-Segment Stabilization. In visual SLAM, active map elements (e.g., current frames alongside their co-visible keyframes, and observed map points) frequently change due to local map optimization on the mobile device. These yet-to-stabilize elements, even when synchronized to the edge server, are rapidly superseded by newer updates, resulting in the wasteful use of network resources with minimal contribution to collaborative mapping. Therefore, we periodically upload map segments deemed stable, thereby preventing redundant uploading of map elements.

Specifically, we employ a Least Recently Used (LRU) strategy [29]. We maintain a queue of keyframes, whenever a new keyframe is chosen, triggering local map optimization, we shift any modified keyframe to the front of the queue. Simultaneously, keyframes that have *cooled down* are shifted from the queue's rear to the ready-to-sync map segment. When this segment comprises N_{kf} consecutive keyframes (Event #2), we upload the keyframes and their related map elements from that segment.

Our approach of periodically uploading continuous map segments, as opposed to individual frames, aims to enhance map compression efficiency (Section IV-C). Selecting a larger value for N_{kf} yields higher compression ratios, albeit at the expense of real-time synchronization in collaborative mapping.

Event #3: Local Map Inactive. We handle two specific scenarios where the currently tracked local map becomes inactive (Event #3):

(i) When the mobile device completes its map exploration task, and the local map is no longer subject to modifications (e.g., map creation, optimization), at which point the remaining unsynchronized map is uploaded to fulfill the final task of collaborative mapping.

(ii) In cases where tracking is lost during exploration and a new sub-map is initiated for tracking restart, the original local map transitions to a non-active state. Its residual parts are then uploaded and the non-active map are stored on the edge server, awaiting reactivation triggers (i.e., loop closure, map stitching).

Event #4: Collaborative Map Updating. On the edge side, edgeSLAM2 considers map update events from multi-agent collaborative mapping, yet refrains from synchronizing every incremental global map change with clients. Because these incremental map segments, while subject to potential inaccuracies and offsets, often undergo several rounds of optimization. Consequently, their frequent transmission could result in substantial data redundancy.

Therefore, edgeSLAM2 selectively tackles multi-agent mapping updates on the edge side that significantly improve the client's local map (Event #4). This primarily involves *map stitching*, as illustrated in Fig. 6, where the edge server identifies overlaps and merges map segments from various clients. For simplicity, we reuse the stitching approach proposed in [16], yet edgeSLAM2 remains adaptable to other plug-and-play map

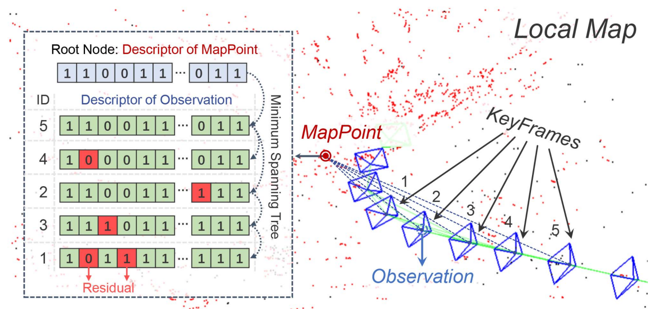


Fig. 7. *Observation consistency based map streamlining*. Along the camera's trajectory, multiple keyframes (i.e., id 1-5) observe and triangulate the *MapPoint*. The minimum spanning tree is constructed with the *MapPoint* as the root and descriptors from the five observations. Residuals between the current observation and its parent node are highlighted in red blocks.

merging modules. After the map stitching process, only the local maps that have been merged with others are synchronized and updated. Besides, the completion of loop closing and mapping task by any client, indicating a significant update for global map, triggers the synchronization of the enhanced global map with all associated clients.

C. *Observation Consistency Based Map Streamlining*

Under the revised edge-assisted architecture and map synchronization strategy, it becomes imperative to reconsider how map elements are compressed during upload/download. Existing edge-assisted SLAM systems reduce data volume through streamlined mapping strategies, namely, stringent constraints on feature extraction and keyframe selection [5]. In contrast, edgeSLAM2 does not intervene in map construction to enhance compression. Instead, it capitalizes on the benefits of its synchronization strategy, where the map segments queued for synchronization are of substantial size. This allows for the identification and elimination of redundancy in map elements, thereby streamlining the map from a holistic perspective.

Specifically, in a continuous visual SLAM scenario, any created map point is observed from multiple keyframes, each represented by a similar descriptor.[2] These observations are associated through feature matching and triangulated to generate corresponding 3D map point, as depicted in Fig. 7. Leveraging the similarity in observations of the same map point across various keyframes, we (i) compress the feature descriptors, the predominant storage component, and (ii) selectively retain the most informative map points, balancing compression ratio and map integrity.

1) Observation Compression Coding: When a map segment is selected for upload, the compression process starts by extracting the *essential map* from the full map, specifically by removing features not associated with any map point (about 70% of the total map storage), as they typically hold no relevance for map optimization or cooperative mapping.

[2]In the ORB feature, the descriptor represents the image patch surrounding the keypoint with a binary string of 256 bits in length.

Following this, for each map point in the essential map, we compress and encode its associated observations. Initially, we calculate the Hamming distance between descriptors to assess the similarity of observations. Then, using the map point as the root and the distances between descriptors as edge weights, we construct a minimum spanning tree with observations as nodes, as illustrated in Fig. 7. This method ensures that starting from the map point, each connection in the tree is based on the highest data similarity. Similar to [30], we traverse the spanning tree iteratively, calculate the residuals between neighboring descriptors, and employ arithmetic coding to compress these residuals.

The effectiveness of integrating this compression method within edgeSLAM2's on-chip system is grounded in two key factors. First, the minimal residuals between observations yielded by the spanning tree, when combined with arithmetic coding, enable efficient map compression, effectively mitigating the redundancy among similar observations. Second, our event-responsive map synchronization strategy (Section IV-B) facilitates the upload of sufficiently large map segments, which assures the scale of the spanning tree (i.e., a map point with adequate observations), thereby enhancing compression efficiency. Additionally, we employ a *Map Compression Encoder* on the PL to alleviate the computational costs associated with Hamming distance calculation and arithmetic encoding [31].

