High-Throughput Event-Based Feature Detection and Tracking on an Embedded CPU

Ethan Elms, Yasir Latif, and Tat-Jun Chin

AI for Space Group, Adelaide University, Australia

Abstract. Event cameras provide asynchronous visual sensing, making them a compelling modality for agile edge AI systems. For practical deployment, event-based vision algorithms must sustain sufficiently high throughput to keep pace with the incoming event stream. Yet existing approaches often fail to do so, as they are rarely designed in close consideration of the underlying sensor and compute hardware. We propose a new event-based feature detection and tracking algorithm that operates at high throughput on an embedded CPU, without GPU acceleration. Our pipeline is motion-defined: it activates only when sufficient motion is detected and dynamically adapts its computational load to the observed motion. At its core is a novel slice-parallel formulation of event-based corner detection and tracking that operates directly on live event streams. The pipeline is enabled by our Motion-Defined Concurrent Slices architecture, which efficiently processes event data and maximizes utilization of available compute resources. On event streams generated by handheld motion in typical scenes, our method achieves throughput above the average event rate, supporting responsive event-based vision on the edge. Extensive benchmarking on standard datasets and live demonstrations confirms the effectiveness of the proposed method. [1]

Keywords: Event-based vision · Feature detection · Feature tracking

1 Introduction

Event sensors are bio-inspired sensors that offer high dynamic range and high temporal resolution sensing while consuming little power. Event-based vision aims to build perception capabilities based on event sensing, and many event-based vision algorithms have been developed for various tasks, *e.g.*, motion analysis, 3D reconstruction, image classification, object detection [4,10,12,22,34,43].

By enabling high-temporal-resolution perception, event-based vision is ideal for building agile and responsive intelligent systems, such as augmented/virtual reality headgear and unmanned aerial vehicles. To support such applications, the perception algorithm should satisfy two basic properties: high throughput (to be defined shortly), and be feasible on power- and compute-constrained devices.

The *throughput* of an event-based vision algorithm is the rate at which it can process event data. The *event rate* is the rate at which events are triggered

[1] Project page: `https://0thane.github.io/SPEEDTrack/`

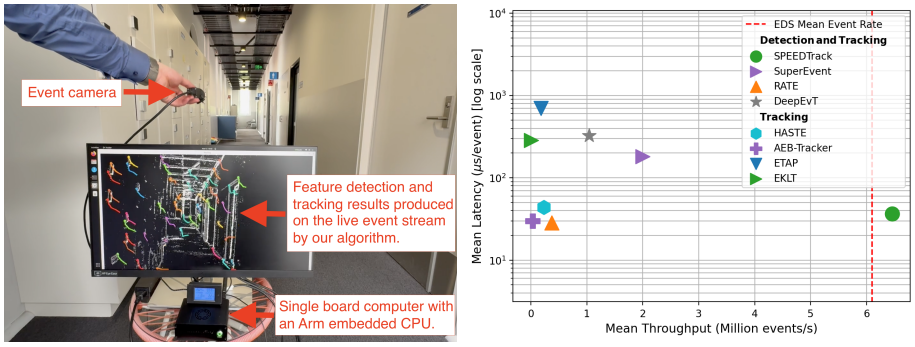

Fig. 1: L) Our high-throughput event-based feature detector and tracker (SPEEDTrack) can process live event streams on a low-power embedded CPU. See videos in the supp. material. **R)** Throughput and latency of common feature detection and/or tracking methods [1, 3, 11, 14, 20, 29, 42, 44] on the EDS dataset [16]. SPEEDTrack's throughput exceeds the mean event rate of the dataset and the throughput of the other methods. Moreover, SPEEDTrack's latency on the dataset is among the lowest.

in the sensor and output as an event stream. If the throughput is greater than the event rate, the algorithm can *keep processing* the *live* event stream without needing to sample the events[2]. This allows the algorithm to fully consider the information in the stream and potentially react to the dynamics therein.

Note that sampling events on-the-fly and in a manner that faithfully respects the dynamics in an event stream (to avoid under- or over-sampling) is nontrivial, since this often requires up-to-date knowledge of the dynamics of the stream. Thus, a competent event stream sampler also benefits from high throughput.

In this work, we focus on event-based feature detection and tracking (see Fig. 1**L**), which are fundamental vision tasks that support many downstream applications, including the perception stack of agile systems [41, 45]. While often combined in usage, feature detection and tracking are different tasks and separate algorithms for event-based feature detection [24, 31, 40] and tracking [11, 25, 30] have been developed, with some works conducting both [2, 35].

Fig. 1**R** compares the mean throughput of several state-of-the-art (SOTA) event-based feature detection and/or tracking algorithms [1,11,29] with the mean event rate of the EDS dataset [16]. Since the event rate outstrips the throughput, the algorithms will struggle to keep up with the event stream. Moreover, many SOTA methods employ deep learning and require GPU acceleration [26, 29, 38], which challenges the power budget of edge devices.

Earlier non-learning methods [2, 42, 44] leveraged concepts from the conventional vision literature, typically by producing intensity-like frames through event accumulation. If the event accumulation process is simple enough (*e.g.*, fixed cadence windows), the methods can potentially operate at high through-

[2] Note that the *instantaneous* event rate, which varies according to the scene complexity and camera motion, can be higher than the *instantaneous* throughput. However, if the average throughput over a time window W is higher than the average event rate over W, *all events* in W can be handled by the algorithm *within* W.

