edgeSLAM2: Rethinking Edge-Assisted Visual SLAM with On-Chip Intelligence

Danyang Li[1*], Yishujie Zhao[1*], Jingao Xu[1†], Shengkai Zhang[2], Longfei Shangguan[3], Zheng Yang[1]
[1]School of Software and BNRist, Tsinghua University
[2]Wuhan University of Technology, China [3]University of Pittsburgh, USA
[*]Co-primary author [†]Corresponding author
lidanyang1919@gmail.com, cobs14@126.com, xujingao13@gmail.com
shengkai@whut.edu.cn, longfei@pitt.edu, hmilyyz@gmail.com

***Abstract*—Edge-assisted visual SLAM stands as a pivotal enabler for emerging mobile applications, such as search-and-rescue, smart logistics, and industrial inspection. Limited by the computing capability of lightweight mobile devices like MAVs, current innovations balance system accuracy and efficiency by allocating lightweight and time-sensitive *tracking* tasks to mobile devices, while offloading the more resource-intensive yet delay-tolerant map *optimization* tasks to the edge. However, our pilot study in a large-scale oil field reveals several limitations of such a *tracking-optimization decoupled* paradigm, arising due to the disruption of inter-dependencies between the two tasks concerning data, resources, and threads.**

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 hierarchical and heterogeneous computing units offered by the latest 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 fully embracing the on-chip intelligence, edgeSLAM2 simultaneously enhances system accuracy and efficiency through software-hardware co-design. We deploy edgeSLAM2 on an industrial drone and conduct comprehensive experiments in a large-scale oil field over three months. The results show that edgeSLAM2 surpasses comparative methods by achieving an 80% reduction in bandwidth consumption, a 32% improvement in accuracy, and a 26% reduction in tracking delay.

I. Introduction

Visual Simultaneous Localization and Mapping (SLAM) employs video streams to simultaneously construct a 3D environmental map and estimate the camera's pose (*i.e.*, position and orientation) [1]–[3]. 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 [4]–[7], and capabilities like drone flight control, obstacle avoidance, and intelligent interaction [8]–[11].

Visual SLAM's computational intensity impedes efficient and accurate operations on lightweight devices such as Micro Aerial Vehicles (MAV) and smartphones [4], [5]. 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, including pose tracking (camera's pose estimation) and map tracking (new map points and keyframes generation). Meanwhile, the resource-intensive tasks of local and global map *optimization* are offloaded to edge servers.

TABLE I
Edge-Assisted SLAM System Comparison

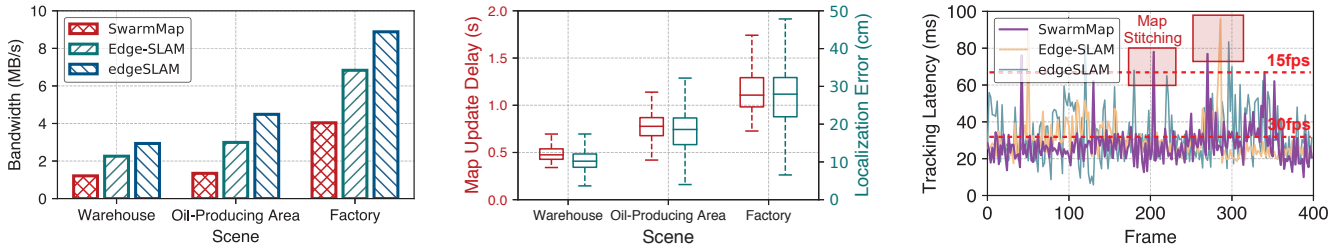
System	Bandwidth(MB/s)	Accuracy(cm)	Latency(ms)
SwarmMap [12]	1.35	13.2±5.3	32.2±6.4
Edge-SLAM [13]	2.99	17.9±5.9	34.3±8.0
edgeSLAM [14]	4.49	11.3±5.1	37.2±10.9
edgeSLAM2	**0.27**	**7.6±2.9**	**23.7±1.2**

Such a *tracking-optimization* decoupled strategy not only alleviates resource pressures on mobile devices [12]–[14] but also allows edge servers to centralize and optimize visual maps from multiple agents, enhancing collaborative efforts for tasks like cooperative scheduling [15]–[17]. This edge-assisted paradigm underpins numerous applications, *e.g.* smart logistics [18], warehouse sorting [19], and industrial inspection [20].

While existing edge-assisted SLAM systems show promise, our deployment of these systems on drones for industrial inspections within a large-scale oil field highlighted several drawbacks. 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 allows mobile agents to access timely, optimized local maps for tracking performance maintenance [12]. However, this process involves streaming large volumes of map data (*i.e.*, map points and keyframes) via wireless links, straining the network. Further, SLAM agents continually gather new map data in evolving environments, necessitating ongoing frequent map syncs. Such constant, voluminous data streaming rapidly saturate the limited and congested wireless spectrum, inducing significant map update delays that may result in tracking drift or even loss.

