TrinitySLAM: On-board Real-time Event-image Fusion SLAM System for Drones

XINJUN CAI, Tsinghua University, Beijing, China
JINGAO XU, School of softwore, Tsinghua University, Beijing, China
KUNTIAN DENG, Tsinghua University, Beijing, China
HONGBO LAN, Tsinghua University, Beijing, China
YUE WU, Tsinghua University, Beijing, China
XIANGWEN ZHUGE, Tsinghua University, Beijing, China
ZHENG YANG, School of Software, Tsinghua University, Beijing, China

Drones have witnessed extensive popularity among diverse smart applications, and visual Simultaneous Localization and Mapping (SLAM) technology is commonly used to estimate the six-degrees-of-freedom pose for drone flight control systems. However, traditional image-based SLAM cannot ensure the flight safety of drones, especially in challenging environments such as high-speed flight and high dynamic range scenarios. The event camera, a new vision sensor, holds the potential to enable drones to overcome these challenging scenarios if fused with the image-based SLAM. Unfortunately, the computational demands of event-image fusion SLAM have grown manifold compared with image-based SLAM. Existing research on visual SLAM acceleration cannot achieve real-time operation of event-image fusion SLAM on on-board computing platforms for drones. To fill this gap, we present *TrinitySLAM*, a high-accuracy, real-time, low-energy consumption event-image fusion SLAM acceleration framework utilizing Xilinx Zynq, an on-board heterogeneous computing platform. The key innovations of *TrinitySLAM* include a fine-grained computation allocation strategy, several novel hardware–software co-acceleration designs, and an efficient data exchange mechanism. We fully implement *TrinitySLAM* on the latest Zynq UltraScale+ platform and evaluate its performance on one custom-made drone dataset and four official datasets covering various scenarios. Comprehensive experiments show that *TrinitySLAM* improves the pose estimation accuracy by 28% with half end-to-end latency and 1.2× energy consumption reduction compared with the most comparable state-of-the-art heterogeneous computing platform acceleration baseline.

CCS Concepts: • **Human-centered computing** → *Ubiquitous and mobile computing*; Ubiquitous and mobile devices; Mobile devices;

Additional Key Words and Phrases: Drone, pose estimation, event camera, SLAM, software–hardware co-design

Authors' Contact Information: Xinjun Cai, Tsinghua University, Beijing, Beijing, China; e-mail: cxj18@mails.tsinghua.edu.cn; Jingao Xu, School of softwore, Tsinghua University, Beijing, China; e-mail: xujingao13@gmail.com; Kuntian Deng, Tsinghua University, Beijing, Beijing, China; e-mail: dkt21@mails.tsinghua.edu.cn; Hongbo Lan, Tsinghua University, Beijing, Beijing, China; e-mail: lhb18@mails.tsinghua.edu.cn; Yue Wu, Tsinghua University, Beijing, Beijing, China; e-mail: ywu92@bjfu.edu.cn; Xiangwen ZhuGe, Tsinghua University, Beijing, Beijing, China; e-mail: zgxw18@gmail.com; Zheng Yang (Corresponding author), School of Software, Tsinghua University, Beijing, China; e-mail: hmilyyz@gmail.com.

ACM 1550-4859/2024/11-ART121
https://doi.org/10.1145/3696420

ACM Reference Format:
Xinjun Cai, Jingao Xu, Kuntian Deng, Hongbo Lan, Yue Wu, Xiangwen ZhuGe, and Zheng Yang. 2024. TrinitySLAM: On-board Real-time Event-image Fusion SLAM System for Drones. *ACM Trans. Sensor Netw.* 20, 6, Article 121 (November 2024), 22 pages. https://doi.org/10.1145/3696420

1 Introduction

With the rapid development of smart cities, commercial drones have witnessed extensive popularity among diverse smart applications, including smart logistics, industrial inspection, and emergency rescue due to their flexible flight control systems [27, 43]. **Six-degrees-of-freedom (6-DoF)** pose estimation is one of the key parts of drone flight control systems for these applications that allow drones to accomplish complex tasks such as navigation, localization, mapping, and obstacle avoidance. Vision-based methods — in particular, **visual simultaneous localization and mapping (visual SLAM) technology** —has become one of the most attractive solutions because of the abundance of environmental information provided by visual sensors [29, 34].

Given that drones usually fly with uneven velocity across complex environments to perform tasks in practice, traditional image-based SLAM (using a standard camera as the sole visual sensor) easily suffers from low image quality caused by motion blur, poor illumination, or HDR scenarios, in which the performance of image feature matching decreases significantly and further decreases the accuracy of pose estimation. Recently, a kind of bio-inspired sensor called an *event camera* has become one of the promising visual sensors in visual SLAM because it samples light based on the scene brightness changes and can offer several advantages such as very high temporal resolution (around 1 *MHz*) and very high dynamic range (140 dB compared with 60 dB of standard cameras). As illustrated in Figure 1, event cameras do well in sensing motion with very low latency and measure changes of brightness but without absolute brightness value, whereas standard cameras provide absolute brightness value for each pixel but are susceptible to motion blur. Therefore, fusing the event camera into image-based SLAM (we call it **event-image fusion SLAM**) has great potential for enabling drones to fly in the aforementioned challenging scenarios. Research on event-image fusion SLAM has subsequently surfaced [7, 42].

Obviously, high-accuracy and real-time SLAM technology to achieve a drone's 6-DoF pose estimation is always crucial for drone flight safety in practice. Many studies have investigated how to run image-based SLAM systems in real time on drone on-board computing platforms. The authors of [2, 36, 45] propose offloading part of the workload to the edge server. That approach relies on network conditions that are unsuitable for drones flying in harsh outdoor environments. While some works employ graphics processing units (GPUs) [1, 9] or utilize **field-programmable gate arrays (FPGAs)** [3, 8, 10] to speed up the computation-intensive SLAM, they still fall short of real-time performance (e.g., 30 fps). Furthermore, the computational demands of event-image fusion SLAM have increased manifold compared with the original image-based SLAM, primarily due to the added pipeline of processing a vast amount of event data. Thus, existing works are even more incapable of achieving real-time event-image fusion SLAM on on-board platforms. Obviously, achieving high-accuracy and low-latency event-image fusion SLAM on drone on-board computing platforms is non-trivial.

Recently, two new opportunities have arisen, prompting us to explore the design of a high-accuracy and real-time event-image fusion SLAM system on the drone on-board computing platform:

(1) The emerging study of on-chip intelligence [11, 38, 40] is powerful for accomplishing computation-intensive tasks locally by utilizing heterogeneous on-board computing

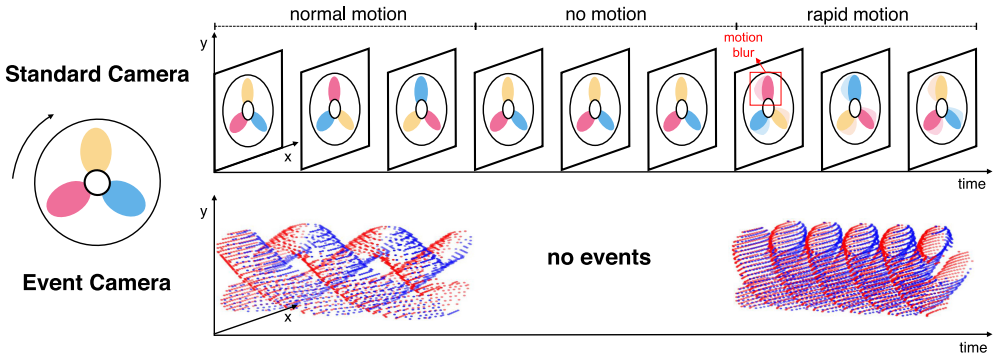


Fig. 1. Illustration of the distinct characteristics of standard cameras and event cameras.Figure referred from [19]. While standard cameras can capture RGB values, they are susceptible to motion blur. On the other hand, event cameras can detect motion and changes in brightness, but they cannot capture absolute brightness values.

platforms (e.g., Xilinx Zynq [18] and NVIDIA Tegra [31]), which not only possess on-chip FPGAs but also equip general-purpose central processing units (CPUs).

(2) The high-speed parallel processing capabilities of on-chip FPGAs align well with the high sampling frequency characteristics of event data, offering the potential for rapid event data processing.

Albeit inspiring, realizing the above intuitions in the design of the high-accuracy and real-time event-image fusion SLAM upon on-board computing platforms is non-trivial and faces the following three key challenges.

— **Complex choices in allocating hierarchical tasks:** The event-image fusion SLAM algorithm comprises hierarchical task modules (details in Section 2.1), and on-board computing platforms similarly have tiered computational units (details in Section 3.1). Partitioning the SLAM algorithm into sub-modules and allocating them to different computational units, however, are also non-trivial as both SLAM functional units are tightly coupled and the characteristics of computational units vary. An improper allocation may result in redundant data exchange and inefficient utilization of computational resources that, in turn, increase the algorithm latency.

— **Mining computational power from heterogeneous units:** Although the hierarchical computational resources of the on-board computing platform empower lightweight devices to perform some complex tasks, simply running the decoupled sub-module on heterogeneous computational units without fully mining their computational power cannot meet the requirements for real-time performance (Section 2.2).

— **Arduous collaboration between heterogeneous units:** Intermediate data interaction between heterogeneous computational units comes at a relatively high cost. Facilitating efficient collaboration among these units to minimize data exchanges is essential for reducing system latency.