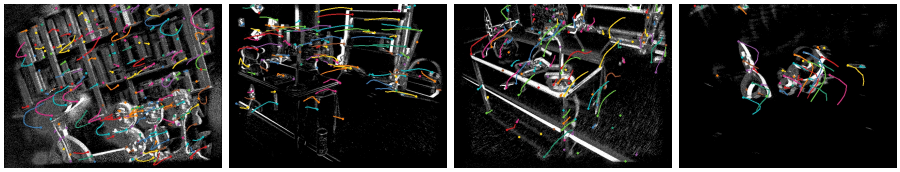

Fig. 2: Sample feature tracks from our method on various scenes (left to right: `peanuts dark`, `ziggy flying pieces`, `ziggy in the arena` and `rocket earth light` from the EDS dataset [16]).

put. However, basic event accumulation cannot adjust to highly variable motion speeds, leading to either blurring or over-sparsity in the event frames.

1.1 Contributions

We propose a Slice Parallel Embedded Event-based Feature Detector *and* Tracker (SPEEDTrack); see Fig. 1**L**. SPEEDTrack is a learning-free, event-only detection and tracking pipeline whose central contribution is a *system architecture* that aligns algorithmic structure with hardware characteristics to maximize resource utilization. At its core is the *Motion-Defined Concurrent Slice (MDCS)* architecture, which combines motion-adaptive event slicing with task-graph scheduling [18, 33]. Rather than introducing new detection or tracking primitives, SPEEDTrack co-optimizes established components so the full pipeline runs on an embedded CPU—without GPU acceleration—on live event streams; see Supplementary videos. This integration is non-trivial: MDCS scales near-linearly across CPU cores (Sec. 5.4), yielding throughput that substantially exceeds prior methods and can surpass the event rate of benchmark datasets; see Fig. 1**R**. Benchmark results (Fig. 2) show accuracy competitive with state-of-the-art learning-based methods at a fraction of the cost.

Emphasis of our work SPEEDTrack is a systems-oriented contribution aimed at maximizing throughput for live event streams on embedded CPUs while retaining competitive tracking accuracy. Results indicate a favorable accuracy–efficiency trade-off: a small reduction in accuracy yields a substantial runtime improvement, which is advantageous for power- and latency-constrained edge deployments. While SPEEDTrack cannot yet handle extreme event rates (*e.g.*, 100 M events/second), it achieves substantially higher throughput than prior methods on standard hardware.

On throughput and latency An algorithm's *latency* is the time between receiving an event and completing its processing. Reducing latency tends to raise throughput but is often bounded by hardware and task difficulty; when it cannot be reduced further, increasing *concurrency* raises throughput instead. SPEEDTrack exploits this through MDCS, maximizing concurrency—and thus throughput—while maintaining low latency. Fig. 1**R** confirms SPEEDTrack achieves low latency alongside high throughput.

2 Related work

Growing interest in event-based vision motivates development of 2D event-based feature detection and tracking algorithms. Here, we survey the relevant works.

2.1 Event-based feature detection

Event-based corner detection methods have evolved from classical adaptations to learned approaches. Early detectors fall into two groups: Harris-based and arc-based. eHarris [40] adapted the Harris corner detector to events but was limited by its computational cost, prompting faster alternatives such as eFAST [31] and ARC [2]. Later variants, including FA-Harris [24] and luvHarris [13], improved runtime performance. Learning-based methods have since emerged: Manderscheid *et al.* [28] used Speed-Invariant Time Surfaces with Random Forests, while Rebecq *et al.* [36] reconstructed intensity frames via RNNs to apply standard corner detectors. Chiberre *et al.* [6] further modeled image gradients with an RCNN. More recent works, EventPoint [19] and SuperEvent [3], learn event keypoints in a self-supervised or cross-modal manner.

2.2 Event-based feature tracking

Event-based feature tracking methods can be grouped by their underlying principle: template tracking, event alignment, spatio-temporal proximity or learning. Template-based methods such as HASTE [1] and RATE [20] maintain local event templates and search over 2D transformations to update feature positions, with RATE adding an event-corner detector and spatial distribution prior. While accurate, such methods are computationally heavy due to per-feature optimization.

Event alignment approaches directly register events in the spatio-temporal domain. ICP-based alignment [23,44] and spline-based trajectories [8,37] achieve high accuracy but at the cost of runtime and noise robustness. Proximity-based methods, *e.g.*, ARC* [2], eCDT [17], and ETC [9], instead exploit local event continuity via clustering or nearest-neighbor matching, trading robustness for speed. AEB [42] demonstrates microsecond-rate per-event updates through nearest-neighbor filtering of events with an EKF.

Learning-based tracking has recently emerged as a dominant paradigm. Chiberre *et al.* [7] predicted long-lived features directly from the event stream, and Messikommer *et al.* [29, 30] used feature networks with attention and stereo adaptation for state-of-the-art accuracy. Recent advances [3, 38] have pushed toward more general and efficient formulations. ETAP [14] achieves event-only tracking of any point via transformer-based attention and motion-invariant feature learning, while MATE [15] introduces motion-aware temporal consistency. For 3D tracking, E-3DTrack [25] jointly estimates 2D motion and 3D trajectories.

These trends reflect a clear shift toward motion-aware, asynchronous, and learning-based event tracking. Yet, most existing methods still depend on heavy neural backbones or serial event accumulation, making them unsuitable for deployment on resource-constrained platforms.

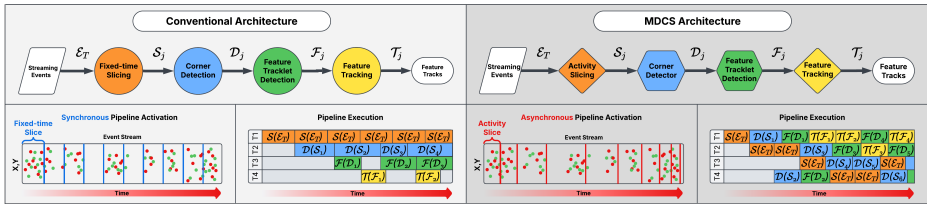

Fig. 3: Conventional vs MDCS architectures. The top row is the task graph of each method. Each tile in the task graph represents a logical task, with the arrows representing data flow and the shape of the tile denoting its concurrency: hexagonal tiles have unlimited concurrency *i.e.* they can be executed on any thread at any point in time; diamond titles have serial concurrency and can be executed on any thread in the order of their inputs; circle tiles are thread-bound and execute exclusively on a single thread. The bottom row contains a possible execution of the corresponding task graph with four available threads and the cadence of slices with respect to the event stream.