2) Map Point Trimming: Utilizing observation consistency allows us to meticulously trim the map segments awaiting synchronization, focusing on keeping those map elements of superior quality. This strategy aims to decrease both synchronization and storage overheads without adversely affecting the map's accuracy. To elaborate, we prioritize retaining map points with: (i) High visibility across multiple keyframes, as this often signifies their valuable contribution to localization and map refinement; (ii) Lower storage requirements after compression coding among those with comparable observation counts, reflecting greater stability (i.e., the stronger the observation consistency, the more efficient the compression).

Specifically, given a map point p to be synchronized. The number of its observations is N_p, and its total encoded length (including the map point itself and related compressed observations) is L_p. We denote the weight of p as $W_p = (N_{max} - N_p) * L_p$, where N_{max} corresponds to the maximum number of observations linked to any map point in the segment. We prioritize maintaining map points with smaller W_p. Starting with the lowest quality map point (indicated by the highest W_p), we attempt to remove map points and their observations iteratively. To ensure that all keyframes retain their localizability following the removal of map points, a sufficient number of observations must be preserved. Therefore, in this ordered removal, we preserve those map points whose absence would lead to any keyframe's observations falling short of N_{obs}, a threshold set at 40 in our configuration. By tuning the map point trimming rate, we balance compression efficiency with localization accuracy, as evaluated in Section V-D2.

This simple yet effective strategy not only alleviates the burden on the uplink from mobile devices but also proves effective in streamlining the aggregated maps on the edge server. In the multi-agent collaborative mapping scenario, despite the

Algorithm 1: Adaptation of Matching on PL.

```
Input: Feature Descriptors:
       $\mathcal{D}_f = \{D_f\}, \mathcal{D}_m = \{D_m\}$
Output: Matching Results: $\mathcal{M}$
for each $D_f$ in $\mathcal{D}_f$ do // k-Way Loop Unroll
    initialize $\mathcal{M}_{\text{local}} = \emptyset$, $dis_{\min} = +\infty$;
    for each $D_m$ in $\mathcal{D}_m$ do
        $dis = $ Hamming$(D_f, D_m)$; // Parallel XOR ports + Adder tree
        if $dis < dis_{\min}$ then
            $dis_{\min} = dis$; $M_f = D_m$;
        end
    end
    if $dis_{\min} < \tau_{threshold}$ then
        $\mathcal{M}_{\text{local}}$.add$([D_f, M_f])$;
    end
end
$\mathcal{M} = $ ParallelReduction$(\{\mathcal{M}_{\text{local}}\})$;
```

local maps from mobile clients being pre-trimmed, further streamlining on edge side is possible due to (i) repeatedly visited areas in global map introducing redundancy, and (ii) the better compression capabilities when applied to fuller maps. Consequently, we apply the same compression and trimming strategies to the aggregated global map on the edge side, ensuring the collaborative map remains streamlined. However, unlike the smaller map segments on the mobile devices, streamlining the global map requires extensive computational resources over a prolonged period. To avoid obstructing crucial optimization tasks, we streamline the global map only during offline stages (i.e., after all agents have completed their mapping tasks).

D. *Delay Deterministic Tracking*

In the critical *Tracking* module in edgeSLAM2, we leverage the heterogeneous computational resources on-chip to (i) effectively accelerate time-consuming algorithmic bottlenecks in tracking, ensuring real-time performance, and (ii) maintain smooth tracking operations, unhindered by resource contention, guaranteeing tracking determinism.

1) Hardware Adaptation of Tracking: In the tracking process, feature extraction and matching constitute the computational bottlenecks [32]. Specifically, during feature extraction, ORB features [33] are obtained from the input image via a combination of FAST keypoint and BRIEF descriptor, and gain rotational and scale invariance through orientation adjustment and pyramid creation, respectively. During feature matching, each detected feature in the current frame seeks to match with a 3D map point in the local map, based on the Hamming distances between their BRIEF descriptors.

Feature Extraction. We restructure the ORB feature extraction algorithm to align with hardware processing capabilities. As shown in Fig. 8(a), we downsample the input image and conduct parallel processing on the generated 4-layer pyramid. For each layer, we first *detect* FAST keypoints within the image, then *calculate* the rotationally symmetric BRIEF descriptors [34] of these keypoints, and finally *filter* the optimal feature points based on Harris scores [35] through a max heap. This rescheduled approach, as opposed to the conventional CPU workflow of filtering keypoints before calculating descriptors, allows for simultaneous execution of keypoint detection and descriptor calculation, as depicted in Fig. 8(b). This efficient pipeline significantly reduces hardware idle periods, thereby optimizing feature extraction latency.

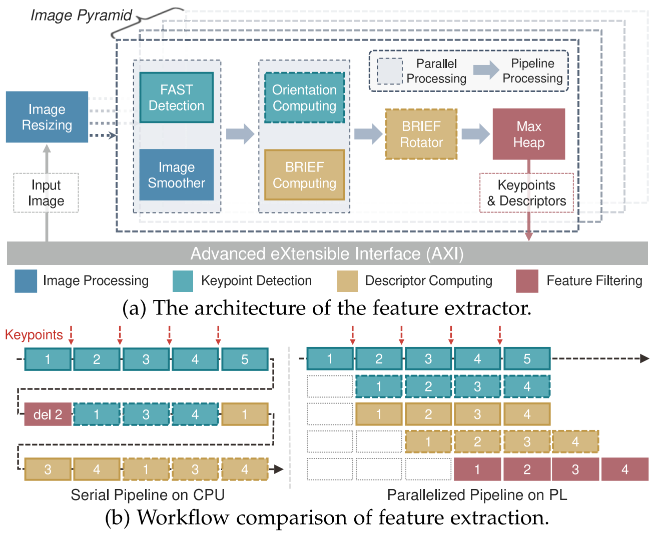


Fig. 8. *Adaptation of feature extraction on PL.* (a) This highly parallelized module receives images from AXI and outputs extracted features (i.e., keypoints with descriptors). Different line styles and colors denote varied algorithm modules. (b) Numbers (1,2,3,4) symbolize detected keypoints. In the serial pipeline, keypoints are detected, filtered (delete 2), and then descriptors (1,3,4) are computed. In the parallelized pipeline, the keypoint extraction and descriptor computing are performed concurrently, followed by feature filtering.