We validate our analysis by measuring the bandwidth usage of existing edge-assisted SLAM systems in varied industrial scenarios (§IV). As depicted in Fig. 1a, map sync data volume gradually outstrips the available network bandwidth in edgeSLAM [14] and Edge-SLAM [13] systems. While SwarmMap [12], the most recent work, designs a lighter map sync framework for reducing data volume in quasi-static environments, it faces limitations in large-scale, dynamic scenes such as factories with constant movement. As shown in



(a) Excessive bandwidth consumption (b) Considerable localization error (c) Highly dynamic tracking delay

Fig. 1. **Limitations of current practice observed in our field study.** (a) As synchronization frequency and map scale increase, each agent requires over 4MB/s bandwidth in factory scenarios, $>2\times$ that in simpler warehouse scenarios. (b) Current edge-assisted SLAM systems [13] suffer notable map update delays, leading to substantial localization errors (*i.e.*, exceeding $1.5s$ and $40cm$ in the factory). (c) Resource contention arising from map stitching, which occurs at certain frequencies, induces significant spikes in tracking delay, potentially exceeding 70ms (*i.e.*, less than 15fps).

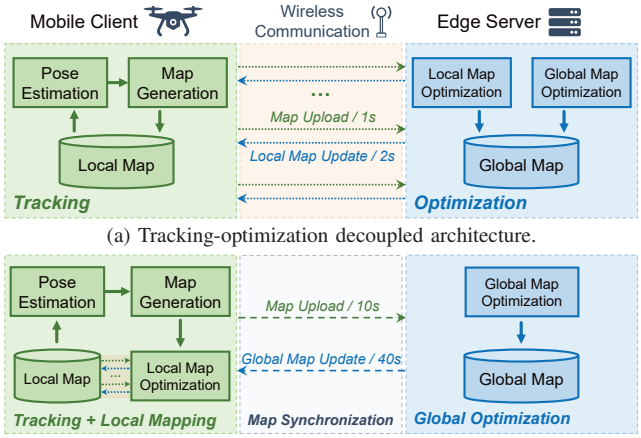
Fig. 1b, the overcrowded network, in turn, causes significant map update delays and harms tracking accuracy.

• **Map stitching disrupts tracking performance**. During map synchronization, mobile devices need to stitch received edge-optimized maps with their local ones. Typically, map stitching and pose tracking are handled in separate threads [13], [14], using locking to prevent read/write conflicts in the local map database. However, frequent map stitching and its complex mobile-edge data fusion operations (*e.g.*, perspective splicing [15], map points retrieval [5]) extend the database locking time, which results in unexpected tracking thread interruptions and further compromises the tracking performance.

We recorded the pose tracking latency for a snapshot of 400 frames ($13s$ duration). As shown in Fig. 1c, although current systems could achieve an average frame rate of around 30fps (*e.g.*, latency $<33.3ms$) without map syncs, frequent (*i.e.*, around every $2s$) resource contention induced by map stitching severely impacts the tracking performance, leaving room for improvements.

Lessons Learned. Due to the limited computing capacity of lightweight mobile devices, existing solutions with such *tracking-optimization* decoupled architecture have to position local map optimization on the edge (Fig. 2a). This approach necessitates frequent mobile-edge map syncs, resulting in (i) network-side significant bandwidth overhead and (ii) mobile-side unforeseen commandeering of thread and computational resources. To render edge-assisted visual SLAM more practical for drones in challenging and network-limited industrial environments, it's crucial to rethink the edge-assisted architecture – by integrating *tracking* and *local map optimization* on mobile (Fig. 2b), 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 [21], NVIDIA Tegra [22]) 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 [23]–[25].



(a) Tracking-optimization decoupled architecture.

(b) Upgraded edge-assisted architecture.

Fig. 2. **Edge-assisted SLAM architecture comparison.** (a) The tracking-optimization decoupled architecture necessitates frequent data synchronization (*i.e.*, map upload every $1s$, and local map update every $2s$ [14]). (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 $10s$, update every $40s$ on average).

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 hierarchical and heterogeneous computing units offered by the latest commercial Zynq SoCs to enhance the computational capacity of mobile devices, and on this basis, accommodates such architectural upgrade through software-hardware co-design. Evaluation results summarized in Table I demonstrate edgeSLAM2's superior performance compared to current practice.

Transforming edge-assisted visual SLAM architecture to boost edgeSLAM2's accuracy, efficiency, and reduce network overhead, poses many challenges that are addressed in this work: (i) how to re-assign visual SLAM's functional modules between mobile and edge to reshape the architecture; (ii) how to re-design an effective map sync framework to adapt to the new architecture; and (iii) how to push forward tracking performance on mobile through software-hardware co-design. 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 (§II).

• On the Algorithm front, we propose a new mobile-edge map synchronization solution compatible with the upgraded architecture. We first propose *Event-Responsive Map Synchronization*, optimizing the timing and frequency of map synchronization under the new paradigm (§III-B). Then, we introduce *Observation Consistency based Map Streamlining*, minimizing the transmission payload by selectively compressing the necessary map elements (§III-C).

• On the Implementation front, we implement edgeSLAM2 on the latest Zynq UltraScale+ MPSoC platform [21] through software-hardware co-design. We fully utilize heterogeneous arithmetic units to enable mobile tasks to run in real time (§III). Particularly, we propose a *Delay Deterministic Tracking* approach, leveraging FPGA and resource isolation strategy to accelerate some critical modules (*e.g.*, feature extraction and matching) in tracking and prevent its thread from being interrupted, to alleviate the tracking delay bottleneck (§III-D).