3 MDCS

Here, we describe our proposed MDCS architecture, which is the key innovation that enables high throughput performance on the edge.

To set the scene, let $\mathcal{E}_T = \{e_i\}_{i=1}^{N_T}$ denote the stream of events up to time T, where each event $e_i = (t_i, x_i, y_i, p_i)$ encodes pixel coordinates (x_i, y_i), timestamp t_i, and polarity $p_i \in \{0, 1\}$. As the sensor generates events asynchronously, N_T increases monotonically over time. A common forerunner to motion-defined processing is periodically cutting $\mathcal{E}_T$ into slices (a.k.a. batches, chunks)

$$\mathcal{E}_T \xrightarrow{\mathcal{S}} \{\mathcal{S}_j\},$$

using a slicing method $\mathcal{S}$, then processing the slices $\mathcal{S}_j$ that incrementally arrive. How an algorithm is structured to perform the above vitally affects its timeliness.

A basic slicing method that breaks $\mathcal{E}_T$ into fixed-size or fixed-duration slices wastes compute during static periods or misses fine temporal detail during rapid motion. Further, a basic scheduling approach that sends different processing functions to distinct threads requires complex inter-thread synchronization and cannot concurrently process multiple function inputs, which causes underutilization of compute resources. The left half of Fig. 3 depicts this approach.

In contrast, MDCS structures an event-based feature detector and tracker as motion-adaptive, self-regulating computation that maximizes CPU utilization while minimizing redundant processing. It does so through *motion-defined computation* and a *task graph scheduling*. Details are provided in the following, and the right half of Fig. 3 depicts MDCS.

Motion-defined computation MDCS employs a slicing subsystem (see Sec. 4.1 for the specific algorithm) that monitors event activity and spawns new slices only when sufficient motion is detected. Thus, slice frequency scales proportionally with motion magnitude, where dense motion with rich spatiotemporal content

yields frequent slices, while static scenes generate few or none. The motion-defined principle ensures computation occurs only when informative events are available, automatically reducing power and CPU load in low-activity periods.

Task graph scheduling We express event-based vision pipelines as computational task graphs [18], where nodes represent processing stages/functions and edges capture data dependencies; see the right half of Fig. 3. This abstraction exposes both serial and parallel relationships between components. In a task-graph, a runtime scheduler [33] dynamically assigns ready tasks to available CPU cores as soon as their dependencies are satisfied, enabling asynchronous and concurrent execution across the pipeline.

Combined with motion-defined computation, the task-graph approach implements a *motion-adaptive scheduling strategy* where compute resources are allocated on demand, driven by scene dynamics rather than a fixed frame rate.

At this stage, it is vital to state the impact of MDCS to our pipeline: in SPEEDTrack, each slice is processed independently to estimate local motion and features, after which results are temporally merged for global consistency. Because slices are self-contained, inter-thread synchronization is minimal and nearly all CPU time is devoted to information-rich parallel work. This design maintains near-continuous compute occupancy when motion is present and negligible utilization otherwise. Thus, SPEEDTrack scales efficiently across multi-core Arm CPUs and achieves high throughput without hardware acceleration.

4 SPEEDTrack

Building on MDCS, we introduce SPEEDTrack, a high-throughput event-based feature detection and tracking pipeline that unifies sensing, temporal partitioning, and feature tracking within a single asynchronous pipeline.

Unlike prior event-based feature detection and/or tracking systems that rely on fixed temporal windows or GPU-based flow estimation, SPEEDTrack dynamically regulates *when* and *how much* to compute based on observed motion activity. The pipeline comprises four tightly coupled stages:

1. An activity slicer that segments the event stream into motion-defined slices, each representing a self-consistent unit of spatio-temporal activity;
2. An event-only corner detector that operates independently within each slice using an ordinal-surface representation and an adaptive ignore mask for high-throughput feature detection;
3. A simple feature tracklet detector which detects features and produces short intra-chunk tracklets for each feature;
4. An asynchronous feature tracker that associates tracklets using a constant-velocity Kalman filter.

Conceptually, SPEEDTrack implements the mapping

$$\mathcal{E}_T \xrightarrow{\mathcal{S}} \{\mathcal{S}_j\} \xrightarrow{\mathcal{D}} \{\mathcal{D}_j\} \xrightarrow{\mathcal{F}} \{\mathcal{F}_j\} \xrightarrow{\mathcal{T}} \{\mathcal{T}_j\},$$

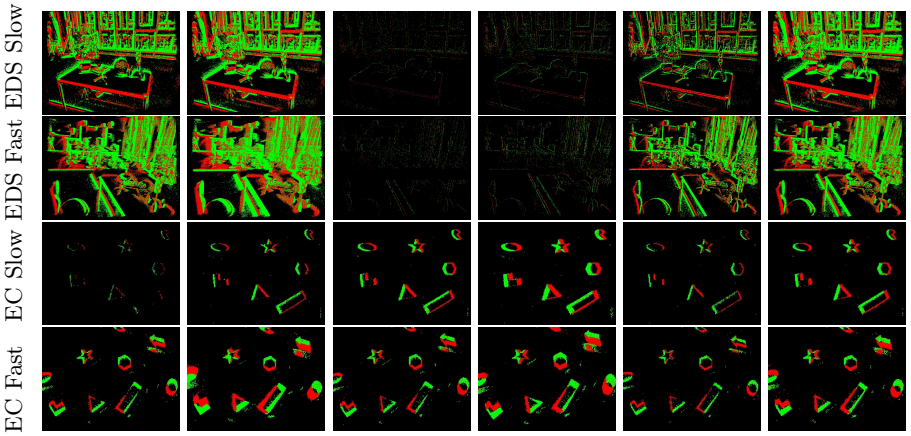

Fig. 4: Sample event frames comparing the quality of different slicing strategies under slow and fast motion on the `shapes_6dof` and `rocket earth light` scenes from the Event Camera [32] and EDS [16] datasets, respectively. We pick two common accumulation times (15ms and 30ms) for fixed-time batching (a & b) and two common event batch sizes (5,000 and 10,000) for fixed-size batching (c & d) to demonstrate that, for the same setting, each of these methods produce very different results across different motion speeds or scene complexity. On the other hand, our slicing method (e & f) tends to produce stable frames with consistent appearance for the same α_{th}.