Feature Matching. Our adaptation for hardware also extends to feature matching. In Algorithm 1, we detail our strategy for parallelization and the associated hardware design. Specifically, the input includes two sets of descriptors, $\mathcal{D}_f$ and $\mathcal{D}_m$, derived from the current frame and local map, respectively. By utilizing a 4-way loop unroll, the algorithm in parallel determines the optimal match in $\mathcal{D}_m$ for every descriptor in $\mathcal{D}_f$ (Line 1). The similarity between the two descriptors is gauged using Hamming distance via the `Hamming` function (Line 3), a pipeline structure formed from parallel XOR ports and a pipelined adder tree, capable of executing a distance computation in every clock cycle. We then track the descriptor in $\mathcal{D}_m$ nearest to D_f (Line 4–7) and perform a final evaluation against the matching threshold (Line 8–11). At Line 13, we use parallel reduction [36] for concurrent updates to $\mathcal{M}$. Similarly, to accelerate the relocalization process, we also implement a hardware-adapted Bag-of-Words matcher, operating under a parallel routine akin to the above feature matching strategy.

2) Resource Isolation: Ensuring uninterrupted and timely execution of the time-sensitive *Tracking* thread on the CPU core is challenging due to the concurrent execution of other background threads, such as *Local Mapping* and *Lightweight loop detection*, controlled by the same operating system. This competition for computational resources can introduce additional end-to-end latency. To minimize such disturbances, we dedicate

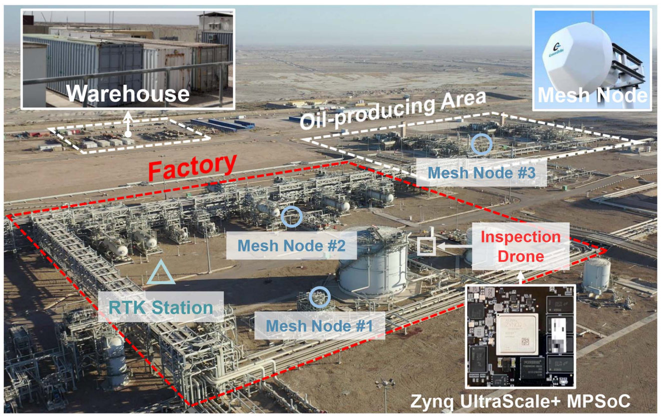


Fig. 9. Industrial inspection is carried out by drones equipped with edgeSLAM2 within one of the world's largest oil fields.

TABLE II
DETAILS OF DATA COLLECTION IN DIFFERENT SCENARIOS

Scene Type	No.of Path (Total)	No.of Path (with Loop)	No.of Frames	Flight Speed (Avg. m/s)
Warehouse	36	32	34,900	0.6
Oil-Producing Area	71	56	63,012	4.8
Factory	81	68	84,355	7.0

one A-Core exclusively for *Tracking*. In our implementation, we realize A-Core isolation by building the Linux OS with the boot parameter `isolcpus=<cpu #A1>`.

Furthermore, simultaneous map access by both *Tracking* and *Optimization* can introduce contention, subsequently increasing tracking latency. To mitigate these effects, we adopt a double map buffering strategy [37]. This methodology utilizes two dedicated memory reservoirs: one hosting the current map necessary for tracking, and another storing the map being updated through optimization. Upon each optimization completion, the refreshed map is shifted to its corresponding buffer, allowing the tracking thread to transition smoothly to this updated map. This approach ensures uninterrupted access and freshness of map data for the tracking process.

V. EVALUATION

A. Experimental Methodology

Field Studies. We incorporate edgeSLAM2 into the ArduPilot APM flight controller and deploy it on AMOVLAB P450-NX drones. We conduct a field study spanning three months, delivering real-time localization services for industrial inspection tasks in one of the world's largest oil fields, as shown in Fig. 9. We select three representative scenarios for detailed system performance evaluation, collecting 188 trajectories with 182,267 frames, as summarized in Table II. The warehouse represents a typical indoor environment, while the oil-producing area and factory exemplify complex industrial outdoor settings. The drones communicate with the edge node via 2.4 GHz WiFi in indoor environments, while in outdoor settings, it switches to a mesh network. The maximum throughput in the outdoor mesh and indoor WiFi networks is 14.3 MB/s and 26.8 MB/s, respectively. The edge side of edgeSLAM2 is deployed on an Nvidia Jetson AGX Xavier edge node, with its power consumption capped at 30 W, within the range of available power supply in industrial settings.

Metrics and Ground Truth. To evaluate system overhead, we measure the *bandwidth requirement* (in MB/s), defined as the average volume of data transferred per second. We assess the real-time performance by recording the *tracking latency*, denoting the time from image receipt to pose output. The localization accuracy is determined using the *Absolute Trajectory Error* (ATE, in cm), a gold standard in SLAM algorithm evaluation [38]. The ground truth for indoor localization is obtained through Opti-Track [39], whereas Real-Time Kinematic (RTK) is utilized for outdoor environments.

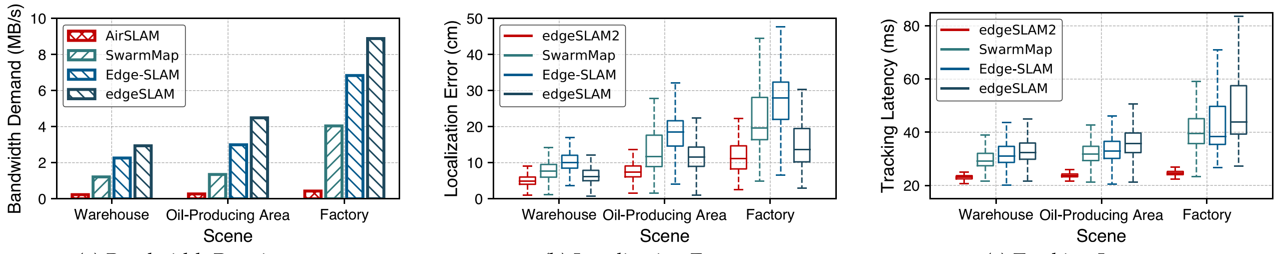
Baselines. We compare edgeSLAM2 with three SOTA edge-assisted SLAM systems, SwarmMap [13], Edge-SLAM [14], and edgeSLAM [15]. Despite these systems not being designed for the Zynq MPSoC, we ensure a fair comparison by deploying them on the same platform. In our setup, we deploy the Petalinux OS on the 4*A-Core and utilize OpenAMP for controlling the 2*R-Core, fully exploiting the general computing resources [40]. Beyond this distinction, they operate under the same edge server and network conditions as edgeSLAM2.