We deploy edgeSLAM2 on a drone testbed. Comprehensive experiments are carried out in a large-scale oil field over three months, covering a variety of scenarios including warehouses, oil-producing areas, and factories, collecting 188 trajectories with 182,267 frames. We compare edgeSLAM2 with three state-of-the-art (SOTA) edge-assisted SLAM systems, SwarmMap [12], edgeSLAM [14], and Edge-SLAM [13]. Evaluation results show that edgeSLAM2 achieves an average bandwidth consumption of 0.27MB/s, a localization accuracy of 7.6cm, and a tracking delay of 23.7ms. 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.

In summary, this paper makes the following contributions.
(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 propose several technologies, spanning event-responsive map synchronization, observation consistency based map streamlining, and delay deterministic tracking, in edgeSLAM2 through software and hardware co-design, to enable mobile devices to obtain accurate pose in real-time with minimal bandwidth consumption.
(3) We fully implement edgeSLAM2 and deploy it on an industrial inspection drone. Our three-month pilot study in a large-scale oil field demonstrates that edgeSLAM2 makes a great process towards fortifying edge-assisted visual SLAM into a fully practical system for wide deployment.

II. System Overview

We first briefly introduce the existing tracking-optimization decoupled architecture. Then, we detail the upgraded architecture adopted by edgeSLAM2.

A. Existing Edge-Assisted Visual SLAM

An edge-assisted visual SLAM system can be abstracted as mobile, edge, and network, three layers [12]. The tasks allocation among them can be summarized as below.

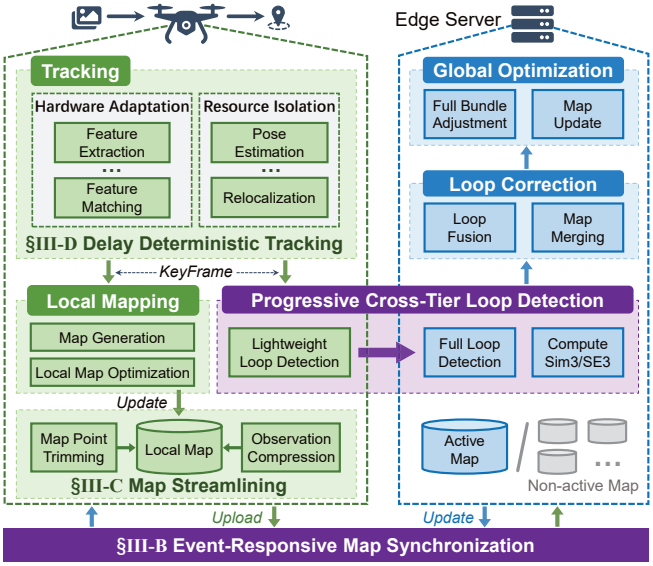


Fig. 3. edgeSLAM2 Overview

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 map optimization* and *global map optimization*. Specifically, local Bundle Adjustment (*local BA* [26]) 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 (from *map tracking*) 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. edgeSLAM2 Overview

edgeSLAM2 upgrades the tracking-optimization decoupled architecture (Fig. 2a) by transferring the *local map optimization* from edge to mobile. Such consolidation maintains data, resource, and thread dependencies between *tracking* and *local map optimization* as shown in Fig. 2b, resulting in reduced bandwidth overhead for map sync and improved pose tracking performance. 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

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

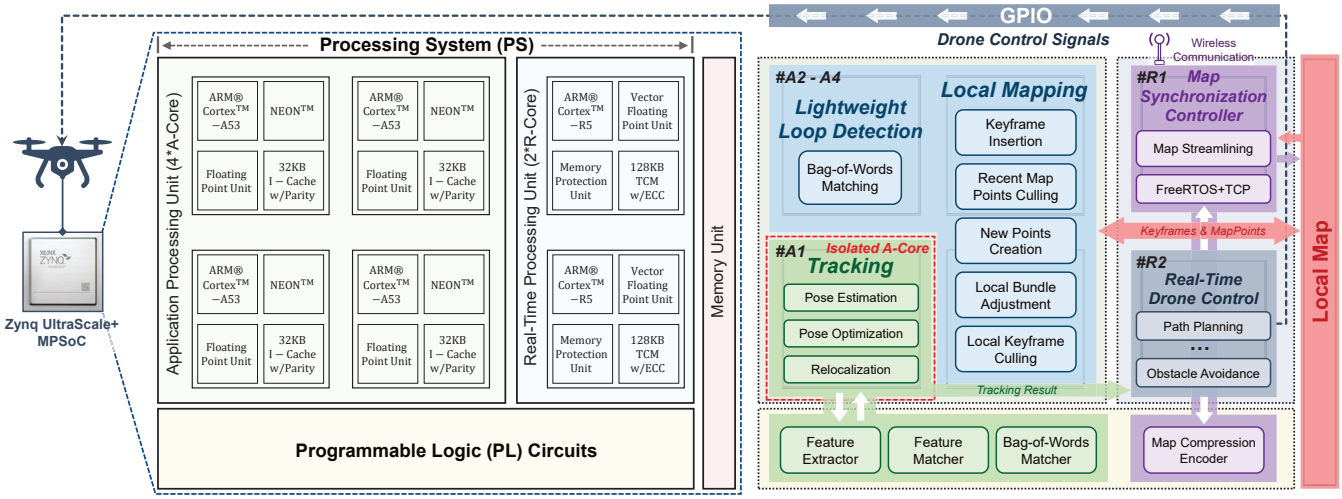


Fig. 4. An exhibition of the Zynq UltraScale+ MPSoC (left), and the edgeSLAM2's on-chip architecture (right).

