Two-Way Street: Efficient VSLAM using Collaborative In-Sensor and Off-Sensor processing

Hongyi Zhang*[1], Laurie Bose*[1], Piotr Dudek[2], and Walterio Mayol-Cuevas[1,3]

[1] University of Bristol, Bristol, UK
[2] University of Manchester, Manchester, UK.
[3] Amazon.com, Seattle, USA

Abstract. We present a novel implementation of Visual Simultaneous Localisation and Mapping (VSLAM), whose computation is performed collaboratively between a sensor-based front-end utilizing efficient in-pixel compute, and a PC-based back-end that sends pose estimation back to the sensor to improve feature matching. The front-end performs the tasks of point-feature detection and tracking at up to 2000 Hz, while the PC back-end conducts the less immediate, higher level tasks, of mapping and localisation at $\approx$ 200 Hz. This division of tasks is somewhat reminiscent of biological vision, in which the retina performs low level computations, while the brain handles slower integrative functions. Only sparse data is exchanged between these two ends, such as feature locations, descriptor data, and sensor pose estimations. Essentially, each side only sends data which the other side requires to perform its respective task. This is a radical departure from the traditional visual pipeline, where dense image frames are streamed from sensor to PC. By comparison, our collaborative compute pipeline is extremely efficient, demonstrating a 10$\times$ improvement to speed and latency, $>$ 1000$\times$ bandwidth and $\sim$ 20$\times$ reduction in power consumption per processed frame.

Keywords: Pixel Processor Array · Feature Tracking · Visual SLAM

1 Introduction

Visual Simultaneous Localisation and Mapping (VSLAM) is a cornerstone technology in computer vision that enables a system to estimate its position and map its surroundings from visual sensor data [7]. The development of this technology has proved essential for autonomous robots, drones, and AR/VR headsets. Despite great advances in software and methodology, the vast majority of systems utilize traditional camera sensors. These necessitate the standard visual compute pipeline, whereby whole images must be streamed from the sensor to external processing, a costly process which requires significant time and energy.

In the particular case of point-feature based VSLAM, whole images are processed and distilled down to a sparse set of keypoints and associated descriptors.

* Equal contribution

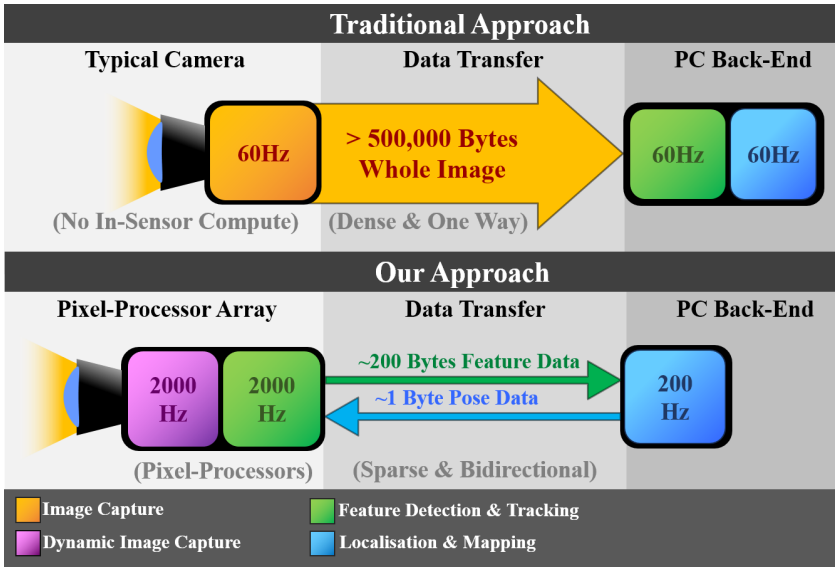

Fig. 1: Standard (top) vs our collaborative visual SLAM pipeline (bottom). We use a PPA sensor to directly detect and track point features in-sensor. The PPA then outputs only sparse feature data (locations and descriptors) to the PC back-end, not whole images. This massively reduces the cost of data transfer and decreases computational load per frame on the PC back-end. Additionally, the PC sends camera pose data to the sensor for better feature matching. All data transfer numbers are given per frame.

These are derived from a tiny subset of salient pixels within the image, while the vast majority of image pixels contain redundant information (as most pixels are correlated to their neighbours), or are within low contrast areas containing no salient detail. Despite this, all pixels are wastefully transferred from sensor to processor, only for the majority to then be discarded during processing. This extreme inefficiency cannot be addressed by software alone, but requires new forms of sensor hardware.

One such sensor paradigm is the Pixel-Processor-Array (PPA), a visual sensor where every light capturing element is itself a tiny pixel-processor. Such sensors consist of thousands of interconnected pixel-processors operating in parallel as a Single Instruction Multiple Data (SIMD) computer, similar to GPU compute. Light entering the sensor can be immediately processed, using "in-pixel" computation. This allows the extraction of sparse higher level information such as features, classifications, bounding boxes etc, within the sensor itself. The sensor can then directly output this sparse data. Such output is minuscule compared to the formation and transfer of whole images. By performing such computations locally, as close as possible to the point of light capture, PPAs can be used to massively reduce the total amount of data transfer across an entire system. As such PPAs can be used to perform certain applications at speeds and power efficiencies far beyond what is achievable with traditional camera sensors.