B. Overall Performance

1) Bandwidth Requirement: We first evaluate the bandwidth requirement of edgeSLAM2 and the three baselines in different scenarios. As shown in Fig. 10(a), edgeSLAM2 requires an average bandwidth of 0.23 MB/s, 0.27 MB/s, and 0.43 MB/s in the warehouse, oil-producing area, and factory settings, respectively. Compared to the baselines, edgeSLAM2 achieves a bandwidth reduction of at least 81.3%, 80.2%, and 89.4% respectively. Such performance enhancement is credited to the implementation of the upgraded edge-assisted SLAM paradigm, paired with (i) event-responsive map synchronization and (ii) observation consistency based map streamlining.

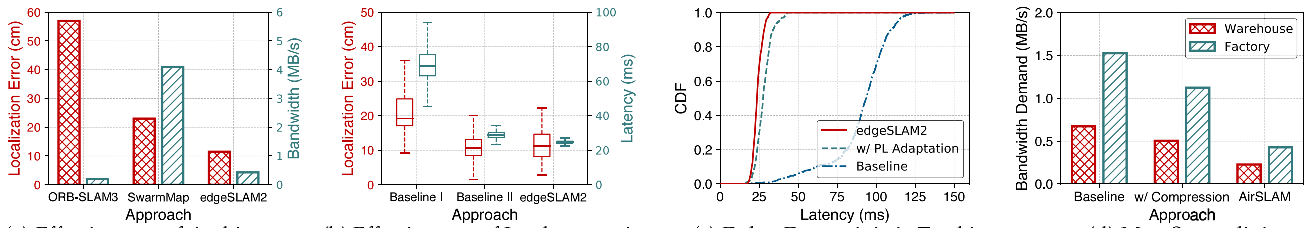
2) Localization Accuracy: Fig. 10(b) depicts the localization performance of edgeSLAM2 and comparative systems in different settings. The average localization error of edgeSLAM2 is 5.3 cm, 7.6 cm, and 11.5 cm in the warehouse, oil-producing field, and factory, respectively. edgeSLAM2 outperforms three baselines across all scenarios, particularly in the challenging factory setting, which is characterized by poor network quality and large-scale maps. Specifically, edgeSLAM2 outperforms SwarmMap, Edge-SLAM, and edgeSLAM by 48.5%, 57.9%, and 22.6%, respectively. This superiority can be attributed to the innovative architecture that re-couples tracking and local map optimization, which effectively mitigates potential performance degradation caused by map update delay.

3) Tracking Latency: We also evaluate the tracking latency of edgeSLAM2 with the baselines. As shown in Fig. 10(c), edgeSLAM2 achieved an average latency of 23.7 ms and 24.6 ms in the oil-producing area and factory, outperforming baselines by at least 26.3% and 44.8%. Furthermore, the corresponding 95th percentile tracking latency in these three settings are 24.4



(a) Bandwidth Requirement (b) Localization Error (c) Tracking Latency

Fig. 10. Overall performance comparison.



(a) Effectiveness of Architecture (b) Effectiveness of Implementation (c) Delay Deterministic Tracking (d) Map Streamlining

Fig. 11. Ablation study.

ms, 25.2 ms, and 26.0 ms, respectively, which is decreased by >53.4%, >41.4%, and >62.1% compared to the baselines. The above results underline the effectiveness of the delay deterministic tracking strategy in enhancing both the real-time and deterministic capabilities of edgeSLAM2, facilitated by the software-hardware co-design approach.

C. Ablation Study

We conduct several experiments within the challenging factory setting to evaluate the effectiveness of edgeSLAM2's architecture, implementation, and algorithms.

1) Effectiveness of Architecture: To evaluate the effectiveness of the upgraded edge-assisted SLAM architecture, we compare edgeSLAM2 with two different architectural baselines on the Zynq platform: ORB-SLAM3, where all tasks run entirely on the mobile client with only the final constructed map being synced, and SwarmMap, a recent method that adheres to the tracking-optimization decoupled approach. As shown in Fig. 11(a), ORB-SLAM3 syncs minimal data but suffers substantial accuracy degradation due to resource exhaustion from concurrent optimization operations. In comparison, edgeSLAM2 significantly reduces localization error and bandwidth usage by 49.3% and 89.5%, respectively, compared to SwarmMap. These results underscore the pivotal role of the upgraded edge-assisted architecture in edgeSLAM2.

2) Effectiveness of Implementation: To evaluate the effectiveness of edgeSLAM2's on-chip implementation, we replaced the Zynq MPSoC on the mobile with a relatively lightweight Jetson TX2 (Baseline I) and a high-performance Intel i7-9700 processor (Baseline II), while keeping the edge server configuration unchanged. This assessment examines the changes in localization accuracy and latency performance of edgeSLAM2 under different implementation scenarios.

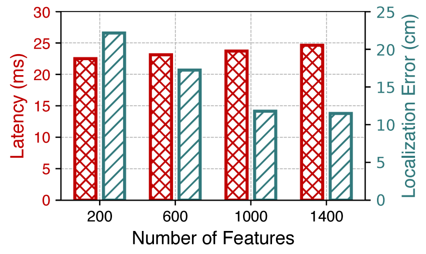
As shown in Fig. 11(b), owing to the superior processing capabilities of i7-9700, Baseline II surpasses Baseline I in both accuracy and latency aspects. On the other hand, edgeSLAM2 showcases a substantial reduction in the 95th percentile tracking latency by 18.8%, compared to Baseline II, incurring only a minimal decrease in localization accuracy by 0.95 cm due to local map optimization delay.

We further evaluate the implementation of *Tracking* in edgeSLAM2 (Section IV-D). As shown in Fig. 11(c), employing hardware adaptation on PL yields an average latency reduction of 68.9%. Furthermore, by applying both hardware adaptation and resource isolation strategies, the 95th percentile tracking latency of edgeSLAM2 is further reduced by 8.5 ms. The above results manifest the effectiveness of the software-hardware co-design implemented on Zynq MPSoC.