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 (§III-A). It integrates FPGA-adapted essential algorithms and resource isolation for *delay deterministic tracking* (§III-D), while retaining most generic computing resources for *local map optimization*. Additionally, an initial loop recognition is conducted by a *lightweight loop detection* module at the mobile side.

•**On the Edge layer,** upon receipt of a loop detection signal from the mobile client, edgeSLAM2 performs thorough loop verification, followed by triggering the resource-intensive tasks of *loop correction* and *global optimization*.

•**On the Network layer,** edgeSLAM2 refrains from frequent uploads of newly generated maps. Instead, it implements an *event-responsive map synchronization* strategy (§III-B) to refine the timing and frequency of synchronization. To further enhance efficiency, an *observation consistency based map streamlining* approach (§III-C) is employed for effective map compression before synchronization.

III. Software-Hardware Co-Design of edgeSLAM2

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 [21], 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.

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 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,

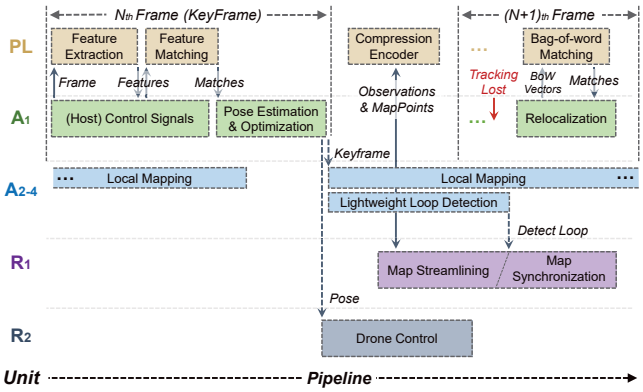


Fig. 5. A typical on-chip workflow of edgeSLAM2.

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.

Architecture. The on-chip architecture of edgeSLAM2, as depicted in Fig. 4, is deftly integrated with the computational capabilities of MPSoC. Firstly, *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 (§III-C), and executing map transmission based on the specified synchronization strategy (§III-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).

Workflow. The workflow of edgeSLAM2, demonstrated in

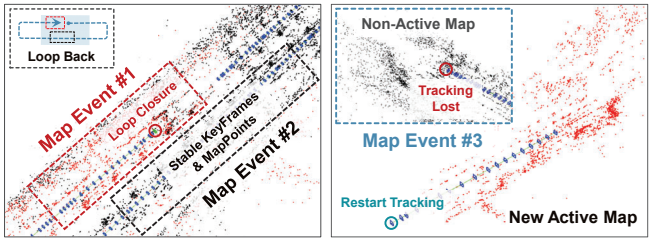


Fig. 6. **Event-Responsive Map Synchronization.** In the presented cases, Events #1 and #2 are triggered in a loop scenario, whereas Event #3 is prompted by *Tracking Lost* due to rapid perspective shifts. Red dots denote active map points observed by the current frame and co-visible keyframes, while black dots denote optimized, stable map points no longer in observation.

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 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 local maps; (ii) facilitate the swift sharing of locally constructed maps with other clients via centralized edge server, promoting cooperative tasks; and (iii) minimize redundant data transfers, maximizing synchronization efficiency. Motivated by these objectives, we react to three typical map events for synchronization, as shown in Fig. 6.

Event #1: Progressive Cross-Tier Loop Detection. We employ a lightweight loop detection module on the mobile client to preliminarily identify potential loops. This module uses Bag-of-Words vector matching for initial keyframe comparison, pinpointing similar keyframes that may indicate a loop. The subsequent resource-intensive tasks, such as full loop detection, loop correction, and global map optimization, are offloaded to the edge server. Upon detecting a loop, the client uploads any unsynchronized map (Event #1). The edge server then confirms the loop. The subsequent actions depend on the type of loop closure: for an intra-map loop closure (*i.e.*, revisiting a location within a single map), loop correction and global optimization are promptly initiated. On the other hand, for an inter-map loop closure (*i.e.*, detecting overlap between two maps), the maps are first merged before global optimization. Once the optimization is complete, the updated map is synced back to the mobile, replacing the local map.

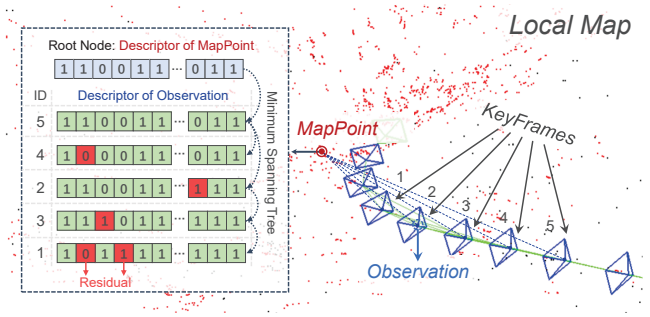


Fig. 7. Illustration of observation consistency based map streamlining.