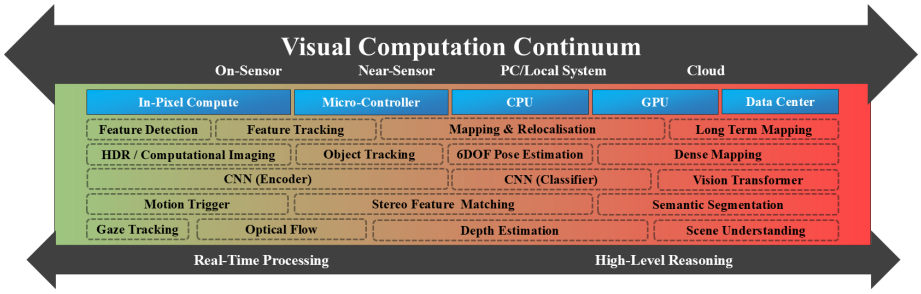

Fig. 2: Visual tasks can be accommodated throughout a *Visual Computation Continuum* where computation happens along the path from sensing to the back end and not only on the latter. A wealth of visual tasks could be performed on-sensor or near sensor bringing benefits in latency, bandwidth and energy efficiency. Arrows are bi-directional illustrating information could also flow from computation to influence sensing.

In this work we utilize a PPA as the front-end to our point-feature based monocular VSLAM implementation, illustrated in Figure 1. Point-feature detection, tracking and matching is performed entirely using the PPA's in-pixel computation at up to 2000 Hz. This high speed front-end outputs sparse feature location and descriptor data to the PC back-end which performs mapping and relocalisation. The back-end, derived from ORB SLAM 3 [5], uses its estimation of the sensor's pose to transmit feature orientation data back to the PPA based front-end. This extra information is utilized by the in-pixel compute to enable rotation-invariant descriptor generation, and point-feature tracking. This sparse, rapid, two-way communication enables both ends to collaboratively perform their respective tasks better than either could achieve in isolation. Crucially, this approach avoids the costly process of whole image transfer as with traditional cameras, and also avoids the complexity of processing abstract event stream data as with event cameras, instead processing standard image frames internally with in-pixel computation. This leads to a VSLAM pipeline demonstrating superior speed, efficiency and power consumption.

2 Visual Computation Continuum

Computer Vision has largely relied on a simplistic abstraction of visual processing: a one-directional pipeline separating sensing, which only collects and transports data, from computation, which handles interpretation and intelligence. This model has shaped artificial vision accordingly: cameras capture and transmit raw information while energy-hungry processors contend with overwhelming data volumes. Yet evidence from nature challenges this division. In [11], the frog's retina responds not merely to light but to specific shapes, sizes, contrasts, and, importantly, behaviours associated with prey. This computing is performed in the eye to enable low-latency responses while minimising unnecessary data transmission to the brain. This calls for a richer model of visual processing,

one where computation can happen at various stages if not all along the visual path, offering greater flexibility and efficiency. This approach we may call the *visual computation continuum* (Fig 2). When vision serves understanding or action rather than mere recording, tasks can be strategically distributed. Feature detection, selection, tracking, HDR, and event detection are natural candidates for the capture stage, reducing downstream processing load, bandwidth, and energy consumption. Object identification may occur closer to the sensor, while tasks requiring large data retention or cross-platform coordination may belong to the back-end. Interestingly, some capabilities, such as relocalisation, could be split: local relocalisation near the sensor for low latency, global relocalisation at the back-end where more data is available. Factored architectures such as CNNs are similarly amenable to distribution, with layers spread across sensor, interconnect, and back-end. Crucially, a computation continuum supports multi-directional information flow, allowing downstream stages to influence upstream processing. We exploit this opportunity in our paper, feeding visual pose estimates from the back-end to guide feature matching at the sensor stage. This two-way coordination between in-pixel processing and a visual SLAM back-end demonstrates how the continuum enables efficient, tightly coupled visual mapping and intelligence.

3 Related work

Prior PPA works. A small number of relevant PPA based works exist at the time of writing, all using either the SCAMP-5 or SCAMP-7 PPAs. Bose et al [1] proposed the first in-pixel visual odometry (VO) approach, where each new image frame is internally aligned against a key-frame stored across the pixel-processors. Sensor output then consists of only the transformation parameters of this image alignment (translation, scaling and rotation). This minimal data transfer allows operation at over 1000 Hz, but only an ambiguous 4 Degrees-Of-Freedom (4-DOF) motion estimation is produced.

Murai & Lisondra extended visual odometry on PPAs to full 6-DoF motion estimation with the point-feature based BIT-VO [15] and BIT-VIO [12]. These approaches use in-pixel compute to detect FAST keypoints and generate binary edge images. Both of these are transferred to a PC back-end where keypoint descriptors are generated from the binary image, and matched against descriptors of previous keypoints to estimate sensor motion. The main drawbacks are the bottleneck of data transfer related to the binary images and back-end image processing, limiting performance to 300 Hz operation.

Performing feature detection and matching entirely in-pixel, removing the need to transfer image data to PC, could avoid both these issues. Zhang et al's in-pixel work [20] conceptualised this idea, detecting FAST keypoints and generating BRIEF keypoint descriptors. Descriptors are stored across multiple pixel-processors, an unfortunate necessity on current PPA hardware where memory per pixel-processor is extremely limited. Keypoint descriptors from the latest frame can then be matched against these stored descriptors. This approach

avoids image transfer, but involves a great number of internal data transfers between pixel-processors due to the row-based descriptor storage necessitated by the limited memory, again limiting performance to 300 FPS. It is worth noting that non-feature based mapping was presented in the early work of [6], that used an image-based mapping and a probabilistic localisation filter which allowed for topological rather than spatial localisation.