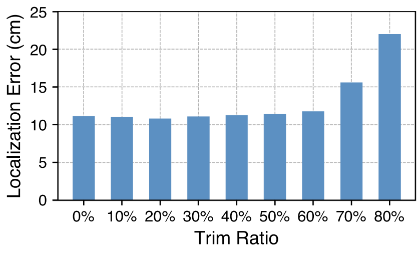
3) Effectiveness of Algorithm: In warehouse and factory environments, we evaluate the bandwidth savings from the introduced map streamlining strategy, which includes compression and trimming techniques. In these experiments, the baseline method transmits only the *essential map* segments for synchronization. As depicted in Fig. 11(d), applying observation compression coding (w/ Compression) reduces bandwidth usage by more than 24.9% compared to the baseline. Integrating this with map point trimming (w/ Both) further decreases bandwidth consumption by over 55.1%. Furthermore, the map streamlining strategy proves to be more effective in complex factory environments than in indoor scenarios, achieving a 72% data volume compression and an average bandwidth saving of over 1 MB/s.

D. Parameter Study

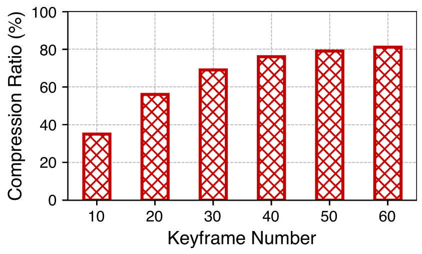
We conduct a parameter study to understand the impact of crucial parameter selection on the system performance.



(a) Impact of Number of Features



(b) Impact of Map Trimming Ratio



(c) Impact of Map Segment Length

Fig. 12. Parameter study.

1) Impact of Number of Features: In visual SLAM, the number of feature points extracted per frame plays a crucial role in the system's performance. On one hand, extracting more feature points allows for the association and creation of more 3D map points, typically resulting in a more comprehensive map and improved localization accuracy. However, extracting additional feature points consumes more computational resources and increases latency.

We evaluate the impact of the number of features on system accuracy and latency. As indicated in Fig. 12(a), opting for 1,000 feature points over 200 leads to a 10.4 cm decrease in localization error and a minor 1.2 ms increase in tracking latency. The slight increase in latency is attributable to edgeSLAM2's hardware-accelerated feature extractor, which processes feature points in parallel (Section IV-D1). Utilizing a higher number of feature points (>1000) does not yield significant improvements in accuracy. Therefore, we identify 1,000 feature points as the optimal number for extraction.

2) Impact of Map Trimming Ratio: In the process of map point trimming (Section IV-C2), our strategy of sorting map points by weight W_p and progressively releasing them allows us to manage the proportion of map points retained. Typically, removing lower-quality map points does not affect localization performance, yet an excessive trimming ratio could compromise localization stability. As demonstrated in Fig. 12(b), when the trimming ratio reaches 30%, the localization error reduces by 3.2% compared to the original untrimmed map. At a 60% trimming ratio, the error marginally increases by only 0.64 cm. However, escalating the ratio from 60% to 70% incurs a notable spike in localization error by 3.8 cm. Therefore, we find a trimming ratio of 60% to be optimal in practice.

3) Impact of Map Segment Length: Although the upgrade architecture does not necessitate real-time synchronization for local map optimization, edgeSLAM2 still uploads stable map segments to facilitate multi-agent collaborative mapping tasks (Section IV-B Event #2). We examine how the map segment length affects the data compression ratio, as illustrated in Fig. 12(c). Larger map segments lead to more efficient map streamlining but also result in lower frequencies of collaborative map updates. In oil field scenarios, a compression ratio of nearly 80% is achieved when the keyframe count reaches 40, albeit with an update delay of about 20 s. In the multi-agent inspection task, edgeSLAM2 opts for a keyframe number of 20 to strike a balance between compression efficiency and the frequency of collaborative mapping.

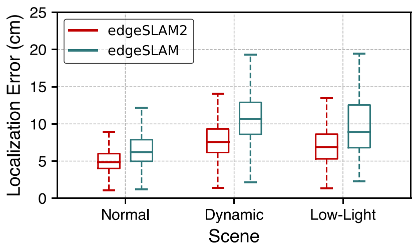


Fig. 13. System robustness evaluation.

E. System Robustness Evaluation

We evaluate the robustness of under interference conditions in a warehouse environment by selecting three common scenarios: *Normal*, representing typical conditions with normal lighting and no dynamic objects; *Dynamic*, representing a scenario with workers moving around, which interferes with the SLAM to extract stable features; and *Low-light*, where a lack of illumination significantly reduces the number of usable visual features.

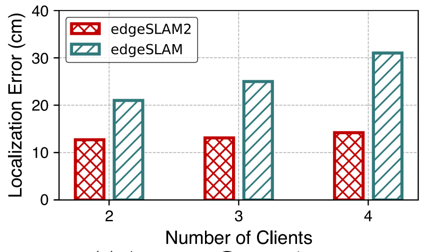
As shown in Fig. 13, the localization accuracy of edgeSLAM2 decreased by 45.3% and 32.1% in the dynamic and low-light scenarios compared to the normal scenario, respectively. In contrast, Edge-SLAM's accuracy decreased by 61.5% and 49.2% under the same conditions.

Although edgeSLAM2 does not employ specialized algorithms for these challenging scenarios, its upgrade edge-assisted SLAM architecture enables faster map optimization compared to traditional architectures. This capability helps mitigate the rapid decline in map quality and localization accuracy in adverse environments.

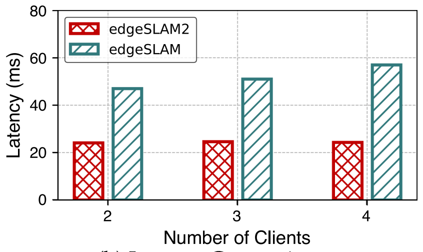
F. Multi-Agent Collaborative Mapping

The architectural design of edgeSLAM2, coupled with its synchronization and compression algorithms, empowers it to naturally support efficient multi-agent collaborative mapping. To verify the system's performance in practical applications, we utilize several drones for a collaborative inspection task in a factory area. The drones' flight paths intentionally overlap, facilitating their integration into a global map. For experimental repeatability, the drones log image data and timestamps, which are later replayed and processed on multiple offline computing platforms equipped with identically configured MPSoCs.

We compare edgeSLAM2 with the traditional edgeSLAM architecture in terms of localization accuracy and latency, as



(a) Accuracy Comparison



(b) Latency Comparison

Fig. 14. Multi-agent collaborative mapping.