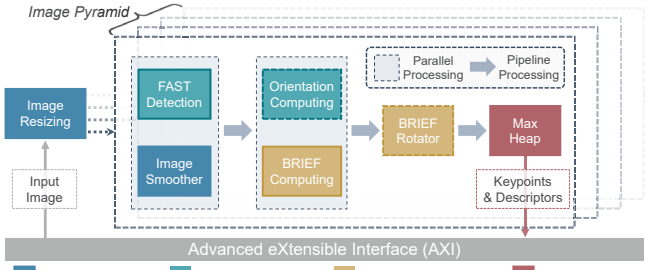
Event #2: Submap Stabilization. In the absence of other map events, we periodically upload stable map sections for sharing with other clients. In SLAM, active map elements like the current frame, its co-visible keyframes, and their observed map points, are subject to changes during map optimization. To identify stable sections, we employ a Least Recently Used (LRU) strategy [27]. We maintain a queue of keyframes, shifting any keyframe to the queue's front when it's modified during optimization. Periodically, we shift the "cooled down" keyframes from the rear of the queue to the stable submap. We upload such a submap when it contains more than 20 keyframes (Event #2). This approach prevents the redundant uploading of map elements that are prone to frequent changes.

Event #3: New Active Map Generation. When tracking is lost and relocalization fails, the system restarts tracking and activates a new map to support subsequent tracking. Concurrently, the original local map switches to a non-active state. As this non-active map no longer participates in local map optimization, we upload any unsynchronized map elements it contains (Event #3). At the edge server, the non-active maps are preserved until an inter-map loop closure event occurs, at which point they are reintegrated through map fusion.

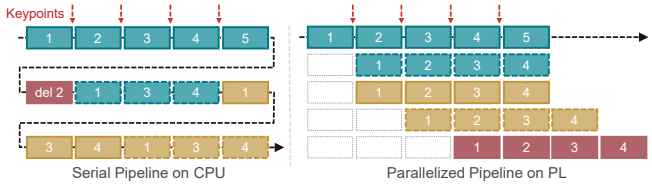
C. Observation Consistency based Map Streamlining

In visual SLAM, each map point is typically observed across multiple keyframes, with each observation represented by a feature descriptor (*i.e.*, binary string). These corresponding observations, when matched and triangulated [28], yield the associated map point, as illustrated in Fig. 7. By utilizing the observation similarity from different keyframes for the same map point, we (i) compress the feature descriptors, which constitute the largest proportion of storage, and (ii) selectively retain the most informative map points.

Observation Compression Coding. Upon identifying a submap for upload, the compression process begins with the removal of features unassociated with any map point, as these are generally irrelevant for map optimization or cooperative mapping. Next, we apply compression encoding to the remaining observation descriptors. Similar to [29], we create a minimum spanning tree where each observation descriptor serves as a node, and the edges between them are weighted according to their minimum Hamming distances. In this tree, we designate the map point as the root node, as illustrated in Fig. 7. We traverse the spanning tree iteratively, calculate



(a) The architecture of the feature extractor.



(b) Workflow comparison of feature extraction.

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.

the residuals between neighboring descriptors, and employ arithmetic coding to compress these residuals.

The effectiveness of this compression technique is based on two factors. First, the minimal residuals 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 (§III-B) facilitates the upload of sufficiently large maps, 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 [30].

Map Point Trimming. We further utilize observation consistency to selectively preserve high-quality map points, ensuring a balance between map quality and compression efficiency. 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 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. We prioritize maintaining map points with smaller W_p, *i.e.*, those with more observations and robust observation consistency. These map points are typically more stable and exhibit higher compressibility. Additionally, following the map point trimming, if a certain keyframe observes fewer than 40 map points, we gradually restore the associated map points based on their weights until every keyframe has at least 40 effective observations, ensuring the localizability of keyframes.

D. Delay Deterministic Tracking

In the critical *Tracking* module in edgeSLAM2, we leverage the heterogeneous computational resources on-chip to

Algorithm 1: Adaptation of Matching Routine on PL

Input: Feature Descriptors: $\mathcal{D}_f = \{D_f\}, \mathcal{D}_m = \{D_m\}$
Output: Matching Results: $\mathcal{M}$

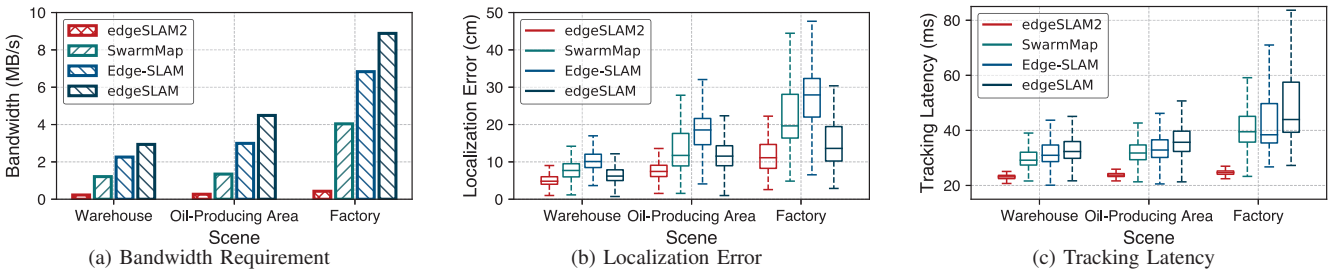
1 **for** each D_f in $\mathcal{D}_f$ **do** `// k-Way Loop Unroll`
2 **initialize** $\mathcal{M}_{\text{local}} = \emptyset$, $dis_{\text{min}} = +\infty$;
3 **for** each D_m in $\mathcal{D}_m$ **do**
4 $dis = \texttt{Hamming}(D_f, D_m)$; `// Parallel XOR ports + Adder tree`
5 **if** $dis < dis_{\text{min}}$ **then**
6 $dis_{\text{min}} = dis$; $M_f = D_m$;
7 **end**
8 **end**
9 **if** $dis_{\text{min}} < \tau_{threshold}$ **then**
10 $\mathcal{M}_{\text{local}}$.add($[D_f, M_f]$);
11 **end**
12 **end**
13 $\mathcal{M} = \texttt{ParallelReduction}(\{\mathcal{M}_{\text{local}}\})$;