In this article, we present the design and implementation of *TrinitySLAM*, the first **real-time**, **high-accuracy** and **low-energy consumption** event-image fusion SLAM system on a commercial heterogeneous on-board platform following a software and hardware co-design paradigm. *TrinitySLAM* leverages the state-of-the-art event-image fusion SLAM algorithm, UltimateSLAM [42], as the foundational algorithm and further improves its accuracy and speeds it up by

meticulous task allocation strategies, efficient computation task acceleration, and rational data interaction.

To determine the optimal strategy for allocating hierarchical tasks, we carry out extensive experiments to profile the runtime of each task module across different heterogeneous computing platforms. We further take our carefully designed allocation principles into consideration and determine the optimal **Fine-grained Hierarchical Module Allocation Strategy** to decompose the event-image fusion SLAM algorithm (Section 4.1).

To fully harness the computational power of each heterogeneous computational unit and reduce the algorithm operation latency, we design two **Hardware–Software Co-acceleration** technologies (Section 4.2) to accelerate the SLAM algorithm. (1) *Front-end Hardware Acceleration:* We first design tailored feature point detection and feature point matching algorithm fitting for FPGAs, significantly reducing the latency of two time-consuming sub-modules in the front-end. Additionally, by utilizing the bare-metal R-core, we further minimize the front-end latency. These techniques together ensure real-time pose estimation for drones. (2) *Back-end Scheduling Optimization:* We employ adaptive core isolation and multi-threading parallelization techniques to optimize the scheduling of the SLAM algorithm's complex back-end tasks. This significantly improves the utilization efficiency of multiple processors equipped by the computing platform, enabling timely mapping and promptly eliminating accumulated tracking errors for the front-end.

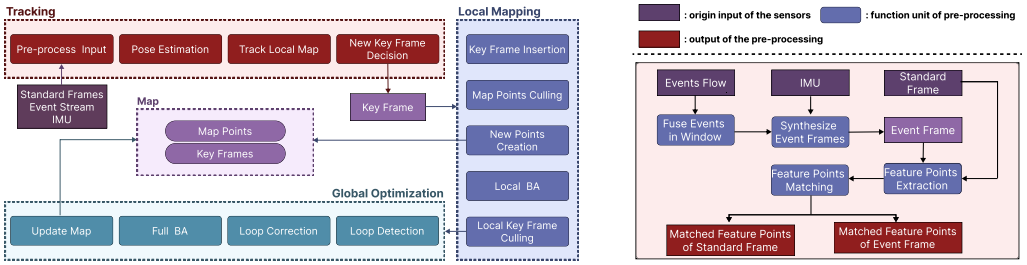
To minimize the considerable transfer latency between different heterogeneous computational units, we propose **Tri-DMA Across-Layer Data Exchange** technology, which adopts **Direct Memory Access (DMA)** to facilitate data exchange between different hierarchical computational units, achieving high-speed data exchange and ensuring reliable collaboration among several SLAM modules (Section 4.3).

We fully implement *TrinitySLAM* on the latest Zynq UltraScale+ platform. Comprehensive experiments are carried out on one custom-made drone dataset using a drone equipped with a DAVIS event camera and four official datasets [28] covering various scenarios. We compare *TrinitySLAM* with two state-of-the-art heterogeneous computing platform acceleration systems [1, 10] in terms of accuracy, latency, and energy consumption. Evaluation results show that *TrinitySLAM* significantly outperforms all baselines. Compared with the closest performing system, the accuracy of *TrinitySLAM* is enhanced by 28%, the end-to-end latency consistently decreases by more than half, remaining within real-time benchmarks, and the energy consumption is reduced by 1.2 times. We will make the code and details of the implementation public available upon acceptance.

Our contributions are summarized as follows.

- We design the system architecture of *TrinitySLAM*, as far as we are aware of, the first onboard event-image fusion SLAM system that can offer real-time, high-accuracy, and low-energy consumption performance.
- We propose several technologies: the Fine-grained Hierarchical Module Allocation Strategy ascertains the optimal task allocation scheme; the Hardware-Software Co-acceleration enhances the speed of the SLAM algorithm; and the Tri-DMA Across-Layer Data Exchange ensures minimal data transfer latency.
- We implement a prototype system on the Zynq platform and conduct extensive evaluations using both our custom-made drone dataset and official datasets, comparing them with the state-of-the-art baselines. Evaluation results demonstrate that, across various scenarios, *TrinitySLAM* achieves superior performance in terms of accuracy, latency, and energy consumption.

The rest of this article is organized as follows. We present the research background and analyze preliminary experiments to elucidate our motivations in Section 2. Section 3 contains a brief



(a) Common architecture of image-based SLAM. Figure referred from [30].

(b) The Input Pre-processing sub-module of event-image fusion SLAM.

Fig. 2. System architecture of event-image fusion SLAM.

introduction of the heterogeneous platform we employed and an overview of *TrinitySLAM* . Our proposed key technologies and strategies are presented in Section 4. Details of the implementation setup are presented in Section 5, followed by the performance evaluation results in Section 6. Related works are presented in Section 7. In Section 8, we provide a detailed discussion of the limitations of our work and directions for future research. We conclude the article in Section 9.

2 Background and Motivation

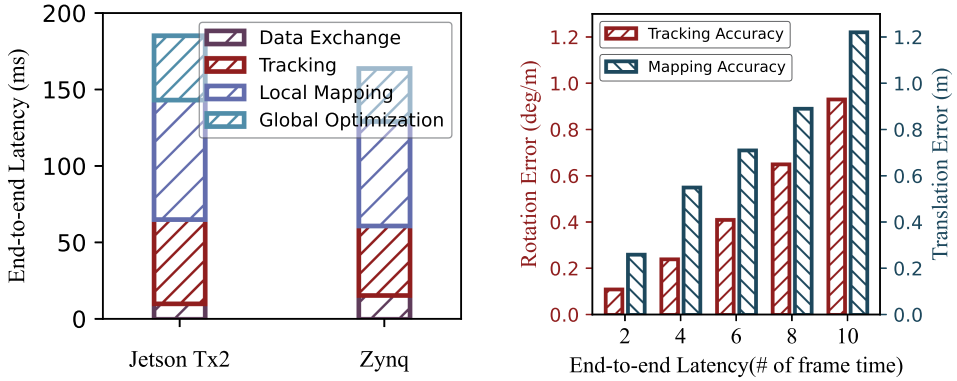
2.1 Event-Image Fusion SLAM

The system architecture of event-image fusion SLAM is essentially consistent with conventional image-based SLAM architecture (as shown in Figure 2(a)). Compared with image-based SLAM, event-image fusion SLAM possesses not only a pipeline for processing standard frames but also another pipeline dedicated to handling event data that synthesizes motion-compensated event frames with the standard frames in the pre-process input sub-module (as shown in Figure 2(b)). Event-image fusion SLAM also consists of three components: tracking, local mapping, and global optimization, as shown in Figure 2. We briefly introduce each module.

— The **tracking** module estimates the coarse-grained pose based on the consecutive standard frames and synthetic event frames. As shown in Figure 2(b), before processing the standard frames from the traditional camera, the system first synthesizes motion-compensated event frames. By utilizing the high-frequency IMU data, it synchronizes the continuous event streams from the event camera on the timestamps of the standard frames (e.g., for a 30-fps camera, this would be 33.3 ms). Once the event frame and standard frame are generated, feature extraction algorithms are applied to extract 2D feature points. These feature points are then matched with the 2D feature points extracted from the previous frame. Then, 3D map points are selected from nearby keyframes to match the current frame's 2D feature points. Finally, it employs triangulation algorithms to obtain a more accurate pose estimation. The tracking module is also commonly referred to as the *front end*.

— The **local mapping** module then creates new 3D points in the map storage. The local pose can then be optimized by solving a Bundle Adjustment problem. This module runs repeatedly with the continuous arrival of event frames and standard frames, resulting in a trajectory of the camera pose, a map of the 3D landmarks and the corresponding keyframes.

— The **global optimization** module consists of two crucial steps: loop closure detection and global pose optimization. In loop closure detection, features extracted from image frames are compared with keyframes. If a substantial similarity with a certain keyframe is detected, a global pose optimization for all keyframes is conducted by solving a larger Bundle



(a) The end-to-end latency on different heterogeneous computing platform.

(b) The tracking and mapping accuracy with respect to different end-to-end latency.

Fig. 3. The latency and accuracy analysis.

Adjustment problem. The map is then updated based on the optimization results. The local optimization sub-module and global optimization module are collectively referred to as the *back end*.

2.2 Latency and Accuracy Analysis

Latency Analysis. Simply running event-image fusion SLAM on a heterogeneous computing platform without reasonable task allocation and without full computational power utilization, is hard to run in real-time due to its extensive computation overhead. We have evaluated the end-to-end latency of the event-image fusion SLAM algorithm, UltimateSLAM, on two representative heterogeneous computational platforms: the NVIDIA Jetson TX2 [30] (utilizing both its CPU and GPU) and the Zynq [18] (leveraging both the Cortex-A53 and Cortex-R5). We measure the latency of each functional module and data exchange. The result is shown in Figure 3(a). From the results, it is evident that regardless of the heterogeneous computational platform employed, the end-to-end latency exceeded 150 ms. This implies that the SLAM system processes at a speed of less than 7 fps, which falls significantly short of real-time requirements ($\geq$30 fps). It is worth noting that the latency of data exchange is not negligible (i.e., on the Zynq platform, it exceeds 10 ms). This highlights the importance of accurately pinpointing the "hourglass position" and accelerating the data transfer speed to reduce data exchange latency.