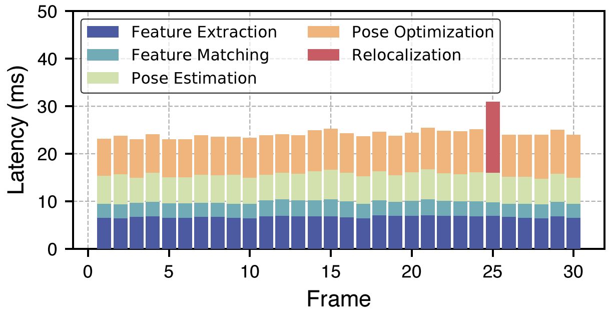


Fig. 15. Tracking latency analysis.

TABLE III
DELAY COMPARISON ACROSS DIFFERENT MODULES

System	Tracking	Local Map Update		Loop Closing
		Optimization	Synchronization	
Edge-SLAM	44.9ms	0.24s	1.35s	6.3s
edgeSLAM2	24.6ms	0.56s	-	7.1s

TABLE IV
ENERGY EFFICIENCY COMPARISON

	Zynq MPSoC	Jetson TX2	Intel i7-9700
Frame Rate (fps)	42	14	38
Power (W)	4.6	6.8	52
Energy/Frame (mJ)	108	485	1368

shown in Fig. 14. For edgeSLAM's tracking-optimization decoupled architecture, the increase in client numbers necessitates more frequent local map optimizations by the edge server, escalating computational demands and potentially delaying map updates for clients, leading to performance degradation. With the number of collaborating clients increasing from 2 to 4, there was a decline in localization accuracy by over 32% and a latency increase of about 21%.

Benefiting from the upgrade architecture that enables local map optimization directly on mobile devices, the majority of the computational load introduced by multi-agent activities does not burden the edge, allowing edgeSLAM2 to consistently achieve low localization errors ($< 15\,cm$) and maintain stable latency ($< 25\,ms$) in scenarios involving four-agent collaboration.

G. *Efficiency Study*

We analyze the latency of each component in *Tracking*. As shown in Fig. 15, during tracking, edgeSLAM2 averages 6.7 ms, 3.1 ms, 5.8 ms, and 8.5 ms for feature extraction, feature matching, pose estimation, and pose optimization, respectively. And it spends 15 ms on relocalization at Frame #25 when tracking is lost. This rapid execution of crucial steps is thanks to hardware adaptation and resource isolation (Section IV-D2).

Table III illustrates the latency of three critical modules in the SLAM system that operate at different frequencies: *Tracking* is executed for every input frame, while *Local Map Update* and *Loop Closing* are performed approximately every 2 s and 40 s, respectively. Edge-SLAM adheres to the traditional tracking-optimization decoupled architecture; thus, a complete local map update consists of two parts: map *synchronization* (1.35 s) between the mobile and edge and *optimization* (0.24 s) at the edge. For edgeSLAM2, despite our on-chip design reserving as much general-purpose computational resource as possible for local map optimization (Section IV-A2), the processing is still constrained by the low-power chip's capabilities, resulting in a 0.56 s optimization delay. However, by integrating local map optimization into the mobile client, edgeSLAM2 eliminates the map synchronization operation, thereby significantly reducing the overall local map update latency. Additionally, the loop closing process of edgeSLAM2 is slightly slower (by less than 1 s) due to the need to transmit yet-to-be-synchronized map-segments upon loop detection on the mobile device. However, the low-frequency nature of loop closing makes the impact of this additional delay almost negligible.

In terms of energy consumption, we deploy edgeSLAM2 on Zynq MPSoC, Jetson TX2, and Intel i7-9700, and analyze the energy required per frame. As indicated in Table IV, Zynq MPSoC and Jetson TX2, compared to high-performance processor i7-9700, consume less power ($< 10\,W$), making them suitable for deployment on lightweight mobile devices. Furthermore, due to the exceptional real-time performance of edgeSLAM2, when adapted to the Zynq MPSoC, it reduces the energy required per frame by 77% and 92% compared to the Jetson TX2 and i7-9700, respectively.

VI. RELATED WORK

Edge-Assisted Visual SLAM. Visual SLAM has been a fundamental area of research in robotics and mobile systems for several decades [7]. It addresses the dual tasks of mapping an unexplored environment and simultaneously tracking the location of the mobile device within it [2], [41], [42]. Recent research [43], [44] seeks to enable real-time implementation of visual SLAM on mobile devices through edge offloading. edgeSLAM [15] and Edge-SLAM [14] reallocate the resource-intensive optimization to an edge server, retaining only the lighter tracking module on the mobile client. However, this segregation of the inherently intertwined tracking and optimization tasks limits their performance in multiple aspects (Section II-B). In an alternative approach, AdaptSLAM [45] strives to execute both tracking and local mapping on the mobile client by applying an adaptive mapping strategy in response to resource constraints. Its effectiveness, however, is confined to high-end devices (e.g., Intel i7-9700 K). Diverging from these methodologies, edgeSLAM2 reshapes the edge-assisted SLAM paradigm by software and hardware co-design. By remodeling the versatile open-source SLAM system ORB-SLAM3 [2], it enables real-time visual SLAM on lightweight mobile devices.

Software-Hardware Co-Design for SLAM. The rise of hardware and software co-design has empowered the parallelizable and computationally-intensive SLAM [32], [34], [46], [47], [48], [49]. Innovations like eSLAM [34] and ac ^{2} SLAM [47] have developed FPGA-centric acceleration methods specifically for ORB feature extraction and matching. π-BA [48], on the other hand, has devised a hardware-friendly differentiation method to speed up the BA optimization. These methods, while exploring SLAM performance enhancement via dedicated hardware design, fall short of offering fully deployable systems for real-world environments. Addressing this gap, edgeSLAM2 distinguishes itself by harnessing the heterogeneous computing platform in conjunction with edge-assisted mechanisms. It systematically identifies and abstracts key bottlenecks in the SLAM algorithm and, building on the newly defined edge-assisted SLAM paradigm, introduces a comprehensive hardware and software co-design strategy (Section IV).

Multi-Agent Collaborative Mapping. Collaborative mapping typically involves multiple SLAM-capable devices exploring the environment concurrently, with their local maps merged and optimized on the central or edge server [13], [16], [17], [50], [51]. These global maps are then used to enhance the mobile devices' local maps or for downstream tasks like navigation and obstacle avoidance. Systems like C2TAM [51], CarMap [16] and CCM-SLAM [17] focus on efficiently integrating multiple sub-maps on the server, rather than supporting real-time performance or map quality on mobile devices. SwarmMap [13] goes a step further by alleviating the computational pressure on servers caused by the multi-agent scalability issues under the traditional edge-assisted real-time SLAM paradigm. In contrast, edgeSLAM2, within its upgrade edge-assisted architecture, naturally supports real-time multi-agent collaborative mapping with tailored map synchronization and compression strategies. Moving forward, it can seamlessly integrate advanced techniques for collaborative map merging, optimization, and compression, which are left as future work.