(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.

Hardware Adaptation of Tracking. In the tracking process, feature extraction and matching constitute the computational bottlenecks [31]. Specifically, during feature extraction, ORB features [32] 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.

We restructure the ORB feature extraction algorithm to align with hardware processing capabilities. As shown in Fig. 8a, we downsample the input image and conduct parallel processing on the generated 4-layer pyramid. For each layer, we first **detect** keypoints within the image, then **calculate** the descriptors of these keypoints, and finally **filter** the optimal feature points based on Harris scores [33] 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. 8b. This efficient pipeline significantly reduces hardware idle periods, thereby optimizing feature extraction latency.

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



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

Fig. 9. Overall Performance Comparison

matching threshold (Line 8–11). At Line 13, we use parallel reduction [34] 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.

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 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 [35]. 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.

IV. Evaluation

A. Experimental Methodology

Field Studies. We incorporate edgeSLAM2 into the ArduPilot APM flight controller and deploy it on an AMOVLAB P450-NX drone. We conduct a field study spanning three months, delivering real-time localization services for industrial inspection tasks in oil fields. 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 drone communicates with the edge node via 2.4 GHz WiFi in indoor environments, while in outdoor settings, it switches to a mesh network[2]. The edge side of edgeSLAM2 is deployed on an Nvidia Jetson AGX Xavier edge node, with its power consumption capped at 30W, within the range of available power supply in industrial settings.

[2]In our measurements, the maximum throughput in the outdoor mesh and indoor WiFi networks is 14.3MB/s and 26.8MB/s, respectively.

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

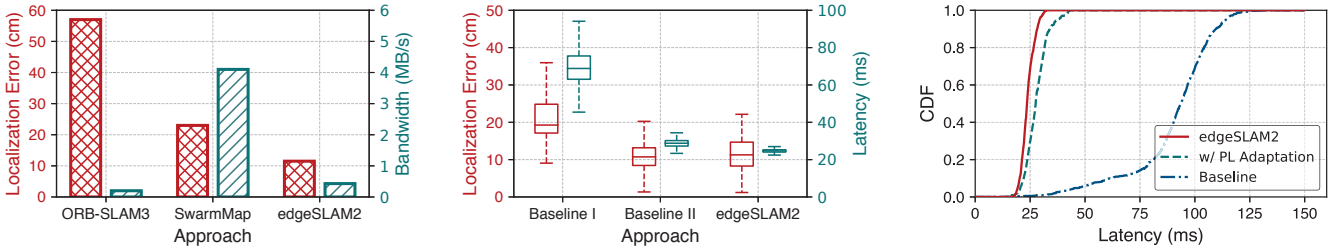
Metrics and Ground Truth. To evaluate system overhead, we measure the *bandwidth requirement* (in MB/s), defined as the average volume of data transferred per second. We assess the real-time performance by recording the *tracking latency*, denoting the time from image receipt to pose output. The localization accuracy is determined using the *Absolute Trajectory Error* (ATE, in cm), a gold standard in SLAM algorithm evaluation [36]. The ground truth for indoor localization is obtained through Opti-Track [37], whereas Real-Time Kinematic (RTK) is utilized for outdoor environments.

Baselines. We compare edgeSLAM2 with three SOTA edge-assisted SLAM systems, SwarmMap [12], Edge-SLAM [13], and edgeSLAM [14]. 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 [38]. 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. 9a, edgeSLAM2 requires an average bandwidth of 0.23MB/s, 0.27MB/s, and 0.43MB/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. 9b depicts the localization performance of edgeSLAM2 and comparative systems in different settings. The average localization error of edgeSLAM2 is 5.3cm, 7.6cm, and 11.5cm 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



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

Fig. 10. Ablation Study

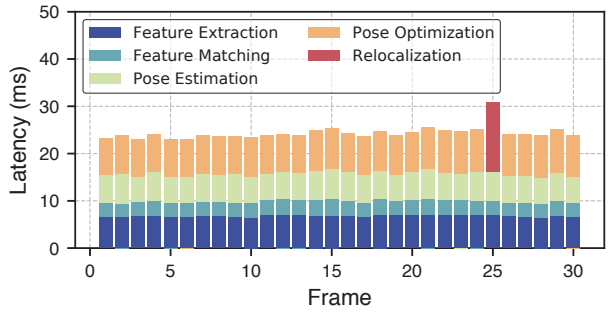


Fig. 11. Tracking Latency Analysis

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. 9c, edgeSLAM2 achieved an average latency of 23.7ms and 24.6ms 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 24.4ms, 25.2ms, and 26.0ms, 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. 10a, 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,

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

compared to SwarmMap. These results underscore the pivotal role of the upgraded edge-assisted architecture in edgeSLAM2.