Accuracy Analysis. We conduct further experiments to delve into the impact of different latency intervals on the system's localization accuracy. We recorded the 3D position trajectory output by UltimateSLAM. By setting varying latency intervals, we calculated the corresponding translation error and rotation error between the 3D trajectory and the ground truth to evaluate mapping and localization precision. Figure 3(b) depicts the translation and rotation errors under different end-to-end latency intervals. It is evident that when the system latency is low (i.e., four-frame delay), both errors remain at a relatively low level. However, as the latency reaches 333 ms (a gap of 10 frames), the mapping error exceeds 1 m. This result clearly demonstrates that end-to-end latency has a significant impact on the system's performance.

3 System Design

We leverage the latest commercial Zynq UntralScale+MPSoC [18] (abbreviated as **MPSoC**), a heterogeneous computing platform launched by Xilinx [18], to implement *TrinitySLAM* through

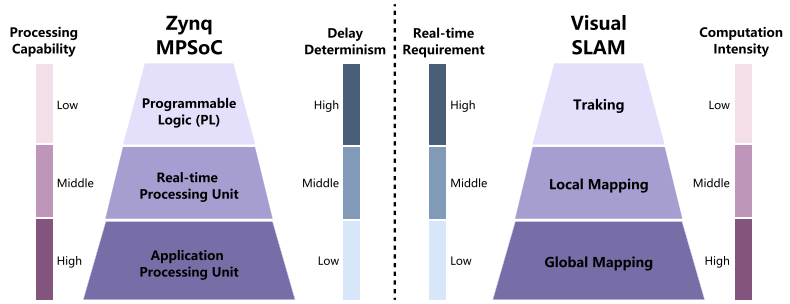


Fig. 4. Hierarchical computing architecture of MPSoC and hierarchical task architecture of SLAM.

elaborate software and hardware co-design. We will briefly introduce the MPSoC platform and then present our system design.

3.1 MPSoC Platform Primer

The left half of Figure 5 shows the versatile hierarchical heterogeneous computational units provided by MPSoC, including a **processing system (PS)** for software development) and **programmable logic (PL)** for hardware design, two modules. The PS features a 64-bit Cortex-A53 quad-core processor (4*A-Core) and a Cortex-R5 dual-core real-time processor (2*R-Core). The four A-Cores are typically centralized and scheduled by a general-purpose **operating system (OS)**, for example, PetaLinux [15] and Debian [6]. The two R-Cores, designed for real-time applications, are typically scheduled by a real-time OS (e.g., upgraded CentOS with Xenomai [44]), enabling a shorter response time compared with a general-purpose OS. The PL possesses FPGA computational units, providing both PL units and **digital signal processing (DSP)** units, which need hardware design based on specific task requirements. Benefiting from the above versatile computational resources, MPSoC is considered a critical driving force for many essential applications, such as intelligent robotics [21], autonomous driving [20], and the industrial **Internet of Things (IoT)** [39].

Obviously, MPSoC has three tiers of computational units: the FPGAs, two R-Cores, and four A-Cores. As shown in Figure 4, when we move through these units, their hardware architecture becomes increasingly generic; the processing power strengthens, but the deterministic latency gradually diminishes. Similarly, the three modules of event-image fusion SLAM — tracking, local mapping, and global optimization — exhibit increasing computational complexity with decreasing real-time requirements.

However, we find that simply migrating these three SLAM modules onto tiered MPSoC without an elaborate software and hardware co-design could not achieve desired performance (Section 6.3). There are still three challenges that ought to be addressed: (i) how to design an optimal allocation strategy for maximizing the strengths of each computation unit; (ii) how to further accelerate the tasks on different computational units for minimizing operation latency, and (iii) how to achieve efficient data transfer between different computational units for minimizing the transmission delay.

3.2 TrinitySLAM's Architecture Design

The right half of Figure 5 sketches the architecture of *TrinitySLAM*. To tackle the above challenges, *TrinitySLAM* first comprehensively considers the operation time of various sub-modules on the heterogeneous computational units of MPSoC as well as the inherent characteristics of each computational unit. Consequently, we devise a *Fine-Grained Hierarchical Module Allocation* strategy,

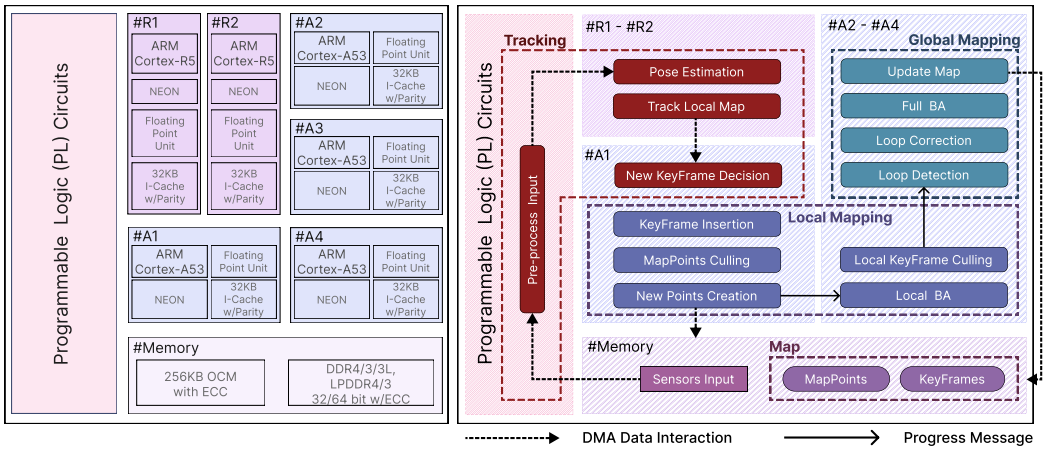


Fig. 5. *TrinitySLAM*'s architecture.

ingeniously mapping the sub-modules of the event-image fusion SLAM onto different computational units of the MPSoC. On this basis, we introduce a suite of software and hardware co-design techniques to guarantee real-time performance.

As for the specific workflow, *TrinitySLAM* first leverages the *Front-End Hardware Acceleration* method to further accelerate the feature extraction and feature matching sub-modules of the tracking module on the FPGAs. Then, it runs the Pose Estimation and Track Local Map sub-module on two R-cores. Based on the results of the tracking module, *TrinitySLAM* introduces the *Back-End Scheduling Optimization* scheme to enable timely optimization of the coarse-grained pose for the front end. It utilizes the *Tri-DMA Cross-Layer Data Exchange* technology to minimize the transmission latency of intermediate data among different computational units, including the origin sensor input to the PL, the matched feature points from the PL to the R-cores, and map points between the R-cores and the A-cores.

In the following section, we will present the details of the above proposed technologies.

4 Real-Time On-Board Event-Image Fusion SLAM

4.1 Fine-Grained Hierarchical Module Allocation Strategy

The strategy that allocates hierarchical event-image fusion SLAM sub-modules to the heterogeneous computing units of the MPSoC should fulfill two principles: (1) The system's localization (tracking camera pose) and mapping (creating map points) functionalities need to be executed in real time. Specifically, the latency of the modules associated with these functionalities — including the entire tracking module and relevant parts of the local mapping module, collectively referred to as the *front end* — should be less than 33.3 ms based on a benchmark frame rate of 30 fps. (2) Regarding the storage capacities of both the PS and the PL, only the PS possesses the requisite ability to store the map. Given the intensive interactions of the pose optimization functions (i.e., the back end) with the map database, they should also be assigned to the PS. Otherwise, there would be frequent data transfers leading to significant transmission delays.

To inform our task allocation strategy, as illustrated in Figure 6, we measure and analyze the runtime of each sub-module on both A-core and R-core. It is evident that relying solely on the A-core results in a cumulative front-end latency exceeding 60 ms, which fails to meet the real-time requirements. Moreover, the pose optimization and map updates in the back end are particularly time-intensive, markedly hampering the rate of cumulative error corrections. Although the R-core

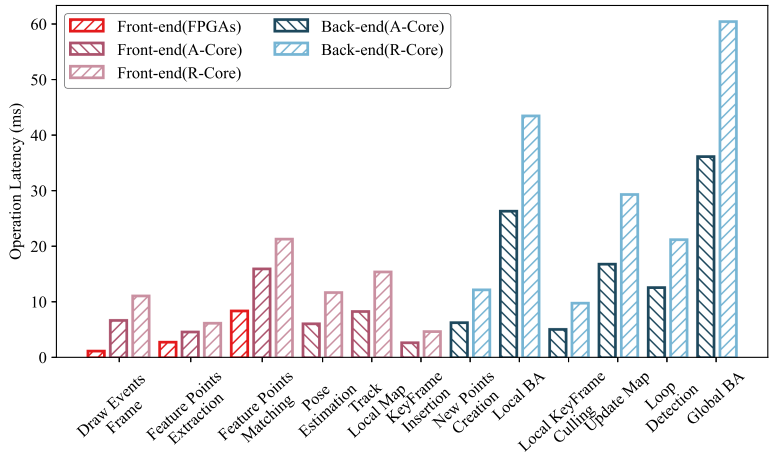


Fig. 6. Operation latency of each function module of *TrinitySLAM* running on the MPSoC.

has a longer overall runtime compared with the A-core, it is essential to leverage its computational resources to alleviate the computational burden on the A-core.