Descriptor-In-Pixel. The previously described works, along with many traditional approaches, use the FAST algorithm [16] for keypoints detection. FAST uses no spatial or temporal priors (i.e. information from previous frames) to inform detection, treating each image as completely separate. This produces keypoints with very poor temporal stability, which regularly fail to be re-detected from one frame to the next. For video stream based tasks such as visual odometry where sequential images have very strong temporal correlation, such a naive approach is absurdly inefficient. This is especially true for PPAs, capable of operating at 1000s of frames per second, where each internally captured image is almost identical to the next.

Descriptor-In-Pixel (DIP) [2] is an in-pixel feature detection and tracking approach designed to exploit this high temporal correlation between image frames on PPA. DIP works by placing tiny point-feature descriptors inside each pixel-processor and computing the match between each processor's surrounding local image and this stored descriptor, generating a "descriptor response" within each pixel-processor. This then forms a "descriptor response map" spread across the array of pixel-processors. Key-points are then identified by local maxima occurring within this response map, with the pixel-processor containing such a local maxima giving both the key-point's location and descriptor.

By dynamically updating the layout of descriptors stored across the pixel-processor array each frame, this descriptor response mechanism can be used to both detect and track key-points. This is done by first spreading each existing key-point's descriptor into all nearby pixel-processors, essentially allocating them to the re-detection of that key-point. Any remaining pixel-processors that did not receive a descriptor this way are instead given a random (global) descriptor, allocating them to the task of detecting new keypoints. All pixel-processors then compute the response of their stored descriptors in parallel, and key-points, both old and new, are identified by locating local maxima within this descriptor response.

Thus, under this approach, there is no distinction between the computation performed for key-point detection and key-point tracking, allowing everything to be performed in parallel. This makes the approach extremely efficient, capable of operating at up to 3000 FPS, at which keypoints barely move between frames. Combined with its inherent re-detection, and the approach produces a stream of extremely robust keypoints. We implement a modified version of DIP for the point-feature detection and tracking front-end to our system.

Event Cameras. Event cameras are another type of alternative visual sensor, which do not produce images, but instead output sparse *event* data consisting of the coordinates and polarity of pixel intensity changes. These sensors pro-

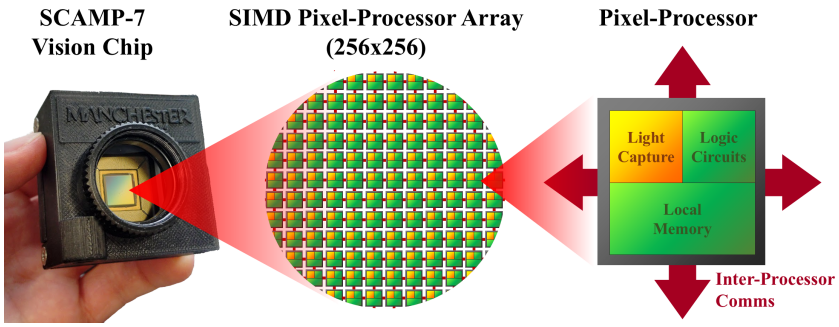

Fig. 3: Illustration of SCAMP Architecture. The sensor plane consists of 256 × 256 (65536) pixel processors (PEs), with all PEs executing SIMD Instructions provided by the ARM micro-controller.

vide very high temporal resolution, but bring additional complexity in interpreting their event data compared to processing image data. Indeed, many existing works [9, 10, 17–19] rely upon slow and computationally expensive methods to reconstruct images on the back-end from the event stream, followed by more traditional image processing. Other works utilize dedicated hardware such as spiking neural network accelerators, to process the event stream directly. The increased complexity involved with handling event data compared to traditional images is the price event cameras pay for achieving their extreme temporal resolution. In this respect PPAs and the concept of in-pixel processing provide an arguably more effective solution for the majority of computer vision applications, simply avoiding the image transfer bottleneck by processing images internally and outputting sparse extracted information.

4 SCAMP7 Architecture

At the time of writing there are very few sensors with true "in-pixel" computing capability, with the only fully programmable PPAs being the SCAMP (Switched Current Analogue Matrix Processor) systems developed by Dudek and Carey [8]. In this work we utilize the SCAMP-7 [3] system for our PPA based front-end, with an array size of 256×256, giving a total of 65,536 pixel-processors (see Figure 3) . These processors do not locally store program instructions, but execute instructions broadcast from a centralised controller, with all 65,536 processors then executing the same instructions concurrently. However, a 1-bit “activity flag" within each processor can be used to accept or ignore instructions, enabling conditional execution at the per pixel level. Thus SCAMP-7 PPA operates as a Single Instruction Multiple Data (SIMD) computer, and by programming the controller’s issue of instructions, entire vision tasks can be executed directly within the pixel-processor array.

The pixel-processors of SCAMP are mixed-signal, capable of performing both analogue and digital processing, but have very limited local resources. Each pro-

cessor has 23 bits of digital memory, and 7 analogue registers (continuous values). Importantly each is connected to its four immediate neighbours, which enables data transfer across the entire array via multiple hops. In terms of light capture, every processor separately captures one pixel, together forming a 256×256 image spread across the array. Each processor's light capturing element accumulates charge from light exposure, which when instructed, can then be readout into an analogue register for processing or reset. In this way, each pixel-processor is able to individually control its time window and length of exposure, enabling sophisticated, novel image capture modalities such as demonstrated in [13].