where $\mathcal{S}$, $\mathcal{D}$, $\mathcal{F}$ and $\mathcal{T}$ denote activity slicing, corner detection, feature tracklet detection and feature tracking respectively. Details are provided in the following.

4.1 Activity slicing

When an instance of the activity slicer is active, it aims to include the incoming events into the latest motion-defined slice. As the slice $\mathcal{S}_j$ is progressively filled-in, we monitor the activity score α of the slice

$$\alpha = \frac{|\mathcal{S}_j|}{A_{\text{active}}}, \quad A_{\text{active}} = |\{(x, y) \mid \mathbf{A}(x, y) = 1\}|, \tag{1}$$

where $\mathbf{A}(x, y) = 1$ if pixel (x, y) has fired within $\mathcal{S}_j$. Put simply, α is the mean number of events per active pixel. Events are included into $\mathcal{S}_j$ until α exceeds a threshold α_{th}, after which $\mathcal{S}_j$ is finalized and a new one begins.

Intuitively, α captures how densely events are concentrated in active regions of the sensor: a higher α indicates more motion (and thus greater event overlap), while a lower α corresponds to sharper and more temporally localized event distributions. For efficiency, α is re-evaluated every δ_α, which is a sensor-aligned quantity, *e.g.*, δ_α is 100 times the refractory period of the sensor.

While also dependent on a tuned parameter α_{th}, our activity slicer is qualitatively different from basic fixed-time and fixed-size batching. As shown in Fig. 4, the same α_{th} value in our slicer produces slices with consistent appearance across

an event sequence, whereas the other approaches yield very different appearances as the speed and scene structure vary, even under the same fixed-time or fixed-size settings. Our motion-adaptive slicing thus enables stable operation of the downstream tasks such as corner detection and tracking. Though the motion-adaptive idea is also employed by more advanced slicing methods [5, 27, 39], the simplicity of our slicer benefits high-throughput operations.

4.2 Event-only corner detection

We adapt eHarris [31,40] to process events slice-wise, enabling independent, high-throughput corner detection. Two innovations enable embedded CPU execution:

- **Slice-local ordinal surfaces:** Instead of keeping a global per-pixel event queue, we construct an *ordinal surface* $\boldsymbol{\Sigma}_o(x, y)$ per slice, storing the index of the most recent event at each pixel. Binary patches are formed locally for Harris score computation, avoiding global dependencies.
- **Ignore mask:** An *ignore matrix* $\mathbf{V}(x, y)$ stores the current classification (corner, edge, or background) to skip redundant events and computations.

Together, these modifications approximate eHarris within each slice, retaining accuracy while achieving significantly higher throughput.

Harris score Let $\mathcal{S}_j$ be the slice currently being processed with events $e'_c = (t'_c, x'_c, y'_c, p'_c)$ ordered by time. The ordinal surface $\boldsymbol{\Sigma}_o(x'_c, y'_c)$ records the index of the latest event:

$$\boldsymbol{\Sigma}_o(x'_c, y'_c) = c. \tag{2}$$

Within an $L \times L$ window centered on (x'_c, y'_c), the event is not regarded as a corner if fewer than Θ nonzero entries exist. Otherwise, the Θ-th largest index S^* defines the binary patch $\boldsymbol{\Sigma}_b$:

$$\boldsymbol{\Sigma}_b(u, v) = \begin{cases} 1 & \text{if } \boldsymbol{\Sigma}_o(x'_c - \lfloor \frac{L}{2} \rfloor + u,\, y'_c - \lfloor \frac{L}{2} \rfloor + v) > S^*, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $u, v \in \{1, \ldots, L\}$. $\boldsymbol{\Sigma}_b$ is convolved with 5×5 Sobel kernels $\boldsymbol{G}_x, \boldsymbol{G}_y$ to obtain $\boldsymbol{I}_x = \boldsymbol{\Sigma}_b * \boldsymbol{G}_x$ and $\boldsymbol{I}_y = \boldsymbol{\Sigma}_b * \boldsymbol{G}_y$. The Harris matrix and score are computed as

$$\mathbf{M}(e'_c) = \sum_{p \in W} \mathbf{g}(p) \nabla \boldsymbol{I}(p) \nabla \boldsymbol{I}^\top(p), \tag{4}$$

$$H(e'_c) = \det[\mathbf{M}] - k \operatorname{tr}[\mathbf{M}]^2, \tag{5}$$

where $\mathbf{g}$ is a Gaussian kernel (with standard deviation $\sigma = 1$). An event is classified as a corner if $H(e'_c) > \bar{H}$.

Ignore mask To reduce redundant computation, an ignore matrix $\mathbf{V}(x, y)$ tracks the classification at each pixel:

$$\mathbf{V}(x'_c, y'_c) = \begin{cases} -1 & \text{if } H(e'_c) < 0, \\ 0 & \text{if } 0 \le H(e'_c) < \bar{H}, \\ 1 & \text{if } H(e'_c) \ge \bar{H}. \end{cases} \tag{6}$$

The Harris score of a subsequent event e'_c is computed only if $\mathbf{V}(x'_c, y'_c) = 0$; otherwise the event is ignored. This results in smoother spatio-temporal corner trajectories and reduced computational load for subsequent tracking.