Our proposed fine-grained hierarchical module allocation strategy effectively aligns with the aforementioned principles. As shown in Figure 5, the input pre-processing sub-module, the most time-intensive within the tracking module, is promptly processed on the PL. we migrate the input pre-processing sub-module onto the FPGAs for two reasons: (i) the high-frequency logic units on the FPGAs are adept at processing equally high-frequency event streams; and (ii) FPGAs excel at parallel acceleration for matrix operations fundamentally, such as feature extraction and feature matching.

The pose estimation and local map tracking sub-modules are assigned to the R-cores to fully exploit its computational strength (for detailed latency discussion, see Section 6.1). The subsequent modules, including keyframe decision and map point update, are designated to a specific A-core (e.g., #A1). This architecture ensures that the system's localization functionality is executed in real time. Concurrently, the time-intensive and resource-intensive pose optimization tasks, including local and global pose optimizations, are executed on the other three A-cores (e.g., #A2–#A4). When the system identifies a keyframe, it undertakes local pose optimization. If loop closure is ascertained, it also activates the global pose optimization module, updating the optimized pose and map point data in the existing map. This assists the tracking module in correcting accumulated errors in pose tracking and map points.

4.2 Hardware–Software Co-acceleration

4.2.1 Front-End Hardware Acceleration.

FPGA Acceleration. Simply migrating the input pre-processing sub-module to FPGAs results in a relatively high operation latency (as illustrated in Figure 6, it exceeds 10 ms). This is because the primary strength of FPGAs lies in their parallel processing capabilities. A straightforward migration may not fully exploit the FPGA's parallelism, thus, we should further design customized algorithms for FPGAs to accelerate computation. As detailed in Figure 7, we utilize FPGAs to parallelize and accelerate the computations for three tasks within the input pre-processing module: (i) synthesizing event frames from the event stream, (ii) feature extraction, and (iii) feature matching. The process of synthesizing event frames from the event stream adheres to Equation (1), where X_i represents the i^{th} event data. Both π (the projection matrix of the event camera) and T_{t_k,t_i} (the

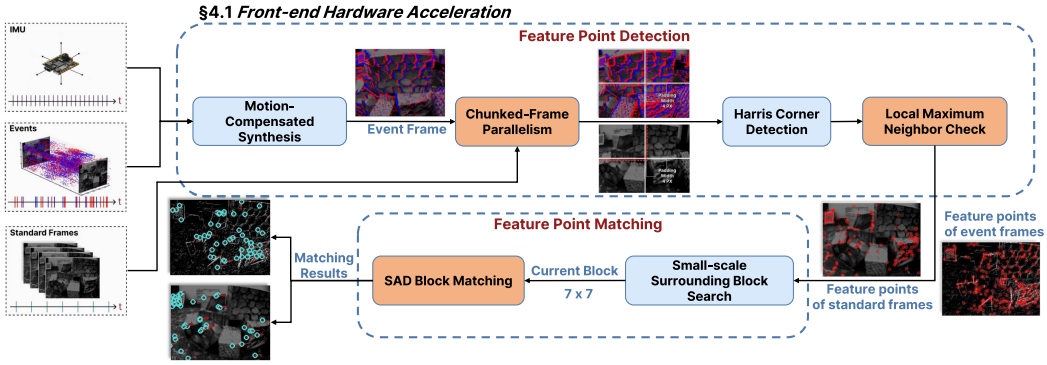


Fig. 7. The parallelism processing of the pre-process input sub-module on the FPGAs.

pose transformation obtained from the **Inertial Measurement Units [IMUs]**) are known parameters. Notably, each event frame generation involves independent computations for individual event data. Therefore, by leveraging the millions of logic units within the FPGAs, we have substantially expedited this process.

$$X_i' = \pi \left(T_{t_k, t_i} \left(Z(X_i) \pi^{-1}(X_i) \right) \right) \tag{1}$$

After generating the event frames, we initially divide both the event frame and the standard frame into four equal sub-blocks, thereby enabling the subsequent computations to be parallelized into four threads in our hardware structure and the kernels processed in each thread are all pipelined, which significantly enhances the parallelism of the subsequent feature detection and feature matching. Then, to ensure the high quality of the detected feature points, we employ the Harris feature detection method, which is more robust and can handle scenarios with viewpoint rotations [4]. For further hardware acceleration, we adopt the local maximum neighbor check mechanism. We assess the center value by comparing it with the surrounding values rather than sorting based on the global Harris value. This approach prevents pipeline waiting, which can impact the execution speed. Finally, we design an improved KLT algorithm for feature matching. Unlike the original KLT [25], which constructs image pyramids to search over a relatively large range to obtain an accurate matching pair, we propose our improved KLT algorithm considering the proximity of adjacent frames. Specifically, we restrict our search within the 7x7 surrounding block, which is not only more efficient but also sufficient to achieve high matching accuracy. During the block matching process, we employ the **sum of absolute difference (SAD)**, which offers reduced computational complexity, as opposed to the **sum of squared difference (SSD)**. Thus, we have designed hardware-friendly, expedited algorithms for feature detection and matching that deliver ultra-low latency.

Bare-metal R-core. We also leverage the R-core to accelerate pose estimation and map point matching tasks, subsequently outputting a preliminary pose estimation. First, we offload the Pose Estimation and the Track Local Map sub-module onto the two R-cores for several reasons: (i) The R-core is specifically tailored for real-time applications, offering high levels of reliability and determinism. (ii) Both of these modules have straightforward input and output, which reduces the data exchange volume with the FPGAs and A-Core. However, the R-core's computational capacity is objectively inferior to that of the A-core. To further accelerate the sub-modules on the R-cores, although it is common to run a real-time operating system on the R-core for improved resource management and sophisticated task scheduling, we finally employ bare-metal R-cores (i.e., R-cores with no operating system) to bypass the uncertain OS scheduling delay.

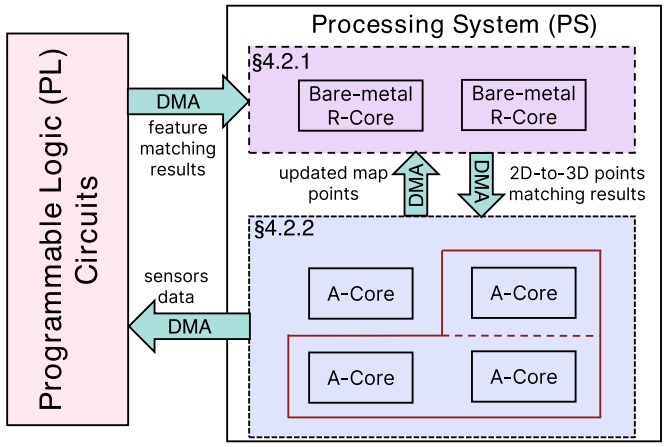


Fig. 8. Illustration of adaptive A-core isolation and Tri-DMA data exchange.

4.2.2 Back-End Scheduling Optimization. The back end of SLAM is responsible for local and global pose optimization based on historical keyframes and map points, thereby rectifying the cumulative tracking errors from the front end. Clearly, the operational speed of the back end has a significant impact on the SLAM system's accuracy. However, given that the back-end tasks are highly computation intensive and we only have four A-cores at our disposal, we have devised a series of resource scheduling and optimization strategies to accelerate the back end's performance on these four A-cores.

Adaptive A-core Isolation. The computational resources on the A-core are scheduled by a time-sharing operating system (e.g., Debian [6]), leading to potential competition among different tasks. To mitigate unnecessary resource contention and enhance program efficiency, we initially employ adaptive core isolation techniques, assigning distinct processes to each core. One core (e.g., #A1) is allocated to run keyframe decisions and some local mapping tasks with modest computational demands. The remaining three A-cores cater to tasks requiring higher computational power, including local pose optimization and global pose optimization. During the allocation process for the three A-cores, considering the relationship between the operational frequency of local pose estimation and global pose estimation and their required computational load, we opted for an adaptive allocation strategy. Specifically, the frequency of local pose optimization (corresponding to the keyframe frequency) is much higher than that of global pose optimization (aligned with loop closure detection frequency). However, the computational resources required for a single optimization are relatively less. Thus, when loop closure detection is unsuccessful, all three A-cores are used for local pose optimization. However, upon successful loop closure detection, one A-core is allocated for local pose optimization, whereas the other two are dedicated to global pose optimization (as shown in Figure 8).

OpenMP Parallelization. Given our use of multiple cores to run the computationally demanding pose optimization module, we employ OpenMP technology in our implementation to transform the original serial back-end tasks into parallel code. This generated multiple threads to concurrently execute the converted code segments, ultimately enhancing the program's execution efficiency and improving the utilization of multiple processors.

4.3 Tri-DMA Cross-Layer Data Exchange

When sub-modules are allocated across different computational layers, data exchange becomes a consideration. Despite efforts to minimize it, the volume of data that needs to be exchanged is also

significant. Therefore, we also strive to maximize the speed of the data exchange. We utilize DMA technology to facilitate the data exchange process between different hierarchical computational units.

DMA allows data to be transferred directly between the memory and peripheral devices (such as the FPGA) without the intervention of the CPU. In traditional data transfer schemes, the CPU is heavily involved in managing and moving data between memory and devices, which consumes CPU cycles and limits the data transfer bandwidth. By employing DMA, the CPU can initiate a data transfer transaction and then proceed with other tasks while the DMA controller takes over the actual data movement. Moreover, modern DMA controllers, such as the one in the Zynq UltraScale+ platform, support advanced features such as scatter-gather and multi-channel modes. Scatter-gather allows non-contiguous memory blocks to be transferred in a single DMA transaction, reducing the setup overhead. The multi-channel mode enables parallel data transfers between different source-destination pairs, further enhancing the data exchange bandwidth.