5 VSLAM Approach Overview

Our implementation of visual SLAM splits the computation of various tasks between a PPA sensor front-end and a PC back-end. These two sides communicate bi-directionally, transferring sparse data related to feature locations, descriptors and pose information. The PPA front-end uses massively parallel in-pixel compute to rapidly process raw pixel-data, performing point-feature detection, tracking and matching. The PC back-end then performs the slower, more abstract, high-level tasks of localisation and mapping. Both ends can run at different speeds appropriate to their respective tasks. While the front-end operates up to 2000 Hz to ensure point-feature data provided to the back-end is highly robust to rapid motion, the back-end performs mapping at a slower speed. Figure 1 shows an overview of our system.

6 PPA Front-End

Our system's front-end, implemented upon the SCAMP-7 PPA, performs point-feature detection, tracking and matching entirely using in-pixel compute. In each frame, the sparse locations of these internally tracked features are transferred to the PC back-end, where they are used for pose tracking and localisation. Additionally the front-end performs piece-wise generation and transfer of longer (256-bit) BRIEF descriptors, used by the back-end's relocalisation and mapping.

6.1 Point-Feature Detection & Tracking

The PPA based front-end makes use of the Descriptor-In-Pixel [2] (DIP) point-feature detection and tracking algorithm, but with two new enhancements.

1. **Maximum Local Contrast Imaging** We incorporate a novel image capture algorithm into DIP, controlling exposure at the per pixel-processor level. Referred to as Maximum Local Contrast (MLC), this approach not only avoids overexposure, but ensures each area of the image is captured at the exposure under which details have maximum contrast. A comparison of images captured using fixed exposure, a reimplementation of Martel's HDR approach for SCAMP [14], and our MLC method is shown in Figure 4. Salient

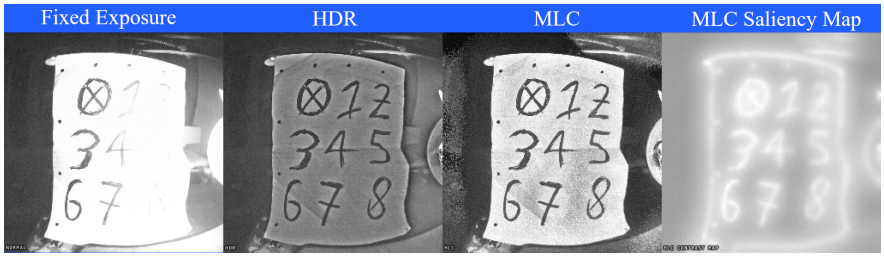

Fig. 4: Comparison between different approaches of internal image capture on the SCAMP PPA. From left to right, fixed exposure, in-pixel HDR imaging, and finally our proposed in-pixel MLC imaging with its saliency map. While HDR ensures over-exposure is avoided, unlike MLC it does not necessarily capture each pixel at the exposure at which detail is most clearly visible. By comparison, in areas with no local detail MLC produces a nosier result and the saliency map should be used to identify and avoid these areas during feature detection.

features will in general be more pronounced in MLC images, making the detection and tracking of point-features an easier task. A downside of MLC's approach is that areas of low-saliency (such as blank surfaces) produce noisy results. This is because the exposure containing the most high contrast detail in such areas cannot be precisely determined. However, this imaging methodology is specifically designed for computer vision not human consumption, and such low saliency areas generally contain no point-features. MLC as part of its processing, generates a saliency heatmap, which is utilized during feature detection to avoid such low saliency areas.

2. **Rotational Invariance** The original DIP feature tracking was not rotational invariant. This is largely due to the very limited resources per pixel-processor on current PPA hardware, where in-pixel descriptors are already limited to only 8 bits of memory. Our work solves this problem by streaming information from the PC back-end into the front-end's pixel-processors, specifically information pertaining to the SLAM system's estimated pose. This additional information is utilized by the pixel-processors to enable rotational invariant point-feature tracking, overcoming a shortcoming of the original DIP implementation. Our solution requires the use of rotationally symmetric sampling patterns for DIP's in-pixel descriptors. Performing left or right bit-shifting upon these descriptors is then equivalent to rotating it clock-wise or anti-clockwise by a fixed angle. Thus, as the orientation at which a point-feature would be observed at changes according to the front-end's pose estimation, its associated in-pixel descriptor can be bit shifted appropriately, allowing DIP to continue tracking its new rotated appearance.

6.2 Long Keypoint Descriptor Generation

In addition to performing point-feature detection and tracking, the PPA front end also generates long BRIEF descriptors for each of these point-features, utilized by the back-end for map building, as will now be further described.

1. **BRIEF Descriptor** The BRIEF keypoint descriptor [4] is constructed by comparing pairs of pixels, sampled relative to the keypoint's location, using a predetermined sampling pattern. Each comparison then generates 1 bit of the total descriptor. Specifically, the i-th bit of a descriptor is given by Equation 1, where p_1^i and p_2^i denote intensities of the i-th pair of pixels. In practice, such generation on PPA involves transferring pairs of analogue pixel data between pixel-processors, subtracting one from the other and examining the sign of the resulting value.

$$\text{descriptor } i^{th} \text{ bit} = \begin{cases} 1 & p_1^i > p_2^i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