4.3 Feature tracklet detector

Let $\mathcal{D}_j = \{e''_d\} \subset \mathcal{S}_j$ denote the set of detected corners within slice $\mathcal{S}_j$, where each $e''_d = (t''_d, x''_d, y''_d, p''_d)$. Since the activity slicer constrains the temporal extent of $\mathcal{D}_j$ based on motion magnitude, the motion of a corner within $\mathcal{D}_j$ can be approximated as locally linear in space and time:

$$x(t) = v_x t + b_x, \qquad y(t) = v_y t + b_y, \tag{7}$$

where (v_x, v_y) denotes the corner's velocity and (b_x, b_y) its spatial intercept. Each feature therefore follows a spatio-temporal line segment within $\mathcal{D}_j$.

To ensure stability and spatial diversity, the local spatial density of each detected corner (within radius r_{tracklet}) is recorded and non-maxima suppression is applied to remove redundant detections within r_{suppress}, retaining the top K corners with the highest local densities.

For the k-th selected corner, spatially neighboring events $\mathcal{M}^{(k)} = \{\tilde{x}_d^{(k)}, \tilde{y}_d^{(k)}, \tilde{t}_d^{(k)}\}$ within r_{tracklet} are used to fit the linear model via least squares:

$$\min_{v_x^{(k)}, b_x^{(k)}} \sum_d (\tilde{x}_d^{(k)} - v_x^{(k)} \tilde{t}_d^{(k)} - b_x^{(k)})^2, \tag{8}$$

$$\min_{v_y^{(k)}, b_y^{(k)}} \sum_d (\tilde{y}_d^{(k)} - v_y^{(k)} \tilde{t}_d^{(k)} - b_y^{(k)})^2. \tag{9}$$

This yields, for each $k \in \{1, \ldots, K\}$, a velocity vector $\mathbf{v}^{(k)} = (v_x^{(k)}, v_y^{(k)})$ that defines the feature's motion. Using $t_{\min}^{(k)} = \min\{\tilde{t}_d^{(k)}\}$ and $t_{\max}^{(k)} = \max\{\tilde{t}_d^{(k)}\}$, the corresponding endpoints are computed via the linear model,

$$\boldsymbol{\tau}_{\text{start}}^{(k)} = (x_{\min}^{(k)}, y_{\min}^{(k)}, t_{\min}^{(k)}), \quad \boldsymbol{\tau}_{\text{end}}^{(k)} = (x_{\max}^{(k)}, y_{\max}^{(k)}, t_{\max}^{(k)}), \tag{10}$$

which results in the tracklet

$$f_k = \langle \boldsymbol{\tau}_{\text{start}}^{(k)}, \boldsymbol{\tau}_{\text{end}}^{(k)} \rangle \tag{11}$$

that summarizes local spatial motion of features within $\mathcal{S}_j$.

In contrast to conventional trackers that maintain inter-slice dependencies, the proposed approach processes each slice independently. Density-based selection promotes re-detection of salient features across adjacent slices, allowing the subsequent tracking stage to establish temporal continuity without global synchronization. This design enables fully parallelized slice-local feature detection and tracking while preserving global consistency over time.

4.4 Feature tracking

We design a lightweight tracker that performs feature tracklet association using a per-feature constant-velocity Kalman Filter (KF) for feature position prediction, paired with a greedy track association mechanism. The motion of each active feature p is modeled via a constant-velocity state:

$$\mathbf{x}_p = [x,\ y,\ v_x,\ v_y]^\top, \quad \mathbf{P}_p \in \mathbb{R}^{4\times4}, \quad \Omega_p \in \mathbb{R},$$

where Ω_p is the timestamp of the last update of p. Given the current timestamp κ, $\Delta t = \kappa - \Omega_p$ and the time-dependent transition and process noise are:

$$\mathbf{F}_p(\Delta t) = \begin{bmatrix} \mathbf{I}_2 & \Delta t\mathbf{I}_2 \\ \mathbf{0}_2 & \mathbf{I}_2 \end{bmatrix}, \mathbf{Q}_p(\Delta t) = q \begin{bmatrix} \frac{\Delta t^4}{4}\mathbf{I}_2 & \frac{\Delta t^3}{2}\mathbf{I}_2 \\ \frac{\Delta t^3}{2}\mathbf{I}_2 & \Delta t^2\mathbf{I}_2 \end{bmatrix}$$

with $q = \sigma_a^2$. We observe only position, $\hat{\mathbf{x}}_p = \mathbf{H}\mathbf{x}_p + \mathbf{n}$ with

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_2 & \mathbf{0}_2 \end{bmatrix}, \quad \mathbf{R} = \sigma_m^2\mathbf{I}_2, \quad \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}).$$

Initialization A new track is spawned from a tracklet f_k:

$$\hat{\mathbf{x}}_p^+ = [x_{\max}^{(k)}, y_{\max}^{(k)}, \frac{x_{\max}^{(k)} - x_{\min}^{(k)}}{t_{\max}^{(k)} - t_{\min}^{(k)}}, \frac{y_{\max}^{(k)} - y_{\min}^{(k)}}{t_{\max}^{(k)} - t_{\min}^{(k)}}]^T,$$

$$\mathbf{P}_p^+ = \mathrm{diag}(\sigma_p^2, \sigma_p^2, \sigma_v^2, \sigma_v^2).$$

Prediction and update Features are predicted to the timestamp κ via:

$$\hat{\mathbf{x}}_p^-(\Delta t) = \mathbf{F}(\Delta t)\mathbf{x}, \quad \mathbf{P}_p^- = \mathbf{F}\mathbf{P}\mathbf{F}^\top + \mathbf{Q}.$$