VII. Discussion

The edge-assisted paradigm is not only designed to relieve the computational burden of running SLAM on lightweight mobile devices but also to facilitate the integration of dispersed maps from different clients at edge node, which is essential for collaborative multi-agent mapping in real-world applications [13].

However, previous systems [5], [6], [14] based on the tracking-optimization decoupled architecture upload local maps solely for self-map and location optimization. As a result, these frequently uploaded maps were often redundant and lacked stability assessments, making them unsuitable for collaborative mapping. The architectural upgrades introduced by edgeSLAM2 eliminate the resource conflicts and inefficiencies caused by frequent local map synchronization. This upgrade also allows the mobile client to respond more flexibly to critical mapping events, determining the appropriate map segments to synchronize and the optimal timing for sharing.

edgeSLAM2 fully leverages the enhanced map-sharing capabilities of the upgrade architecture. Compared to the previous system [52], it designs and deploys map synchronization (Section IV-B) and compression (Section IV-C) mechanisms tailored for multi-agent collaborative mapping tasks. This update of system design is expected to lay the groundwork for future systems based on the edgeSLAM2's upgrade edge-assisted SLAM paradigm, supporting the development of multi-agent collaboration platforms.

VIII. Conclusion

We introduced the design and implementation of edgeSLAM2, an innovative edge-assisted visual SLAM system that reshapes the existing *Tracking-Optimization* decoupled paradigm by relocating the *local map optimization* module from edge to mobile. edgeSLAM2 (i) exploits the heterogeneous computing units of the Zynq UltraScale+ MPSoCs, enhancing mobile devices' computational capacity, which accommodates this architectural upgrade; and (ii) proposes a suite of technologies designed to align with this enhanced architecture via the software-hardware co-design. From this foundation, edgeSLAM2 facilitates lightweight mobile devices to execute real-time, accurate SLAM tasks, both individually and in collaboration.

Acknowledgments

The authors sincerely thank the MobiSense group and the anonymous reviewers for their insightful comments.

References

[1] R. Mur-Artal and J. D. Tardós, "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.

[2] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM," *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021.

[3] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, 2017, Art. no. 16.

[4] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct SLAM with stereo cameras," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 1935–1942.

[5] J. Xu et al., "Edge assisted mobile semantic visual SLAM," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1828–1837.