2. **Piece-Wise Generation**
As shown by [20], generating full 256-bit keypoint descriptors in a single frame using in-pixel computation is prohibitively memory-intensive and computationally costly on current generation PPA hardware. Thus we adopt an approach in which these long descriptors are generated and streamed out of the PPA piece-wise, over multiple consecutive frames. This minimizes data transfer and computational cost per-frame. Every frame 2-bits for each keypoint's long descriptor are generated in parallel, and then transmitted to the PC back-end. This proceeds in a rolling manner, each frame generating the following 2-bits of the long descriptor before looping back to the start. These fragments of the 256-bit descriptors are accumulated on the PC back-end, with a complete descriptor for a point being formed 128 frames after its initial detection. Because our front-end operates at 2000 FPS, this process still happens very quickly in practice ($\approx 64ms$).
3. **Pose-Aided Rotational Invariant Generation** This piece-wise approach allows the visual appearance of point-features to change as their 256-bit descriptors are generated. As the BRIEF descriptor is not rotational invariant, this can become a significant issue in certain situations, particularly sensor motion involving high rotation about the roll axis. To mitigate this, the same pose information transmitted from the back-end into the PPA front-end which enables rotationally invariant feature tracking, is again used here to appropriately rotate the 256-bit BRIEF sampling pattern each frame, ensuring rotational invariance during generation, as illustrated in Figure 5.

7 SLAM Back-End

Our SLAM back-end is derived from the well-known ORB SLAM 3 system [5], from which we remove the original image processing and feature matching

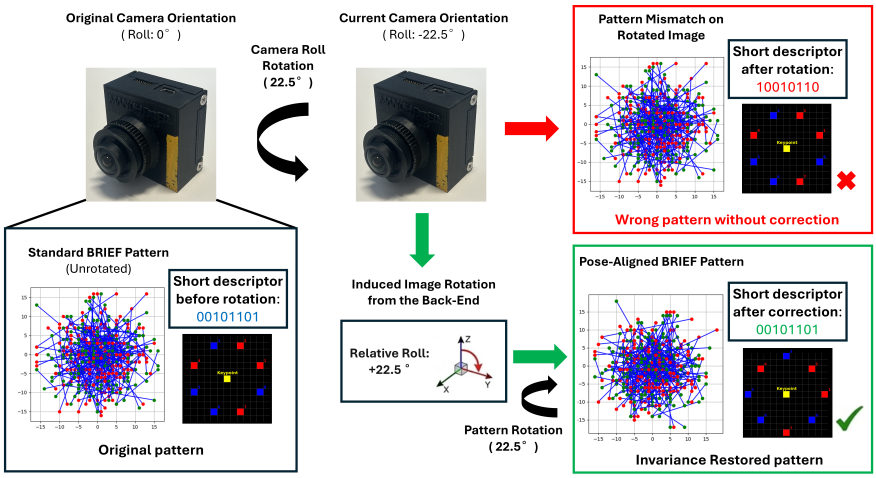

Fig. 5: Illustration of pose-aided rotational invariance. From left to middle, camera rotation induces a corresponding rotation of the image. From middle to right, the BRIEF sampling pattern and short descriptor are rotated according to the estimated pose, aligning it with the rotated image patch for descriptor consistency.

pipeline, replacing it with a tight integration to the DIP feature tracker running upon the SCAMP-7 PPA using in-pixel compute. This fundamental change to the visual pipeline also required us to make various changes to ORB SLAM 3's implementation such as novel rules for keypoint correspondence and gradual descriptor updating, the most important of which are described in this section.

Bidirectional Communication Protocol Rapid two-way communication of sparse data is performed between the front-end DIP feature tracker and back-end ORB SLAM 3 system using a dedicated thread. Each frame, the front-end sends (i) the 2D position of the tracked point in the image and (ii) a 2-bit piece of the long descriptor. In turn, the SLAM system transmits the estimated camera pose back to the PPA whenever a large pose change is detected.

Keypoint Association In the original tracking module of ORB SLAM 3, associations between keypoints across frames are only determined by matching 256-bit ORB descriptors between frames. This was required as the motion between frames can be significant with a traditional camera, however our front-end DIP tracker operates at up to 2000 Hz using the SCAMP-7 PPA. Such high speeds of operation make associating keypoints across frames a trivial task, as features typically change location only one or two pixels between frames. As such the computationally expensive descriptor matching based method of association used by ORB SLAM 3 can largely be replaced by simple positional based tracking. Descriptor matching is then only used as a fallback method for trying to associate features where location based matching fails, such as cases where a feature might disappear due to occlusion, then reappear at some later point in time.

Incremental Descriptor Update Since the PPA computes only 2-bit pieces of each feature's 256-bit BRIEF descriptor per frame, the back-end must monitor each keypoint and incrementally update its descriptor appropriately, accumulating these 2-bit pieces into the full 256-bit descriptor.

8 Analysis & Experiments

In this section, we provide analysis and experiments performed to evaluate our VSLAM approach, using cooperative computation split across a sensor front-end and a PC back-end. We have not made any substantial changes to the mapping component of monocular ORB SLAM 3, and do not expect any improvements in large scale map construction or relocalisation. Instead, this work focuses on the benefits that can be achieved by incorporating a sensor based computational front-end into the visual pipeline, and performing all raw image processing with in-pixel processing, avoiding the transfer of images from sensor to PC. As will be described, this brings very significant improvements in terms of speed, power, and efficiency, which in turn improves the robustness of the SLAM system in challenging scenarios such as rapid sensor motion.