(time-dependence omitted for brevity). Once a new tracklet f_k is associated to the feature p (described below), we construct the measurement $\mathbf{z} = [x_{\text{end}}, y_{\text{end}}]^T$ and then update

$$\mathbf{K}_p = \mathbf{P}_p^-\mathbf{H}^\top(\mathbf{H}\mathbf{P}_p^-\mathbf{H}^\top + \mathbf{R})^{-1}, \hat{\mathbf{x}}_p^+ = \hat{\mathbf{x}}_p^- + \mathbf{K}(\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}_p^-).$$

Track association and management Let $\mathcal{F}_j = \{f_k\}^K$ be the set of tracklets from the previous stage. With each new set of tracklets $\mathcal{F}_j$, we predict the current location of the feature p to the earliest timestamp of all tracklets $\kappa_{\min} = \min\{t_{\min}^{(k)}\}$ and use a KD-tree to collect a set of candidate matches within a fixed-radius r_{match}:

$$U_p = \{f_k \mid \|[x_{\min}^{(k)}, y_{\min}^{(k)}]^T - \mathbf{H}\hat{\mathbf{x}}_p^-(\kappa_{\min} - \Omega_p)\|_2^2 < r_{\text{match}}\}. \tag{12}$$

We then associate and update the state of the filter with the closest candidate feature from U_p:

$$\underset{f_k \in U_p}{\arg\min} \|[x_{\min}^{(k)}, y_{\min}^{(k)}]^T - \mathbf{H}\hat{\mathbf{x}}_p^-(t_{\min}^{(k)} - \Omega_p)\|_2^2. \tag{13}$$

To prevent duplicate tracks around existing ones, an active grid (marking $g \times g$ pixel cells containing active tracks) is maintained and unmatched tracklets within active cells are ignored. Tracks without new matches are removed, and new ones are spawned from unassigned tracklets sorted by motion magnitude $\|[x_{\max}^{(k)}, y_{\max}^{(k)}] - [x_{\min}^{(k)}, y_{\min}^{(k)}]\|$.

The tracking yields the set of active feature points $\mathcal{T}_j = \{\hat{\mathbf{x}}_p^-(\Omega_p)\}$ for the current slice.

5 Results

SPEEDTrack was implemented and evaluated on an NVIDIA Jetson Orin NX dev kit, however note that only the 8-core Arm Cortex A78AE v8.2 CPU and the 16 GB main memory were used. The live demos used a Prophesee EVK4 event camera, and the whole package consumed 12 W on average. Benchmarking of methods was conducted on a desktop with an Intel i5-8400 CPU and 32GB of RAM. Methods requiring a GPU [14, 30] utilized a NVIDIA RTX A6000. Qualitative results and visual comparisons with SOTA methods are available in the Supplementary Material.

Activity score threshold α_{th}	3
Binary patch size L	4
Non-zero ordinal surface values threshold Θ	25
Harris threshold $\bar{H}$	8
Feature detection suppression radius r_{suppress}	10
Max detected features K	200
KF position noise σ_p	1px
KF velocity noise σ_v	100px/s
Track association radius r_{match}	10
Active track suppression grid size g	20

Table 1: Hyperparameters of our SPEEDTrack method.

5.1 Parameter settings

SPEEDTrack depends on several hyperparameters; see Table 1 for their values. We emphasize that our method does not require extensive tuning; the same set of parameters were used for SPEEDTrack in all the benchmarking experiments and the live demonstrations.

5.2 Benchmarking of detectors (on desktop)

Detection quality To evaluate the quality of SPEEDTrack's detections, we followed the protocol of [28] and used the same ATIS Corner Dataset [28]. This

protocol measures corner-detection stability in a planar scene by associating corners with a simple nearest-neighbor strategy: two corners are considered matched if they lie within a predefined spatio-temporal distance. The reprojection error is then computed by estimating a homography from the matched corner sets extracted from two neighboring temporal windows of size δt, each centered at a different time step. Lower reprojection errors indicate more stable detections, as such points remain consistently matchable over time. Following [28], we also report the average association length as a measure of detection consistency.

For comparison, we include methods with publicly available implementations: eHarris [31], eFAST [31], and Arc [2]. Although SuperEvent [3] is an event-based feature detector, it does not classify events and instead outputs only image coordinates at a fixed time step; therefore, it is incompatible with this evaluation protocol. Instead, we compare against its tracking output in the next section.

As shown in Fig. 5**L**, our method produces corner detections that yield long association durations and low reprojection errors. This indicates that the detected corners remain stable over short time intervals. In addition, our method appears to outperform eHarris, of which it is a lightweight adaptation.

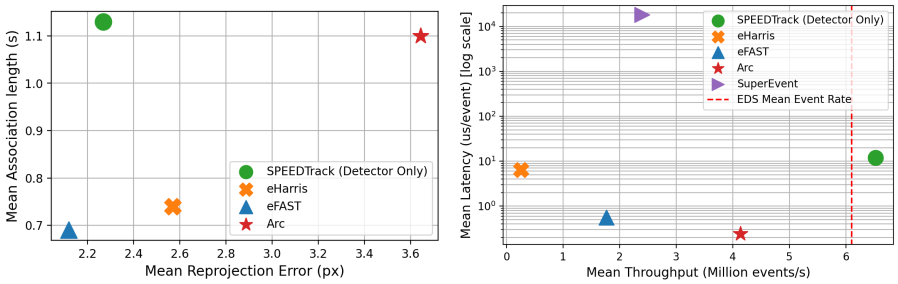

Fig. 5: L) Corner detection evaluation on the ATIS Corner Dataset [28] $\delta t = 25ms$, where the goal is reaching the top left corner. Our method's corners can be associated over a longer duration whilst maintaining a low reprojection error. **R)** Measured mean latency (log scale) and throughput of various detectors [2,3,31] on the EDS Dataset [16]; the goal is the bottom right corner. Our method manages high throughput without a significant increase in latency compared to other methods.