2) Effectiveness of Implementation: We assess the effectiveness of our implementation by deploying edgeSLAM2 on Jetson TX2 and Intel i7-9700 processors, denoted as Baseline I and Baseline II, respectively. As shown in Fig. 10b, 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 0.95cm due to local map optimization delay.

We further evaluate the implementation of *Tracking* in edgeSLAM2 (§III-D). As shown in Fig. 10c, 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 8.5ms. The above results manifest the effectiveness of the software-hardware co-design paradigm implemented on Zynq MPSoC.

D. Efficiency Study

We analyze the latency of each component in *Tracking*. As shown in Fig. 11, during tracking, edgeSLAM2 averages 6.7ms, 3.1ms, 5.8ms, and 8.5ms for feature extraction, feature matching, pose estimation, and pose optimization, respectively. And it spends 15ms on relocalization at Frame #25 when tracking is lost. This rapid execution of crucial steps is thanks to hardware adaptation and resource isolation (§III-D).

Table III exhibits the end-to-end latency of three critical modules: *Tracking*, *Local Map Update*, and *Loop Closing*. Although edgeSLAM2's optimization latency is higher than Edge-SLAM due to its deployment on lightweight devices,

Edge-SLAM takes a significant $1.35s$ for map synchronization, including on-network map transmission and on-mobile map reconstruction. In addition, the loop closing process of edgeSLAM2 is slightly slower (by less than $1s$) due to the need for transmitting yet-to-be-synchronized submaps upon loop detection on the mobile client.

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 ($<10W$), 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.

V. Related Work

Visual SLAM. Visual SLAM remains a cornerstone of robotics and mobile systems research, with its roots extending back several decades [6]. It tackles the dual tasks of mapping an unexplored environment and tracking the mobile device within it simultaneously. The most common sensors employed are monocular cameras [26], stereo cameras [39], and RGB-D cameras [40]. Some of the well-known visual SLAM systems include PTAM [41], LSD-SLAM [42], and ORB-SLAM [1], [2], [26], among which, ORB-SLAM3 [2] stands out as a versatile open-source system, covering monocular, stereo, and visual-inertial solutions. Although edgeSLAM2 is implemented based on the monocular version of ORB-SLAM3, our universal optimization of key modules for on-chip implementation allows seamless adaptation to other vision-centric multi-sensor SLAM approaches.

Edge-assisted real-time SLAM. Recent research [12]–[14], [43]–[45] seeks to enable real-time implementation of visual SLAM on mobile devices through edge offloading. edgeSLAM [14] and Edge-SLAM [13] reallocate the resource-intensive optimization to an edge server, retaining only the lighter tracking module on the mobile client. SwarmMap [12] builds upon this concept and introduces map backbone profiling and synchronization strategies for effective multi-agent operations. However, this segregation of the inherently intertwined tracking and optimization tasks limits their performance in multiple aspects (§I). 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 i7-9700K). Diverging from these methodologies, edgeSLAM2 reshapes the edge-assisted SLAM paradigm by hardware-software co-design, facilitating real-time, accurate visual SLAM on lightweight mobile devices, even in network-constrained conditions.

Software-hardware co-design for SLAM. The rise of hardware and software co-design has empowered the parallelizable and computationally-intensive SLAM [31], [46]–[50]. Innovations like eSLAM [47] and ac^{2}SLAM [48] have developed FPGA-centric acceleration methods specifically for ORB feature extraction and matching. π-BA [49], 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 (§III).

VI. Conclusion

We have presented the design and implementation of edgeSLAM2, an innovative edge-assisted visual SLAM system that reshapes the existing *Tracking-Optimization* decoupled edge-assisted paradigm by transferring the *local map optimization* module from edge to mobile. edgeSLAM2 (i) exploits the advanced, hierarchical computing units of the latest Zynq SoCs, enhancing mobile devices' computational capacity, which accommodates this architectural upgrade; and (ii) proposes several technologies to be compatible with the upgraded architecture through the software-hardware co-design. On this basis, edgeSLAM2 enables mobile devices to achieve real-time, accurate localization with minimized bandwidth consumption. Extensive evaluation in real-world environments across 3 months demonstrates its superior performance.

Acknowledgment

We sincerely thank the MobiSense group and the anonymous reviewers for their insightful comments. This work is supported in part by the National Key Research Plan under grant No. 2021YFB2900100, the NSFC under grant No. 62372265, No. 62302254. No. 62332016, No. 62202262, and No. 62272462.

References

[1] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, 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 Transactions on Robotics*, 2021.

[3] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, 2017.

[4] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *IEEE INFOCOM 2020-IEEE Conference on computer communications*, 2020.

[5] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, 2016.

[7] M. R. U. Saputra, A. Markham, and N. Trigoni, "Visual slam and structure from motion in dynamic environments: A survey," *ACM Computing Surveys (CSUR)*, 2018.