8.1 Back-End Computational Load

Our approach performs the tasks of point-feature detection and tracking entirely within the sensor front-end, utilizing in-pixel computation. The PC back-end only needs to process sparse point-feature data output by this front-end, not whole images. Additionally the standard camera based ORB SLAM 3 implementation must extract and analyse many hundreds of FAST corner points each frame, in-order to find sufficient matches with existing point-features. This high number is necessary due to the poor robustness and temporal stability of FAST, but results in a high proportion of "junk" points being processed and simply discarded each frame. By comparison, the Descriptor-In-Pixel based point-feature detection and tracking used in this work provides a comparatively small, but extremely robust set of point-features. Both these factors result in a very significant reduction in computation required to be performed per frame by the PC back-end as shown in Table 1, in turn allowing the back-end to operate at a higher frequency.

8.2 Power Consumption

SCAMP-7 is an academic prototype, manufactured using an older 180nm CMOS process (compared to modern silicon manufacturing $<$10nm), which makes the device rather power hungry by current day standards. Despite this, the radical changes it can bring to the visual pipeline still result in significant benefits to overall power consumption, and point to even more significant improvements with more mature hardware development. We demonstrated full-system energy savings by measuring the power cost of the original ORB SLAM 3's tracking

Approach	Tracking Thread		Mapping Thread	
	Time(μs)	%	Time(μs)	%
Camera Based	$\approx 14,834$	100	$\approx 87,927$	100
PPA Based	$\approx 1,554$	10.4	$\approx 6,856$	7.8

Table 1: Comparison of average frame processing times of ORB SLAM 3's Tracking and Mapping threads performed on PC back-end. Our PPA based approach removes the computational cost of point-feature tracking and detection, while also providing the back-end with fewer (but much higher quality) point-features.

component (using a RealSense camera at 60 FPS), to our modified implementation's tracking component running at 2000 FPS utilizing the SCAMP-7 PPA. We measure the power consumption of each tracking component implementation by running it upon a fully charged Linux laptop, placed upon on a rotating turn table to provide changing visual input, and recording the time taken for half of the battery charge to be consumed. This is repeated for 3 independent runs for both implementations. Additionally, the baseline energy consumption of the Linux laptop in an idle state is measured and subtracted from the energy consumption of both systems for better relative comparison. Results of these experiments, given in Table 2, show that while the energy cost per second is similar between implementations (as expected), the energy cost per frame is greatly reduced with our PPA based approach with an over 20× reduction.

Device	Energy Cost Per Second(%)	Energy Cost Per Frame(%)
PPA	$(2.34 \pm 0.07) \times 10^{-2}$	$(1.9 \pm 0.1) \times 10^{-5}$
RealSense	$(2.35 \pm 0.08) \times 10^{-2}$	$(4.53 \pm 0.14) \times 10^{-4}$

Table 2: Average energy consumption (in percentage) of the PPA module and the Intel RealSense D455 sensor, measured via onboard laptop battery discharge profiling. Values are shown as mean ± standard deviation over three independent runs.

8.3 Data Transfer & Bandwidth Usage

As shown in Table 3, the conventional pipeline using RealSense transmits over 500,000 bytes per frame. In contrast, our proposed pipeline requires only about 200 bytes of data exchange between the PPA and the PC per frame (assuming around ≈ 100 point-features are tracked), resulting in a vast $> 1000\times$ reduction in data transfer per frame, effectively removing bandwidth as a bottleneck to the system's performance.

8.4 High-speed Tracking, Mapping and Loop Closure Detection

The very high speed point-feature detection and tracking performed by our sensor based front-end enables our approach to remain tracking even under sensor

Direction	PPA	RealSense
Sensor to Back-End	**Feature Locations** 2 Bytes per Feature	**Whole Image** $>$ 500,000 Bytes
Sensor to Back-End	**Feature Descriptor Data** 2 Bytes per Feature	- -
Back-End to Sensor	**Sensor Orientation Data** 1 Byte	-
Total (Per Frame)	**≈200 Bytes**	$>$ **500,000 Bytes**

Table 3: Comparison of the bandwidth used per frame by our PPA based approach and a traditional camera-based approach.

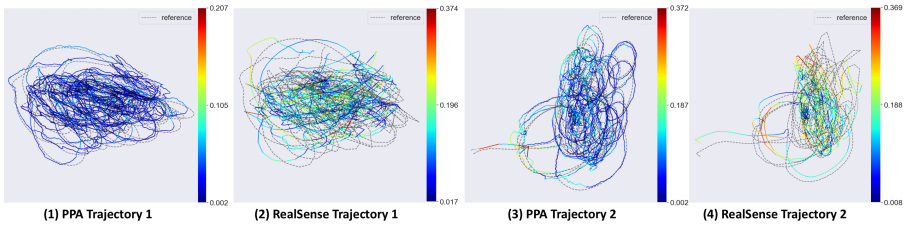

Fig. 6: Visualization of the Absolute Pose Error (APE) in translation for two representative tracking trajectories, evaluated separately using PPA and RealSense.

rapid motion. To compare with the original camera based ORB SLAM 3 implementation we mount an Intel RealSense D455 alongside the SCAMP-7 PPA, then put both under the same aggressive motion. We then record when tracking is lost by either of the SLAM systems. While our implementation utilizing PPA runs in real time, it was necessary to run ORB SLAM 3 offline, using recorded video from the RealSense, allowing it sufficient processing time for frame-processing and loop-closure. We found that real-time operation of ORB SLAM 3 was barely functional under these trajectories.

Figure 6 shows the estimated trajectories from both our proposed approach and ORB SLAM 3, for two of these violent motion sequences. During these sequences, the camera based ORB SLAM 3 frequently loses tracking as illustrated in Figure 7. This is captured in Table 4, showing the number of times tracking is lost by each approach. From this it is clear the PPA system is significantly more robust under such motion, providing more stable and continuous tracking.