Runtime performance The main metrics used to evaluate runtime performance were throughput and runtime; see the Supplementary Material for a more detailed definition of the metrics. As shown in Fig. 5**R**, our corner detector saturated the mean event rate of the EDS Dataset and is thus able to keep up with the event stream. By comparison, existing event-only methods achieve substantially lower throughput and remain below the mean event rate. While SPEEDTrack does not attain the lowest latency, it offers a significant improvement in throughput for only a moderate increase in latency. Additional runtime analysis via the roofline model [21] is available in the Supplementary Material.

Table 2: The performance of the evaluated trackers on the EDS and EC scenes provided by [30] **with our detections**; the feature age of stable tracks (SFA) and the expected feature age (EFA) metrics are also from [30].

		EDS		EC	
Method	**Modality**	**SFA ↑**	**EFA ↑**	**SFA ↑**	**EFA ↑**
EKLT [11]	Events+Frames	0.230	0.150	0.795	0.760
DeepEvT [30]	Events+Frames	0.480	0.400	0.815	0.805
EM-ICP [44]	Events	0.115	0.090	0.320	0.310
HASTE [1]	Events	0.070	0.050	0.425	0.410
AEB-Tracker [42]	Events	0.300	0.290	0.540	0.460
DeepEvT* [29]	Events	0.515	<u>0.430</u>	0.785	<u>0.770</u>
ETAP [14]	Events	**0.742**	**0.613**	**0.870**	**0.851**
Ours	Events	<u>0.558</u>	0.421	<u>0.822</u>	0.713

Table 3: Pose estimation results on the EC [32] and EDS [16] datasets. Our method achieves the second-highest pose estimation accuracy on both datasets.

	EC: AUC (%)			EDS: AUC (%)		
Method	5° ↑	10° ↑	20° ↑	5° ↑	10° ↑	20° ↑
LLAK [7]	0.7	1.4	2.1	0.5	0.7	1.0
RATE [20]	3.3	8.4	18.0	2.1	5.1	10.3
EventPoint [19]	1.6	3.0	5.4	1.6	2.8	5.2
SuperEvent [3]	**22.7**	**35.8**	**46.7**	**15.2**	**26.4**	**40.1**
Ours	<u>14.4</u>	<u>20.8</u>	<u>27.2</u>	<u>10.9</u>	<u>14.0</u>	<u>17.5</u>

5.3 Benchmarking of trackers (on desktop)

To evaluate the performance of our feature tracker, we compared it against existing methods on the Event Camera (EC) [32] and Event-aided Direct Sparse Odometry (EDS) [16] datasets.

Tracking quality We adopted the evaluation protocol from [30], using the Stable Feature Age (SFA) and Expected Feature Age (EFA) metrics to assess the longevity and stability of feature tracks. As our tracker cannot be initialized with predefined detections, all methods were seeded with the detections from our approach rather than those from [30]. This ensured a fair comparison, as each tracker operates on the same set of initial feature locations. To isolate tracking performance, we initialized with only the features detected at the start of each sequence and disabled the initialization of new tracks and termination of existing ones during runtime. Table 2 summarizes the results on EDS and EC.

Our tracker consistently outperformed all traditional event-only methods and approached the accuracy of the strongest learning-based approaches, despite operating entirely without frames or GPU acceleration. On EDS, our method achieved an SFA of 0.558 and an EFA of 0.421, surpassing AEB-Tracker and approaching the performance of DeepEvT* [29]. On EC, it attained an SFA of 0.822, second only to ETAP [14], while maintaining competitive EFA performance. These results demonstrate that our lightweight, hardware-efficient design provides robust, long-lived feature tracking comparable to significantly more complex, hybrid, and deep event-based systems.

Pose estimation To assess geometric consistency, we adopted the relative pose estimation benchmark introduced in SuperEvent [3]. Pairs of ground-truth cam-

era poses within a two-second interval were selected, and feature correspondences between the two views were used to estimate the essential matrix with RANSAC. Performance was measured as the area under the curve (AUC) of the rotation-error accuracy, evaluated at thresholds of 5°, 10° and 20°; a higher AUC indicates more geometric consistency.

Table 3 reports results on both datasets. Our tracker achieved the second-highest overall performance, outperforming all classical and prior learning-based methods except SuperEvent [3]. These results confirm that our event-only features yield strong geometric consistency and accurate relative pose recovery while maintaining high-throughput performance on lightweight embedded hardware.

Runtime performance We compared the trackers with the same metrics as the corner detectors. The scenes of the EDS Dataset was used, as in the tracking quality evaluation. The benchmark measured only the mean latency and mean throughput of the tracking for feature tracking methods, as the detections are provided; whereas the feature detection and tracking methods encapsulate the latency and throughput of both detection and tracking. In terms of both latency and throughput (Fig. 1**R**), our method outperformed all others and achieved low-latency and high-throughput performance. Additional runtime analysis via the roofline model [21] is available in the Supplementary Material.

5.4 SPEEDTrack ablation studies (on embedded device)

We conducted an ablation study to quantify the effect of SPEEDTrack's key components. Specifically, we investigated the effects of (i) the activity slicer against a 15 ms fixed-time slicing strategy; (ii) using the Kalman filter for association versus a simple nearest-neighbors (NN) approach; (iii) ignoring corner detections that occur at the same pixel; and (iv) task-graph concurrency, as opposed to thread-based concurrency where each component runs on its own thread. Table 4 presents a comparison of the results.

It should be noted that replacing KF with NN reduces required computation, leading to a higher throughput and lower latency, but lower overall task performance. Similarly, thread-based concurrency leads to a computation bottleneck. While it does not affect the performance of the pipeline, it reduces throughput and increases latency. The complete SPEEDTrack pipeline achieved the best performance, validating the necessity of the design decisions and components.