[6] 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.
[7] 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.
[8] M. R. U. Saputra, A. Markham, and N. Trigoni, “Visual SLAM and structure from motion in dynamic environments: A survey,” *ACM Comput. Surv.*, vol. 51, 2018, Art. no. 37.
[9] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers, “From monocular SLAM to autonomous drone exploration,” in *Proc. Eur. Conf. Mobile Robots*, 2017, pp. 1–8.
[10] T. Qin, P. Li, and S. Shen, “VINS-mono: A robust and versatile monocular visual-inertial state estimator,” *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.
[11] L. He, N. Aouf, J. F. Whidborne, and B. Song, “Integrated moment-based LGMD and deep reinforcement learning for UAV obstacle avoidance,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 7491–7497.
[12] L. Liu and M. Gruteser, “EdgeSharing: Edge assisted real-time localization and object sharing in urban streets,” in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[13] 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.
[14] A. J. Ben Ali, M. Kouroshli, S. Semenova, Z. S. Hashemifar, S. Y. Ko, and K. Dantu, “Edge-SLAM: Edge-assisted visual simultaneous localization and mapping,” *ACM Trans. Embedded Comput. Syst.*, vol. 22, 2022, Art. no. 18.
[15] H. Cao, J. Xu, D. Li, L. Shangguan, Y. Liu, and Z. Yang, “Edge assisted mobile semantic visual SLAM,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 12, pp. 6985–6999, Dec. 2023.
[16] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, “CarMap: Fast 3D feature map updates for automobiles,” in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 1063–1081.
[17] 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, pp. 763–781, 2019.
[18] P. Schmuck and M. Chli, “Multi-UAV collaborative monocular SLAM,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 3863–3870.
[19] FIXAR, “Fully autonomous VTOL drone for last-mile delivery,” 2023. [Online]. Available: https://fixar.pro/last-mile-delivery/
[20] wired.com, “Inside the Amazon warehouse where humans and machines become one,” 2019. [Online]. Available: https://www.wired.com/story/amazon-warehouse-robots/
[21] Heliguy, “DJI Enterprise’s complete guide to drone inspections based on best use cases,” 2021. [Online]. Available: https://enterprise-insights.dji.com/blog/complete-guide-to-drone-inspections
[22] X. Inc., “Zynq UltraScale+ MPSoC,” 2023. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[23] NVIDIA, “NVIDIA Tegra K1 series processors,” 2023. [Online]. Available: https://developer.nvidia.com/embedded/buy/tegra-k1-processor
[24] N. Pham et al., “PROS: An efficient pattern-driven compressive sensing framework for low-power biopotential-based wearables with on-chip intelligence,” in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, 2022, pp. 661–675.
[25] Z. Wang, J. Xu, X. Wang, X. Zhuge, X. He, and Z. Yang, “Industrial Knee-jerk: In-network simultaneous planning and control on a TSN switch,” in *Proc. 21st Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2023, pp. 356–369.
[26] F. Yu et al., “Brain-inspired multimodal hybrid neural network for robot place recognition,” *Sci. Robot.*, vol. 8, 2023, Art. no. eabm6996.
[27] Ardupilot, “ArduPilot Mega,” 2018. [Online]. Available: https://ardupilot.org/
[28] 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.
[29] K. Konolige and J. Bowman, “Towards lifelong visual maps,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 1156–1163.
[30] D. Van Opdenbosch, T. Aykut, N. Alt, and E. Steinbach, “Efficient map compression for collaborative visual SLAM,” in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2018, pp. 992–1000.
[31] R. Stefo, J. L. Núñez, C. Feregrino, S. Mahapatra, and S. Jones, “FPGA-based modelling unit for high speed lossless arithmetic coding,” in *Proc. 11th Int. Conf. Field-Programmable Log. Appl.*, 2001, pp. 643–647.
[32] V. Vemulapati and D. Chen, “FSLAM: An efficient and accurate slam accelerator on SoC FPGAs,” in *Proc. Int. Conf. Field-Programmable Technol.*, 2022, pp. 1–9.
[33] 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.
[34] R. Liu, J. Yang, Y. Chen, and W. Zhao, “eSLAM: An energy-efficient accelerator for real-time orb-slam on FPGA platform,” in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, Art. no. 193.
[35] T. L. Chao and K. H. Wong, “An efficient FPGA implementation of the harris corner feature detector,” in *Proc. 14th IAPR Int. Conf. Mach. Vis. Appl.*, 2015, pp. 89–93.
[36] R. J. Lipton, “Reduction: A method of proving properties of parallel programs,” *Commun. ACM*, vol. 18, pp. 717–721, 1975.
[37] B. Kisacanin, “Integral image optimizations for embedded vision applications,” in *Proc. IEEE Southwest Symp. Image Anal. Interpretation*, 2008, pp. 181–184.
[38] T. Tools, “Absolute trajectory error,” 2021. [Online]. Available: https://vision.in.tum.de/data/datasets/rgbd-dataset/tools
[39] N. Inc., “OptiTrack,” 2023. [Online]. Available: https://optitrack.com/
[40] S. Alonso, J. Lazaro, J. Jimenez, L. Muguira, and U. Bidarte, “Evaluating the OpenAMP framework in real-time embedded SoC platforms,” in *Proc. 36th Conf. Des. Circuits Integr. Syst.*, 2021, pp. 1–6.
[41] G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces,” in *Proc. 6th IEEE ACM Int. Symp. Mixed Augmented Reality*, 2007, pp. 225–234.
[42] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: Large-scale direct monocular SLAM,” in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 834–849.
[43] J. Zhang and K. B. Letaief, “Mobile edge intelligence and computing for the internet of vehicles,” *Proc. IEEE*, vol. 108, no. 2, pp. 246–261, Feb. 2020.
[44] J. Hofer, P. Sossalla, C. L. Vielhaus, J. Rischke, M. Reisslein, and F. H. Fitzek, “Comparison of analyze-then-compress methods in edge-assisted visual SLAM,” *IEEE Access*, vol. 11, pp. 68728–68743, 2023.
[45] Y. Chen, H. Inaltekin, and M. Gorlatova, “AdaptSLAM: Edge-assisted adaptive SLAM with resource constraints via uncertainty minimization,” in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.
[46] R. Eyvazpour, M. Shoaran, and G. Karimian, “Hardware implementation of SLAM algorithms: A survey on implementation approaches and platforms,” *Artif. Intell. Rev.*, vol. 56, pp. 6187–6239, 2023.
[47] C. Wang, Y. Liu, K. Zuo, J. Tong, Y. Ding, and P. Ren, “ac^2 SLAM: FPGA accelerated high-accuracy slam with heapsort and parallel keypoint extractor,” in *Proc. Int. Conf. Field-Programmable Technol.*, 2021, pp. 1–9.
[48] Q. Liu, S. Qin, B. Yu, J. Tang, and S. Liu, “-BA: Bundle adjustment hardware accelerator based on distribution of 3D-point observations,” *IEEE Trans. Comput.*, vol. 69, no. 7, pp. 1083–1095, Jul. 2020.
[49] J. Huang, G. Zhou, X. Zhou, and R. Zhang, “A new FPGA architecture of FAST and BRIEF algorithm for on-board corner detection and matching,” *Sensors*, vol. 18, 2018, Art. no. 1014.
[50] M. Karrer, P. Schmuck, and M. Chli, “CVI-SLAM—Collaborative visual-inertial SLAM,” *IEEE Trans. Robot. Autom.*, vol. 3, no. 4, pp. 2762–2769, Oct. 2018.
[51] L. Riazuelo, J. Civera, and J. M. Montiel, “C2TAM: A cloud framework for cooperative tracking and mapping,” *Robot. Auton. Syst.*, vol. 62, pp. 401–413, 2014.
[52] D. Li, Y. Zhao, J. Xu, S. Zhang, L. Shangguan, and Z. Yang, “EdgeSLAM2: Rethinking edge-assisted visual SLAM with on-chip intelligence,” in *Proc. IEEE Conf. Comput. Commun.*, 2024.

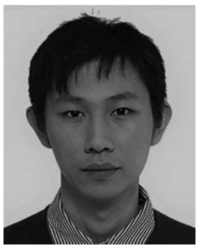
Danyang Li received the BE degree from the School of Software, Yanshan University, in 2019, and the ME degree from the School of Software, Tsinghua University, in 2022. He is currently working toward the PhD degree with the School of Software, Tsinghua University. His research interests include Internet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

Yishujie Zhao (Student Member, IEEE) received the BE and BFA degrees from Tsinghua University, in 2022. He is currently working toward the ME degree with the School of Software, Tsinghua University under the supervision of Prof. Zheng Yang.

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.

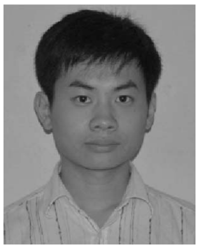
Shengkai Zhang received the MSc degree from the Huazhong University of Science and Technology, Wuhan, China, in 2012, the MPhil degree from the Hong Kong University of Science and Technology, Hong Kong, in 2014, and the PhD degree from the EIC Department, HUST, in 2021. He is currently an associate professor with the Wuhan University of Technology, Wuhan. His recent research interests include state estimation, mobile computing, multisensor fusion, and robot control and planning.

Longfei Shangguan received the BS degree from the School of Software, Xidian University, Shanghai, China, in 2011, and the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2015. He is currently a researcher with Microsoft. His research interests include wireless networks, mobile systems, and low-power communication.



Qiang Ma received the BS degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009, and the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2013. He is currently an assistant researcher with Tsinghua University. His research interests include sensor networks, mobile computing, and data privacy.

Xuan Ding (Member, IEEE) received the bachelor's degree from the School of Software, Tsinghua University, Beijing, China, in 2008, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, in 2014. He is currently a research assistant professor with the School of Software, Tsinghua University. His research interests include privacy-preserving computing, blockchain, RFID, and wireless sensing.



Zheng Yang (Fellow, IEEE) received the BE degree in computer science from Tsinghua University, in 2006, and the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2010. He is 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 and has been awarded the State Natural Science Award (second class).