Additionally for each of these aggressive motion trajectories, we record ground-truth trajectory using an OptiTrack motion capture system. we then attempt to align the (scaleless) estimated trajectories produced by both SLAM systems to this ground-truth for a rough measurement of Absolute Pose Error(APE) and Relative Pose Error(RPE) as shown in Table 5.

In Figure 8, we present two representative maps, one from an indoor laboratory environment and the other from an outdoor environment, to demonstrate our system's mapping and loop closure capabilities. Each red trajectory shows the systems map before loop closure, demonstrating the accumulated drift. The green trajectories then show the result after loop closure, with this drift effectively corrected.

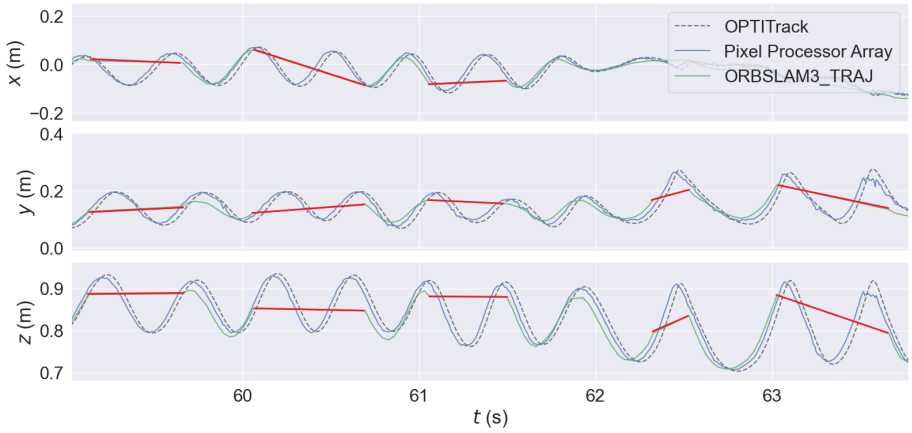

Fig. 7: Comparisons of estimated trajectories under aggressive sensor motion. Black dashed - OptiTrack ground-truth. Blue - original ORB SLAM 3 (RealSense camera). Green - Our approach utilizing the SCAMP-7 PPA. Loss of tracking is illustrated in red in this zoomed view, showing the frequent failures of the camera based approach.

Device	Times lost Tracking	Frames with no tracking(%)
PPA	2.2	1.44
RealSense	80.2	9.06

Table 4: Average number of re-localisation events (Times lost Tracking) and percentage of frames affected by tracking failures over multiple trajectories.

9 Conclusions

In this work we demonstrated the advantages that can be achieved for SLAM in moving away from the traditional visual pipeline. Instead of using a traditional camera sensor, in which whole images are transferred to a back-end for processing, we utilized a SCAMP-7 PPA capable of in-pixel processing. We adopted an approach which split the computation of tasks across this sensor front-end and a PC back-end. This enabled us to move all processing of raw images (for point-feature detection and tracking) into the pixel-processors of the sensor, and reduce sensor output to just sparse feature data used by the back-end for localisation and mapping. This not only eliminates the data transfer bottle-neck between sensor and processor, but the massively parallel in-pixel processing is able to provide significantly more robust point-features at up to 2000 FPS. Additionally sensor pose estimation data, produced by the PC back-end is transmitted into the PPA, enabling rotational invariance for its feature tracking. This bidirectional communication between both sides enables each to enhance the performance of the other, despite only sparse data being transferred each frame. All this results in a visual pipeline that is significantly more efficient than traditional whole image transfer. Our approach demonstrates greater than 1000× reduction

		RPE			APE		
		median	rmse	std	median	rmse	std
Translation(m)	PPA	**0.008**	**0.013**	**0.009**	**0.037**	**0.046**	**0.022**
	RealSense	0.044	0.069	0.043	0.099	0.113	0.047
Rotation(°)	PPA	**0.625**	**1.089**	**0.739**	**4.579**	**5.485**	**2.345**
	RealSense	2.529	4.902	3.446	7.008	8.002	3.110

Table 5: Average relative pose error (RPE) and absolute pose error (APE) of PPA and RealSense under rapid-motion sequences, evaluated over multiple trajectories and compared against ground-truth poses obtained from OptiTrack.

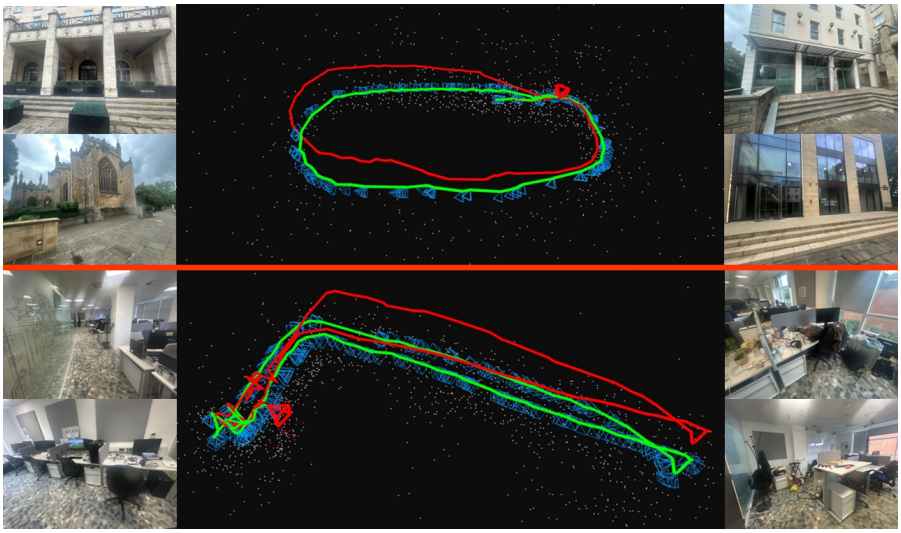

Fig. 8: The top image shows a trajectory in a 30m × 20m outdoor environment, whereas the bottom row depicts a trajectory in an indoor office environment. The red curve represents the historical camera trajectory before loop closure, highlighting the accumulated drift. The green curve illustrates the correct keyframes' position after loop closure.