Conventionally, DMA enables distinct computational units to access the main system memory, allowing for data transfer between the CPU (A-core and R-core). To illustrate, consider the data interaction between an A-core and an R-core. Without DMA, when the R-core carries out programmed input/output, the entire program would be fully occupied by read/write operations and, thus, receive the data from the A-core with frequent blocking. In contrast, with DMA, data transfer channels between the R-core and A-core are initialized first, allowing the program to operate concurrently during data transmission. Inspired by the observations mentioned above, we have established three different types of DMA channels bridging the three-tiered heterogeneous computational units (PL, A-core, R-core) as shown in Figure 8: (1) the A-core transmits original event streams, video frames, and IMU sensor inputs to the PL via DMA; (2) the PL sends feature extraction and matching results to the R-core; and (3) the R-core and A-core exchange intermediate results, with the R-core forwarding the matching results between 2D feature points with 3D map points to the A-core, and the A-core returning optimized map points to the R-core. Unlike existing network-based solutions such as the PL-PS Ethernet interface [16] and OpenAMP [32], our DMA approach provides each critical dataflow with its exclusive transfer channel, ensuring real-time data exchange throughout the task life cycle. Equally vital, compared with network-based solutions, using DMA channels for data transfer between computational units significantly reduces CPU overhead.

5 Implementation

5.1 Experiment Setup

We implemented the *TrinitySLAM* system using Xilinx's recently released commercial heterogeneous computing platform, Zynq UltraScale + MPSoC [18]. Our improved Harris feature extraction and KLT feature matching algorithms running on the FPGAs in the PL were implemented in the C programming language with the Vivado **High-Level Synthesis (HLS)** tool [17]. The remaining functional modules in the PS of the MPSoC were modified and implemented based on the code library from UltimateSLAM [41]. The core isolation feature was realized by setting the parameter *cpu_affinity=<cpu number>* in the code, whereas OpenMP thread parallelization was achieved by adding compilation options. Simultaneously, to validate the performance of our system in real-world deployment, we have made a dataset using a drone equipped with the DAVIS 346 event camera [14]. Details of this dataset can be found in Section 5.2.

5.2 Datasets

To analyze the performance of *TrinitySLAM* across different scenarios, we selected four official datasets that merge event cameras with traditional cameras [28], encompassing diverse lighting conditions, environmental texture complexities, and other characteristics. Notably, as the official

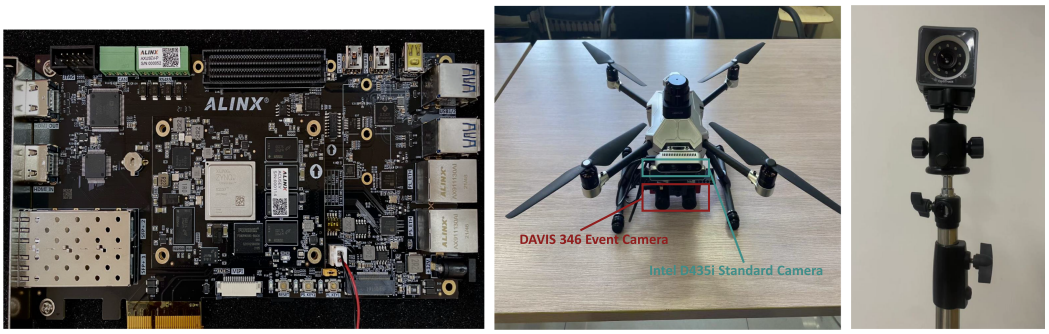


Fig. 9. The experiment devices. (a) Zynq UltraScale + MPSoC. (b) Drone with event camera and standard camera. (c) OptiTrack motion capture system.

datasets lack scenes with moving objects, we craft a custom Drone dataset using a drone equipped with the DAVIS 346 event camera (as shown in Figure 9). For accurate pose tracking, infrared reflective markers were placed on the drone's rotors, and an advanced motion capture system, OptiTrack [33], was employed to obtain the ground truth. Detailed characteristics of the four public datasets and our custom Drone dataset can be found in Table 1. Comprehensive evaluations will be conducted on these datasets to evaluate the system's accuracy, end-to-end latency, and energy efficiency.

5.3 Baselines

To extensively evaluate the performance of our hardware and software co-design of *TrinitySLAM* , we compare *TrinitySLAM* with two relevant state-of-the-art heterogeneous computing platform acceleration systems, **Baseline(MPSoC)** [10] and **Baseline(GPU+CPU)** [1]. The former is a system that also employs Zynq UltraScale + MPSoC to expedite image-based SLAM (KinectFusion). The latter proposed a series of methods to accelerate image-based SLAM (ORB-SLAM) using the heterogeneous CPU and GPU in the NVIDIA Jetson Tx2 platform. For a fair comparison, we replace both the SLAM systems of the above two baselines with our consistent event-image fusion SLAM algorithm.

5.4 Metrics

We employ three metrics to comprehensively evaluate the performance of *TrinitySLAM* and the two baselines: accuracy, end-to-end latency, and energy consumption.

- **Accuracy:** We adopt two standard metrics, **Translation Error (TE)** and **Rotation Error (RE)**, to evaluate the localization accuracy of the systems.
- **End-to-End Latency:** In the context of the SLAM system, end-to-end latency represents the average time taken to output a pose and finalize the mapping. Both front-end latency and back-end latency will influence the end-to-end latency: the front-end latency determines the delay of outputting the pose and the back-end latency affects the front-end latency. Specifically, when the keyframe occurs, the front end will wait for the back end to return the optimized pose and updated map. We obtain the end-to-end latency by logging the latency statistics of each sub-module in the front end and back end.
- **Energy Consumption:** This refers to the average energy consumed by the system while processing a single frame during its continuous operation, measured in millijoule/frame. We measure the energy consumption by obtaining the average power during the program's

Table 1. Dataset Summary

Name	Features	Number of Events	Video Length
Post_Normal	Flat walls, normal lighting condition	133,464,530	1,357 frames
Post_HDR	Flat walls, high dynamic range	102,910,720	1,491 frames
Boxes_Normal	Complex textures, normal lighting condition	133,085,511	1,297 frames
Boxes_HDR	Complex textures, high dynamic range	118,499,744	1,387 frames
Drone	Dynamic environment, moving people	157,174,637	3,430 frames

execution. On the Zynq MPSoC, the power is obtained using the voltage monitoring data provided by the Xilinx Analog-to-Digital Converter module; on the Jetson TX2, it is obtained using the JTOP tool. It is worth noting that this only considers the energy consumed during program execution and excludes the camera's inherent energy usage.

6 Evaluation

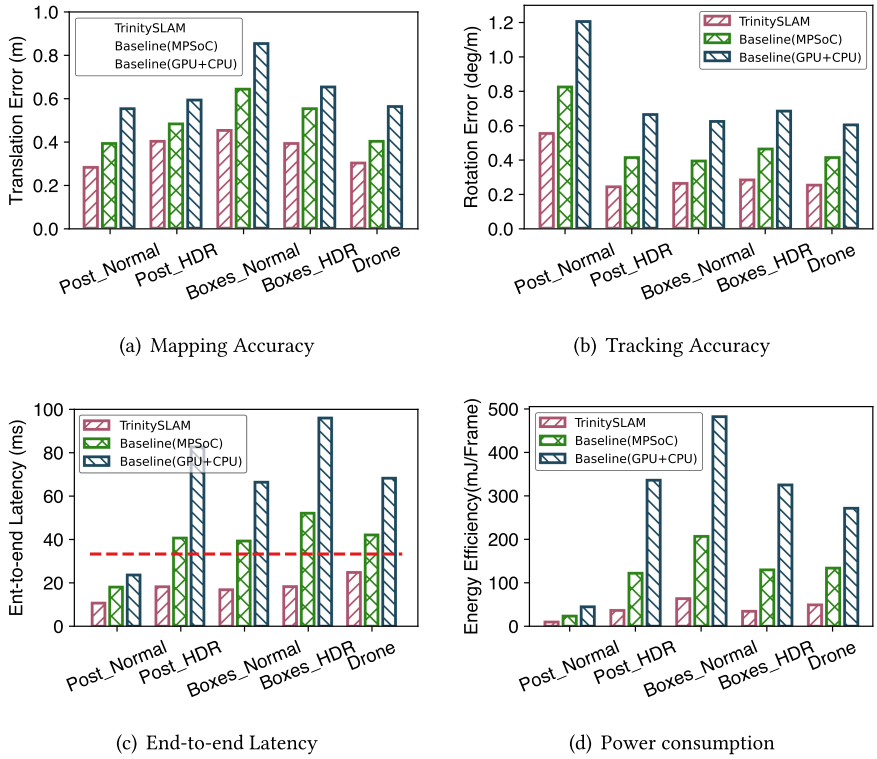
6.1 Overall Performance

Figure 10 illustrates the overall performance of *TrinitySLAM* and two baselines. In a nutshell, *TrinitySLAM* achieves much higher localization accuracy, lower end-to-end latency, and lower energy consumption under five distinct datasets.