[8] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers, "From monocular slam to autonomous drone exploration," in *2017 European Conference on Mobile Robots (ECMR)*, 2017.

[9] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, 2018.

[10] L. He, N. Aouf, J. F. Whidborne, and B. Song, "Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[11] L. Liu and M. Gruteser, "Edgesharing: Edge assisted real-time localization and object sharing in urban streets," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021.

[12] J. Xu, H. Cao, Z. Yang, L. Shangguan, J. Zhang, X. He, and Y. Liu, "{SwarmMap}: Scaling up real-time collaborative visual {SLAM} at the edge," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[13] 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 Transactions on Embedded Computing Systems*, 2022.

[14] H. Cao, J. Xu, D. Li, L. Shangguan, Y. Liu, and Z. Yang, "Edge assisted mobile semantic visual slam," *IEEE Transactions on Mobile Computing*, 2022.

[15] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "{CarMap}: Fast 3d feature map updates for automobiles," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[16] P. Schmuck and M. Chli, "Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *Journal of Field Robotics*, 2019.

[17] ——, "Multi-uav collaborative monocular slam," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[18] FIXAR, "Fully Autonomous VTOL Drone for Last-Mile Delivery," https://fixar.pro/last-mile-delivery/, 2023.

[19] wired.com, "Inside the Amazon Warehouse Where Humans and Machines Become One," https://www.wired.com/story/amazon-warehouse-robots/, 2019.

[20] Heliguy, "DJI Enterprise's Complete Guide to Drone Inspections Based on Best Use Cases," https://enterprise-insights.dji.com/blog/complete-guide-to-drone-inspections, 2021.

[21] X. Inc., "Zynq UltraScale+ MPSoC," https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html, 2023.

[22] NVIDIA, "NVIDIA Tegra K1 Series Processors," https://developer.nvidia.com/embedded/buy/tegra-k1-processor, 2023.

[23] N. Pham, H. Jia, M. Tran, T. Dinh, N. Bui, Y. Kwon, D. Ma, P. Nguyen, C. Mascolo, and T. Vu, "Pros: an efficient pattern-driven compressive sensing framework for low-power biopotential-based wearables with on-chip intelligence," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022.

[24] 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 *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 2023.

[25] F. Yu, Y. Wu, S. Ma, M. Xu, H. Li, H. Qu, C. Song, T. Wang, R. Zhao, and L. Shi, "Brain-inspired multimodal hybrid neural network for robot place recognition," *Science Robotics*, 2023.

[26] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, 2015.

[27] K. Konolige and J. Bowman, "Towards lifelong visual maps," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[28] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, 1997.

[29] D. Van Opdenbosch, T. Aykut, N. Alt, and E. Steinbach, "Efficient map compression for collaborative visual slam," in *2018 IEEE winter conference on applications of computer vision (WACV)*, 2018.

[30] R. Stefo, J. L. Núñez, C. Feregrino, S. Mahapatra, and S. Jones, "Fpga-based modelling unit for high speed lossless arithmetic coding," in *Field-Programmable Logic and Applications: 11th International Conference, FPL 2001 Belfast, Northern Ireland, UK, August 27-29, 2001 Proceedings 11*, 2001.

[31] V. Vemulapati and D. Chen, "Fslam: an efficient and accurate slam accelerator on soc fpgas," in *2022 International Conference on Field-Programmable Technology (ICFPT)*, 2022.

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

[33] T. L. Chao and K. H. Wong, "An efficient fpga implementation of the harris corner feature detector," in *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, 2015.

[34] R. J. Lipton, "Reduction: A method of proving properties of parallel programs," *Communications of the ACM*, 1975.

[35] B. Kisacanin, "Integral image optimizations for embedded vision applications," in *2008 IEEE Southwest Symposium on Image Analysis and Interpretation*, 2008.

[36] T. tools, "Absolute Trajectory Error," https://vision.in.tum.de/data/datasets/rgbd-dataset/tools, 2021.

[37] N. Inc., "OptiTrack," https://optitrack.com/, 2023.

[38] S. Alonso, J. Lazaro, J. Jimenez, L. Muguira, and U. Bidarte, "Evaluating the openamp framework in real-time embedded soc platforms," in *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*, 2021.

[39] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2015.

[40] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, 2011.

[41] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, 2007.

[42] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, 2014.

[43] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the internet of vehicles," *Proceedings of the IEEE*, 2019.

[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*, 2023.

[45] Y. Chen, H. Inaltekin, and M. Gorlatova, "Adaptslam: Edge-assisted adaptive slam with resource constraints via uncertainty minimization," *IEEE INFOCOM 2023-IEEE Conference on computer communications*, 2023.

[46] R. Eyvazpour, M. Shoaran, and G. Karimian, "Hardware implementation of slam algorithms: a survey on implementation approaches and platforms," *Artificial Intelligence Review*, 2022.

[47] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019.

[48] 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 *2021 International Conference on Field-Programmable Technology (ICFPT)*, 2021.

[49] Q. Liu, S. Qin, B. Yu, J. Tang, and S. Liu, "π-ba: Bundle adjustment hardware accelerator based on distribution of 3d-point observations," *IEEE Transactions on Computers*, 2020.

[50] 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*, 2018.