in data transfer, and 20× reduction in energy consumption per frame. All while significantly reducing the computation load upon the PC back-end, and being able to reliably operate under aggressive rapid sensor motion which cannot be handled by standard camera based methods, which instead must fall back to rely on additional sensing such as IMU measurements. Beyond our work in this paper, we see a wider opportunity within the concept of the *visual computation continuum*, where a variety of visual competences can be deployed efficiently on novel architectures by asking the question of what should happen, where?

References

1. Bose, L., Chen, J., Carey, S.J., Dudek, P., Mayol-Cuevas, W.: Visual odometry for pixel processor arrays. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (Oct 2017) 4
2. Bose, L., Chen, J., Dudek, P.: Descriptor-in-pixel : Point-feature tracking for pixel processor arrays. 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp. 5392–5400 (2025), https://api.semanticscholar.org/CorpusID:280016551 5, 7
3. Bose, L., Dudek, P., Carey, S.J., Chen, J.: Live demonstration: Scamp-7. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3995–3996 (2023) 6
4. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: Brief: Binary robust independent elementary features. In: European conference on computer vision. pp. 778–792. Springer (2010) 9
5. Campos, C., Elvira, R., Rodr'iguez, J.J.G., Montiel, J.M.M., Tardós, J.D.: Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. IEEE Transactions on Robotics **37**, 1874–1890 (2020), https://api.semanticscholar.org/CorpusID:220713377 3, 9
6. Castillo-Elizalde, H., Liu, Y., Bose, L., Mayol-Cuevas, W.: Weighted node mapping and localisation on a pixel processor array. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). pp. 6702–6708 (2021) 5
7. Davison, A.J.: Mobile robot navigation using active vision (1998), https://api.semanticscholar.org/CorpusID:17502747 1
8. Dudek, P., Carey, S.: General-purpose 128× 128 simd processor array with integrated image sensor. Electronics Letters **42**(12), 678–679 (2006) 6
9. Duwek, H.C., Shalumov, A., Tsur, E.E.: Image reconstruction from neuromorphic event cameras using laplacian-prediction and poisson integration with spiking and artificial neural networks. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) pp. 1333–1341 (2021), https://api.semanticscholar.org/CorpusID:235692647 6
10. Kim, H., Leutenegger, S., Davison, A.J.: Real-time 3d reconstruction and 6-dof tracking with an event camera. In: European Conference on Computer Vision (2016), https://api.semanticscholar.org/CorpusID:26324573 6
11. Lettvin, J.Y., Maturana, H.R., McCulloch, W.S., Pitts, W.H.: What the frog's eye tells the frog's brain. Proceedings of the IRE **47**(11), 1940–1951 (1959). https://doi.org/10.1109/JRPROC.1959.287207 3
12. Lisondra, M., Kim, J., Murai, R., Zareinia, K., Saeedi, S.: Visual inertial odometry using focal plane binary features (bit-vio). In: 2024 IEEE International Conference on Robotics and Automation (ICRA). pp. 1661–1668. IEEE (2024) 4
13. Martel, J.N., Mueller, L.K., Carey, S.J., Dudek, P., Wetzstein, G.: Neural sensors: Learning pixel exposures for hdr imaging and video compressive sensing with programmable sensors. IEEE transactions on pattern analysis and machine intelligence **42**(7), 1642–1653 (2020) 7
14. Martel, J.N., Müller, L.K., Carey, S.J., Dudek, P.: Parallel hdr tone mapping and auto-focus on a cellular processor array vision chip. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1430–1433. IEEE (2016) 7
15. Murai, R., Saeedi, S., Kelly, P.H.: Bit-vo: Visual odometry at 300 fps using binary features from the focal plane. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 8579–8586. IEEE (2020) 4

16. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: European conference on computer vision. pp. 430–443. Springer (2006) 5
17. Scheerlinck, C., Rebecq, H., Gehrig, D., Barnes, N., Mahony, R.E., Scaramuzza, D.: Fast image reconstruction with an event camera. 2020 IEEE Winter Conference on Applications of Computer Vision (WACV) pp. 156–163 (2020), https://api.semanticscholar.org/CorpusID:210886473 6
18. Tenzin, S., Rassau, A., Chai, D.: Application of event cameras and neuromorphic computing to vslam: A survey. Biomimetics **9** (2024), https://api.semanticscholar.org/CorpusID:271340036 6
19. Vidal, A.R., Rebecq, H., Horstschaefer, T., Scaramuzza, D.: Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios. IEEE Robotics and Automation Letters **3**, 994–1001 (2017), https://api.semanticscholar.org/CorpusID:3738244 6
20. Zhang, H., Bose, L., Chen, J., Dudek, P., Mayol-Cuevas, W.: Focal plane visual feature generation and matching on a pixel processor array. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 29031–29039 (October 2025) 4, 9