Overall Accuracy Comparison: Figure 10(a) and 10(b), respectively, depict the translation error and rotation error of *TrinitySLAM* and two comparative baselines across five distinct datasets. A comprehensive analysis of the results from multiple datasets reveals that *TrinitySLAM*'s average translation error is only 0.36 m, respectively achieving an improvement of 28% and 43.1% compared with Baseline(MPSoC) and Baseline(GPU+CPU). In addition, *TrinitySLAM*'s rotation error is merely 0.32 deg/m, which is an enhancement of 36.1% and 57.7% compared with Baseline(MPSoC) and Baseline(GPU+CPU), respectively. Baseline(GPU+CPU) exhibits a significant gap when compared with both Baseline(MPSoC) and *TrinitySLAM*. We hypothesize that one reason is the superior CPU performance of the MPSoC compared to that of the Jetson Tx2. Additionally, the CPU or GPU may not be adept at processing event data. It is noteworthy that, compared with Baseline(Mpsoc), we utilize the same hardware. However, *TrinitySLAM* clearly achieves superior localization accuracy. This underscores the efficacy of our proposed FPGA acceleration technology and several hardware–software co-design technologies and how they more efficiently harness the potential of the hierarchical computational units in MPSoC.

End-to-End Latency Comparison: We measure the end-to-end latency introduced by both the front-end latency and back-end latency across various datasets. As illustrated in Figure 10(c), the red dashed line represents the real-time benchmark of 33.3 ms. It is evident that *TrinitySLAM*'s average end-to-end latency across the five datasets is only 17.7 ms, consistently staying below the real-time benchmark, confirming *TrinitySLAM*'s real-time localization capability. Baseline(GPU+CPU) exhibits an end-to-end latency of up to 96 ms in the Boxes_HDR dataset, underscoring the limitations of conventional approaches in ensuring real-time operation for event-image fusion SLAM systems in complex environments. Compared with Baseline(MPSoC), *TrinitySLAM* reduces the end-to-end latency by 20 to 30 ms, ensuring stable real-time performance, which directly demonstrates the superiority of our proposed hardware–software co-acceleration technology. Notably, the two baselines only achieve real-time performance in the Post_Normal dataset (detailed reasons will be analyzed in Section 6.2), whereas the latency in the remaining four datasets significantly exceeds the real-time benchmark.

Energy Consumption Comparison: Power consumption is crucial for many SLAM applications as it directly impacts device battery life. We measured the average power of *TrinitySLAM* and



(a) Mapping Accuracy (b) Tracking Accuracy

(c) End-to-end Latency (d) Power consumption

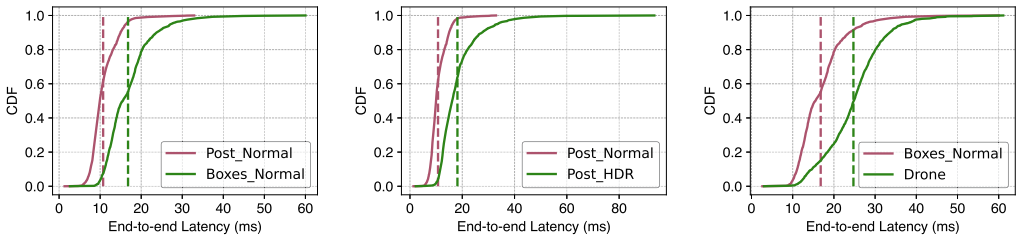
Fig. 10. The overall performance comparison of *TrinitySLAM* and all baselines on various datasets.

baselines across various datasets. The product of this average power and the latency to process one frame gave us the average energy consumption required to process a single image. As depicted in Figure 10(d), the average energy consumption of *TrinitySLAM* is merely 17.7 millijoules. In comparison, Baseline(MPSoC) consumes 38.4 millijoules, and Baseline(GPU+CPU) consumes 67.2 millijoules, which means that TrinitySLAM is more efficient, consuming 1.2 times less energy than Baseline(MPSoC) and nearly 2.8 times less than Baseline(GPU+CPU). We identified two primary reasons for *TrinitySLAM*'s markedly low average energy consumption: (1) A substantial amount of computation is offloaded to the FPGA, alleviating the computational burden on the PS, with the PL operating at significantly lower power compared with the PS. (2) *TrinitySLAM*'s average latency for processing one frame is considerably less than that of the two comparative systems, resulting in lower energy consumption.

6.2 Robustness Study

We further conduct experiments to analyze the end-to-end latency of *TrinitySLAM* in different application scenarios to gauge the system's robustness in real-time localization under different environments.

Impact of Texture Complexity: Figure 11(a) illustrates the distribution of end-to-end latency for *TrinitySLAM* on both the Post_Normal and Boxes_Normal datasets, with the dashed line representing the average latency. It is evident from the figure that the end-to-end latency on the Post_Normal dataset is significantly lower than that on the Boxes_Normal dataset. Upon comparing the two datasets, we noticed that both feature consistent ambient lighting and include stationary objects within their viewing fields. The major difference lies in the environmental



(a) Impact of texture complexity on latency distribution.

(b) Impact of lighting condition on latency distribution.

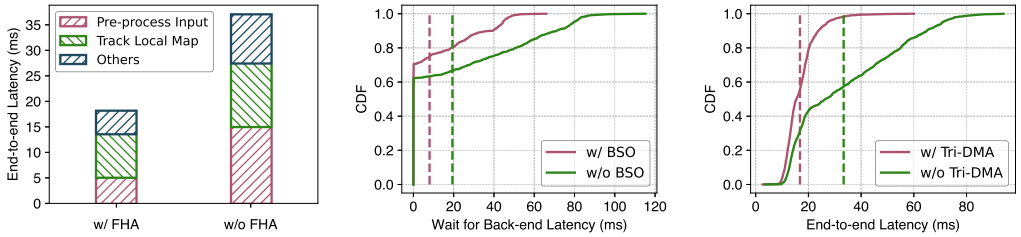
(c) Impact of moving objects on latency distribution.

Fig. 11. Robustness study of *TrinitySLAM*.

texture; the Boxes_Normal dataset has a significantly more complex texture than the Post_Normal dataset. An in-depth analysis of *TrinitySLAM*'s performance on the two datasets reveals that, in the Post_Normal dataset, the simpler environmental texture leads to a scarcity of feature points. Consequently, there is a sparse distribution of keyframes, which, in turn, causes the back-end pose optimization processes to occur infrequently. The front-end then proceeds without waiting for the back end's pose optimization outcomes and needs only minimal map point updates. These combined factors lead to a significant reduction in end-to-end latency. Yet, the extended lack of pose optimization contributes to an accumulation of pose estimation errors at the front end, leading to heightened system localization errors. This trend is substantiated by the pose tracking accuracy presented in Figure 10(b): the rotation error of all systems under the Post_Normal dataset is notably higher than that in other datasets.

Impact of Lighting Conditions: To showcase the efficacy of our system in handling high dynamic range environments, we compared the end-to-end latency distributions of the Post_HDR dataset, which features higher dynamic range lighting, with those of the Post_Normal dataset as depicted in Figure 11(b). While both datasets depict weak-textured scenes, the Post_HDR dataset demonstrates a substantially higher average end-to-end latency and a more diverse latency distribution compared with the Post_Normal dataset. This disparity arises primarily due to the susceptibility of standard cameras to malfunction in weak-textured environments with high dynamic range lighting. In contrast, the event camera captures more environmental details in high dynamic range settings, yielding event frames enriched with feature points. This results in the creation of a greater number of keyframes and map points for back-end pose optimization. As a consequence, the maximum end-to-end latency for the Post_HDR dataset (80ms) is significantly greater than that of the Post_Normal dataset (40 ms). Furthermore, given the limitations of standard frames, event frames hold increased significance during the optimization process, which improves pose optimization accuracy. Alongside more frequent pose optimizations, this guarantees that *TrinitySLAM* attains superior accuracy on the Post_HDR dataset compared with the Post_Normal dataset. By leveraging the enhanced perception capabilities of event cameras, *TrinitySLAM* ensures real-time system operation while adeptly handling high dynamic range scenarios.

Impact of Moving Objects: We also compared the end-to-end latency distributions between the Boxes_Normal dataset and the Drone dataset to investigate the influence of moving objects on system performance. As depicted in Figure 11(c), even though both the Boxes_Normal and Drone datasets exhibit complex environmental textures, the average end-to-end latency for the Drone dataset surpasses that of Boxes_Normal by almost 50%. By analyzing the runtime of each module in the system, we find that the Drone dataset predominantly experiences elevated latency during the map point matching phase compared with the Boxes_Normal dataset. By capitalizing on the high sensitivity of event cameras to moving objects, we observed an influx of event data



(a) *TrinitySLAM* w/ and w/o Front-end Hardware Acceleration (b) *TrinitySLAM* w/ and w/o Back-end Scheduling Optimization (c) *TrinitySLAM* w/ and w/o Tri-DMA Cross-layer Data Exchange

Fig. 12. Ablation study of *TrinitySLAM*.

particularly from the edges of these moving entities. This phenomenon significantly amplifies the feature point count in event frames, leading to an extended runtime when matching 2D feature points to their corresponding 3D map points. Notably, despite the increased end-to-end latency induced by moving objects, *TrinitySLAM* consistently maintains real-time operation, even within the context of the Drone dataset. Coupled with the exemplary performance of *TrinitySLAM* on the Boxes_Normal dataset, it is evident that *TrinitySLAM* is accomplished in handling scenarios involving rapid movements.

6.3 Ablation Study

We conducted a series of experiments to investigate the contributions of the various techniques proposed by *TrinitySLAM* in enhancing performance.

Front-End Hardware Acceleration: We removed the design of the **Front-End Hardware Acceleration (FHA)** technology, which includes FPGA acceleration designs and the bare-metal R-core. The input pre-processing and local map tracking sub-module were transferred to the A-core (#A1) to study its impact on *TrinitySLAM*. Figure 12(a) illustrates the end-to-end latency with and without the front-end hardware acceleration technology. The results indicate that the FHA technology contributes to a reduction of nearly 19 ms in the overall end-to-end latency. This improvement predominantly stems from the reduced latency of the input pre-processing and local map tracking sub-modules, accounting for a significant proportion of the latency enhancement achieved by *TrinitySLAM* over Baseline(MPSoC). It is evident that our front-end hardware acceleration technology plays a pivotal role in boosting the performance of *TrinitySLAM*.

Back-end Scheduling Optimization: Similarly, we removed various designs within the **Back-end Schedule Optimization (BSO)** technology and measured the latency of the front end waiting for back-end optimization results to investigate its impact on the system performance. Figure 12(b) depicts the distribution of waiting latency (the front end waits for the back end to return the optimized pose and updated map) with and without the back-end scheduling optimization module. As we can see from Figure 12(b), about 60% of the waiting latency is zero, signifying that the current frame is neither identified as a keyframe nor associated with loop closure detection. Figure 12(b) illustrates that when the back-end scheduling optimization technique is omitted, the average waiting latency for *TrinitySLAM* increases substantially from 8 ms to 19.4 ms, a dramatic rise of 142%, which will adversely impact the overall end-to-end latency, thereby reducing the system's accuracy. Clearly, the back-end scheduling optimization technology significantly enhances the efficiency of back-end optimization, leading to a marked reduction in front-end waiting latency.

Tri-DMA Cross-Layer Data Exchange: We substituted the DMA data transmission between PS and PL with the PS-PL Ethernet interface to investigate its impact on the performance of *TrinitySLAM*. Figure 12(c) shows the end-to-end latency distribution of *TrinitySLAM* with and without

the PS-PL DMA data exchange. The results reveal that the average end-to-end latency surged by approximately 15 ms. Additionally, latency variability increased, with the peak latency escalating by nearly 35 ms. Such changes can be linked to the continuous transmission of event stream data and image frames from PS to PL via the Ethernet port, which inherently introduces noticeable transmission latency. This emphasizes the DMA data exchange's superiority in ensuring not just low-latency communication between PS and PL but also the stability of system latency.

7 Related Work

Real-Time Visual SLAM Systems: Visual SLAM systems, which enable pose estimation for drones, have garnered significant attention from both the academic and industrial sectors, leading to continuous advancements in related technologies. A pressing challenge for researchers is how to execute Visual SLAM systems in real time on drones with extremely limited computational capacity, thereby achieving reliable high-precision pose estimation. In recent years, several studies have proposed enhancing the real-time capabilities of Visual SLAM systems by leveraging the computational power of edge devices [2, 26, 45]. This involves offloading specific tasks to edge servers, allowing these more powerful servers to return results faster, ultimately augmenting the system's overall real-time performance. Methods reliant on edge assistance can be vulnerable to network conditions. In situations with network instability or insufficient bandwidth, the latency introduced by network communication becomes a dominant factor, potentially resulting in delays exceeding those of systems not using edge servers. The *TrinitySLAM* system presented in this article operates entirely on local devices. Even when facing a more intricate visual SLAM system (integrated with input from event cameras), the system's real-time capability is maintained. *TrinitySLAM* does not rely on external device assistance, making it independent of network conditions and more versatile for intricate application scenarios.

FPGA-Based Acceleration for Visual SLAM: The highly parallel computational nature of FPGAs has drawn the attention of numerous researchers. A plethora of related works have explored using FPGAs to accelerate vision tasks [5, 13, 22, 24, 37]. For instance, the authors of [13] implemented the FAST and BRIRF algorithms on an FPGA to accomplish feature extraction and feature matching tasks. The authors of [24] executed the ORB feature extraction algorithm on an FPGA. The authors of [22] introduced a block-matching optical flow algorithm, which, when implemented on an FPGA, significantly sped up optical flow computations. The authors of [5] targeted the enhanced KLT algorithm and utilized an FPGA for parallel acceleration. The authors of [37] harnessed FPGA to accelerate and optimize the GMapping algorithm, a filter-based SLAM algorithm. Distinct from these works, which simply leverage FPGAs to accelerate a specific task, the *TrinitySLAM* system proposed in this article utilizes a heterogeneous computing platform to fully realize a low-latency, high-precision complex SLAM framework. For the first time, the FPGAs were employed to handle intricate operations related to event stream processing. We design improved feature detection and feature matching algorithms tailored for the FPGAs. This research designed schemes for processing unit parallelization and pipeline parallelization, enhancing FPGA computation.

Event-Based SLAM Algorithms: Event cameras, with their ability to handle high-speed and high-dynamic range scenarios, have garnered significant attention, leading to numerous studies on their integration into existing SLAM frameworks [12, 23, 35, 42, 46]. Zhou et al. [46] introduced a visual odometry algorithm solely based on stereo event cameras. Following this, Hidalgo-Carrió et al. [12] proposed a monocular visual odometry algorithm using only event stream input. Rebecq et al. [35] presented a visual-inertial odometry algorithm that leverages both event cameras and **Inertial Measurement Units (IMUs)**. Building on this, UltimateSLAM [42] introduced the first visual SLAM algorithm that integrates event cameras, traditional cameras, and IMUs, representing the most recent and popular SLAM approach incorporating these three sensor

inputs. This article introduces *TrinitySLAM*, aiming to implement and optimize event-image fusion SLAM on on-board computing platforms, achieving superior performance in terms of low latency, high precision, and energy efficiency. The techniques proposed in this article are orthogonal to the UltimateSLAM algorithm, ensuring that improvements to the SLAM algorithm can be swiftly integrated into our system.

8 Discussion

8.1 Applicability to Other Platforms

While *TrinitySLAM* is implemented and evaluated on the Xilinx Zynq UltraScale+ platform in our work, the proposed techniques have the potential to be applied or adapted to other heterogeneous computing platforms, such as NVIDIA Jetson TX2 and Jetson TX2 NX. The fine-grained hierarchical task allocation strategy can be generalized to partition the SLAM pipeline and map the modules to the CPU and GPU on Jetson platforms, considering their computation characteristics and data dependencies. The scheduling optimizations on multi-core CPUs are also applicable. However, the front-end hardware acceleration leveraging FPGAs needs to be redesigned to utilize the GPU's computing power on Jetson. Potential solutions include GPU-based feature extraction and matching algorithms. The efficient DMA-based data transfer should also be replaced by techniques suitable for CPU–GPU communication, such as zero-copy memory and unified memory. In summary, when migrating to other platforms, the general ideas behind the proposed techniques can be followed, but platform-specific characteristics need to be considered to tailor the implementations.

8.2 Limitations and Future Work

Our work pushes the boundary of real-time event-image fusion SLAM on embedded platforms. However, some limitations still exist, which point to future research directions. We believe these limitations and future work will inspire more research on high-performance SLAM systems to enable intelligent drones.

8.2.1 Enhancing Adaptability to Complex Environments. One limitation of the current *TrinitySLAM* system is that its adaptability to more complex environments, such as outdoor scenes with dynamic objects, needs further investigation and improvement. In real-world applications, drones often need to operate in challenging outdoor environments with moving objects such as vehicles and pedestrians. These dynamic objects can introduce outliers in the visual odometry and mapping process, degrading the SLAM performance.

To address this issue, we plan to explore robust outlier rejection techniques to filter out the features on moving objects. For example, we can leverage the high temporal resolution of the event camera to detect and track dynamic objects and then exclude the features on these objects during pose estimation and map updating. Incorporating semantic understanding of the environment can also help to identify and handle dynamic objects. By leveraging deep learning–based object detection and segmentation models, we can recognize and label different objects in the scene. This allows for adoption of adaptive strategies for feature extraction and matching based on the semantic information. However, the integration of these techniques may introduce extra computational overhead, requiring further optimization of the algorithm and computing system.

8.2.2 Integration with Downstream Navigation Tasks. Another important future direction is to integrate *TrinitySLAM* with downstream tasks such as obstacle avoidance and path planning to form a complete autonomous navigation system for drones. While *TrinitySLAM* provides reliable localization and mapping capabilities, it needs to work in close collaboration with planning and control modules to enable intelligent and safe navigation in complex environments.

This integration requires the co-design of SLAM and planning algorithms to ensure effective information sharing and coordinated decision-making. For example, the obstacle information from the mapping module should be efficiently transferred to the path planning module to compute safe and optimal trajectories. The planned trajectory should be fed back to guide the keyframe selection and local mapping process in SLAM. The co-design also needs to consider the computing system to balance the workload and meet the real-time requirements of each module.

One potential approach is to develop a hierarchical planning framework that leverages the multi-scale maps from *TrinitySLAM* . The global planning module can use the coarse global map to compute high-level navigation paths, whereas the local planning module can utilize the fine-grained local map to generate safe motion commands. The planning modules can be implemented on the CPU cores and closely interact with the SLAM modules through efficient inter-process communication mechanisms. We will investigate the co-design methodology and system architecture to enable tight integration and real-time performance of the entire navigation system.

9 Conclusion

Traditional visual SLAM techniques offer drones the capability for positioning and mapping, providing significant value in scenarios such as logistics delivery, emergency rescue, environmental exploration, and formation flight. However, these techniques face challenges in high-speed flight and dynamic environments, leading to decreased self-localization performance. In recent years, the emergence of event cameras has attracted extensive research to integrate them into the conventional visual SLAM framework to address issues of dynamic blurring and high dynamism. Yet, the incorporation of event cameras increases the computational complexity of the system, posing greater challenges for mobile computing platforms with limited computational power. This article presents a high-accuracy, low-latency, and low-energy consumption SLAM acceleration framework, *TrinitySLAM*, implemented on the lightweight embedded platform Zynq. We introduce a fine-grained hierarchical module allocation technique to deploy the intricate SLAM system on a tiered computing platform. Through a collaborative software–hardware design, we propose a series of computational acceleration techniques, enhancing the overall system performance. Extensive experimental tests on official datasets and custom drone datasets indicate that the performance of *TrinitySLAM* surpasses the closest comparative system by improving localization accuracy by 28%, consistently reducing end-to-end latency by more than half, always maintaining real-time benchmarks, and reducing energy consumption by 1.2 times.

References

[1] Stefano Aldegheri, Nicola Bombieri, Domenico D. Bloisi, and Alessandro Farinelli. 2019. Data flow ORB-SLAM for real-time performance on embedded GPU boards. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'19)*. IEEE, 5370–5375.

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

[3] Konstantinos Boikos and Christos-Savvas Bouganis. 2016. Semi-dense SLAM on an FPGA SoC. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. IEEE, 1–4.

[4] Jie Chen, Li-hui Zou, Juan Zhang, and Li-hua Dou. 2009. The comparison and application of corner detection algorithms. *Journal of Multimedia* 4, 6 (2009).

[5] Wenjie Chen, Yangyang Ma, Zhilei Chai, Mingsong Chen, and Daojing He. 2017. An FPGA-based real-time moving object tracking approach. In *Algorithms and Architectures for Parallel Processing: 17th International Conference (ICA3PP 2017), Helsinki, Finland, August 21–23, 2017, Proceedings*. Springer, 65–80.

[6] Debian. 2023. Debian. Retrieved from https://www.debian.org/

[7] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. 2018. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3867–3876.

[8] Quentin Gautier, Alric Althoff, and Ryan Kastner. 2019. FPGA architectures for real-time dense SLAM. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP'19)*, Vol. 2160. IEEE, 83–90.
[9] Riccardo Giubilato, Sebastiano Chiodini, Marco Pertile, and Stefano Debei. 2019. An evaluation of ROS-compatible stereo visual SLAM methods on a nVidia Jetson TX2. *Measurement* 140 (2019), 161–170.
[10] Maria Rafaela Gkeka, Alexandros Patras, Christos D. Antonopoulos, Spyros Lalis, and Nikolaos Bellas. 2021. FPGA architectures for approximate dense SLAM computing. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE'21)*. IEEE, 828–833.
[11] Maria Rafaela Gkeka, Alexandros Patras, Nikolaos Tavoularis, Stylianos Piperakis, Emmanouil Hourdakis, Panos Trahanias, Christos D. Antonopoulos, Spyros Lalis, and Nikolaos Bellas. 2023. Reconfigurable system-on-chip architectures for robust visual SLAM on humanoid robots. *ACM Transactions on Embedded Computing Systems* 22, 2 (2023), 1–29.
[12] Javier Hidalgo-Carrió, Guillermo Gallego, and Davide Scaramuzza. 2022. Event-aided direct sparse odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5781–5790.
[13] Jingjin Huang, Guoqing Zhou, Xiang Zhou, and Rongting Zhang. 2018. A new FPGA architecture of FAST and BRIEF algorithm for on-board corner detection and matching. *Sensors* 18, 4 (2018), 1014.
[14] Inivation Inc. 2022. DAVIS 346. Retrieved from https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf
[15] Xilinx Inc. 2020. Petalinux. Retrieved from https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html
[16] Xilinx Inc. 2022. MPSoC PS and PL Ethernet Example Projects. Retrieved from https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/478937213/MPSoC+PS+and+PL+Ethernet+Example+Projects
[17] Xilinx Inc. 2022. Vivado HLS. Retrieved from https://docs.xilinx.com/v/u/en-US/dh0012-vivado-high-level-synthesis-hub
[18] Xilinx Inc. 2022. Zynq UltraScale+ MPSoC. Retrieved from https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[19] Hanme Kim, Stefan Leutenegger, and Andrew . Davison. 2016. Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VI 14*. Springer, 349–364.
[20] Akira Kojima. 2021. Autonomous driving system implemented on robot car using SoC FPGA. In *2021 International Conference on Field-Programmable Technology (ICFPT'21)*. IEEE, 1–4.
[21] Balint Barna Kövari and Emad Ebeid. 2021. MPDrone: FPGA-based platform for intelligent real-time autonomous drone operations. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR'21)*. IEEE, 71–76.
[22] Min Liu and Tobi Delbruck. 2017. Block-matching optical flow for dynamic vision sensors: Algorithm and FPGA implementation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS'17)*. IEEE, 1–4.
[23] Min Liu and Tobi Delbruck. 2022. EDFLOW: Event driven optical flow camera with keypoint detection and adaptive block matching. *IEEE Transactions on Circuits and Systems for Video Technology* 32, 9 (2022), 5776–5789.
[24] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. 2019. eSLAM: An energy-efficient accelerator for real-time ORB-SLAM on FPGA platform. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
[25] Bruce D. Lucas and Takeo Kanade. 1981. An iterative image registration technique with an application to stereo vision. In *7th International Joint Conference on Artificial Intelligence (IJCAI'81)* , Vol. 2. 674–679.
[26] Soumyadip Maity, Arindam Saha, and Brojeshwar Bhowmick. 2017. Edge SLAM: Edge points based monocular visual SLAM. In *2017 IEEE International Conference on Computer Vision Workshop (ICCVW)*. IEEE Computer Society, 2408–2417.
[27] Mohammad Moshref-Javadi and Matthias Winkenbach. 2021. Applications and research avenues for drone-based models in logistics: A classification and review. *Expert Systems with Applications* 177 (2021), 114854. https://doi.org/10.1016/j.eswa.2021.114854
[28] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. 2017. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *The International Journal of Robotics Research* 36, 2 (2017), 142–149.
[29] Raul Mur-Artal and Juan D. Tardós. 2017. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262.
[30] NVIDIA. 2022. Jetson TX2 Module. Retrieved from https://developer.nvidia.com/embedded/jetson-tx2
[31] NVIDIA. 2022. Tegra. Retrieved from https://developer.nvidia.com/tegra-hardware-sales-inquiries
[32] OpenAMP. 2022. OpenAMP Project. Retrieved from https://www.openampproject.org/
[33] OptiTrack. 2022. Motion Capture Cameras. Retrieved from https://optitrack.com/cameras/

[34] Tong Qin, Peiliang Li, and Shaojie Shen. 2018. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics* 34, 4 (2018), 1004–1020.
[35] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. 2017. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. (2017).
[36] Peter Sossalla, Johannes Hofer, Justus Rischke, Christian Vielhaus, Giang T. Nguyen, Martin Reisslein, and Frank H. P. Fitzek. 2022. DynNetSLAM: Dynamic visual SLAM network offloading. *IEEE Access* 10 (2022), 116014–116030.
[37] Keisuke Sugiura and Hiroki Matsutani. 2021. An FPGA acceleration and optimization techniques for 2D LiDAR SLAM algorithm. *IEICE Transactions on Information and Systems* 104, 6 (2021), 789–800.
[38] Jie Tang, Bo Yu, Shaoshan Liu, Zhe Zhang, Weikang Fang, and Yanjun Zhang. 2018. π-SoC: Heterogeneous SoC architecture for visual inertial SLAM applications. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'18)*. IEEE, 8302–8307.
[39] Yunchao Tang, Mingyou Chen, Chenglin Wang, Lufeng Luo, Jinhui Li, Guoping Lian, and Xiangjun Zou. 2020. Recognition and localization methods for vision-based fruit picking robots: A review. *Frontiers in Plant Science* 11 (2020), 510.
[40] Vibhakar Vemulapati and Deming Chen. 2022. ORB-based SLAM accelerator on SoC FPGA. *arXiv preprint arXiv:2207.08405* (2022).
[41] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. 2018. A C++ implementation of UltimateSLAM. Retrieved from https://github.com/uzh-rpg/rpg_ultimate_slam_open (2018).
[42] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. 2018. Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters* 3, 2 (2018), 994–1001.
[43] Xiaohua Wang, Vivek Yadav, and S. N. Balakrishnan. 2007. Cooperative UAV formation flying with obstacle/collision avoidance. *IEEE Transactions on Control Systems Technology* 15, 4 (2007), 672–679.
[44] Xenomai. 2023. Xenomai. Retrieved from https://www.xenomai.org/
[45] Jingao Xu, Hao Cao, Danyang Li, Kehong Huang, Chen Qian, Longfei Shangguan, and Zheng Yang. 2020. Edge assisted mobile semantic visual SLAM. In *IEEE INFOCOM 2020—IEEE Conference on Computer Communications*. IEEE, 1828–1837.
[46] Yi Zhou, Guillermo Gallego, and Shaojie Shen. 2021. Event-based stereo visual odometry. *IEEE Transactions on Robotics* 37, 5 (2021), 1433–1450.

Received 8 September 2023; revised 7 May 2024; accepted 27 August 2024