5.5 SPEEDTrack live demonstrations (on embedded device)

The demos show that SPEEDTrack could keep up with the live event streams and adaptively adjust its computational load in response to variations in the instantaneous event rate. Note that each frame in the rendered demo videos includes the accumulated event frame from the most recent activity slice, thus, periods with little or no motion produce fewer tracking updates, as expected.

Table 4: Ablation study of SPEEDTrack, **conducted on the embedded device**.

Ablation	Latency (μs)	Thrput (Mev/s)	EDS		EC		EC: AUC (%)			EDS: AUC (%)		
			SFA ↑	EFA ↑	SFA ↑	EFA ↑	5° ↑	10° ↑	20° ↑	5° ↑	10° ↑	20° ↑
Activity slicing (fixed window)	71.2	4.01	0.401	0.298	0.658	0.541	8.9	13.7	18.4	6.1	9.0	11.9
Kalman filter (nearest tracklet)	**45.1**	**5.02**	0.438	0.330	**0.829**	0.672	10.2	15.4	20.2	7.2	10.5	13.5
Ignore mask (recompute H)	83.2	3.52	0.491	0.375	0.774	0.662	12.7	18.8	24.9	9.1	12.9	16.4
Task-graph (thread-based concurrency)	96.4	2.97	**0.558**	**0.421**	0.822	**0.713**	**14.4**	**20.8**	**27.2**	**10.9**	**14.0**	**17.5**
Baseline (SPEEDTrack)	49.7	4.92	**0.558**	**0.421**	0.822	**0.713**	**14.4**	**20.8**	**27.2**	**10.9**	**14.0**	**17.5**

6 Weaknesses

SPEEDTrack relies on the spatio-temporal smoothness of the event stream. In extremely dynamic scenarios, events might be dropped due to data-rate limitation of the USB interface, leading to degraded performance.

Since events are generated due to brightness changes, they do not necessarily represent sensor egomotion. Events triggered by light sources (AC frequencies) are indistinguishable from those by egomotion and may lead to phantom tracks.

Lastly, our method does not recover feature tracks once they are lost (occlusion or tracking failure). Recovery requires a SLAM map and is currently left as future work. Nevertheless, SPEEDTrack sustains live event-based feature detection and tracking on embedded hardware while keeping pace with the event stream, which few prior methods achieve under the same power and compute constraints.

7 Conclusions and future work

Through the proposed MDCS framework, SPEEDTrack was able to fully utilize computational resources to conduct high-throughput event-only feature detection and tracking. Our method could process live event streams using a standard CPU. Benchmark results show that SPEEDTrack's detection and tracking accuracies were comparable to previous methods. In future work, we plan on building upon SPEEDTrack to develop high-throughput event-based algorithms for 3D vision tasks.

Acknowledgements

Ethan Elms was partially supported by the Australian Institute for Machine Learning's Industrial AI program, funded by the South Australian Government through the Department of State Development's Research and Innovation Fund, and by the SmartSat CRC, whose activities are funded by the Australian Government's CRC Program.

References

1. Alzugaray, I., Chli, M.: HASTE: multi-hypothesis asynchronous speeded-up tracking of events. In: 31st British Machine Vision Conference 2020, BMVC 2020. BMVA Press (2020), https://www.bmvc2020-conference.com/assets/papers/0744.pdf
2. Alzugaray, I., Chli, M.: Asynchronous corner detection and tracking for event cameras in real time. IEEE Robotics and Automation Letters **3**(4), 3177–3184 (2018). https://doi.org/10.1109/LRA.2018.2849882
3. Burkhardt, Y., Schaefer, S., Leutenegger, S.: Superevent: Cross-modal learning of event-based keypoint detection for slam. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 8918–8928 (October 2025)
4. Chakravarthi, B., Verma, A.A., Daniilidis, K., Fermuller, C., Yang, Y.: Recent event camera innovations: A survey. In: ECCV. pp. 342–376. Springer (2024)
5. Chen, H., Wu, Q., Liang, Y., Gao, X., Wang, H.: Asynchronous tracking-by-detection on adaptive time surfaces for event-based object tracking. In: Proceedings of the 27th ACM International Conference on Multimedia. p. 473–481. MM '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3343031.3350975, https://doi.org/10.1145/3343031.3350975
6. Chiberre, P., Perot, E., Sironi, A., Lepetit, V.: Detecting stable keypoints from events through image gradient prediction. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 1387–1394 (2021). https://doi.org/10.1109/CVPRW53098.2021.00153
7. Chiberre, P., Perot, E., Sironi, A., Lepetit, V.: Long-lived accurate keypoints in event streams (2022)
8. Chui, J., Klenk, S., Cremers, D.: Event-based feature tracking in continuous time with sliding window optimization (2021), https://arxiv.org/abs/2107.04536
9. Elms, E., Latif, Y., Park, T.H., Chin, T.J.: Event-based structure-from-orbit. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 19541–19550 (2024)
10. Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A.J., Conradt, J., Daniilidis, K., et al.: Event-based vision: A survey. IEEE transactions on pattern analysis and machine intelligence **44**(1), 154–180 (2020)
11. Gehrig, D., Rebecq, H., Gallego, G., Scaramuzza, D.: EKLT: Asynchronous, photometric feature tracking using events and frames. Int. J. Comput. Vis. (2019)
12. Gehrig, M., Scaramuzza, D.: Recurrent vision transformers for object detection with event cameras. In: CVPR. pp. 13884–13893 (2023)
13. Glover, A., Dinale, A., Rosa, L.D.S., Bamford, S., Bartolozzi, C.: luvharris: A practical corner detector for event-cameras. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(12), 10087–10098 (2022). https://doi.org/10.1109/TPAMI.2021.3135635
14. Hamann, F., Gehrig, D., Febryanto, F., Daniilidis, K., Gallego, G.: ETAP: Event-based tracking of any point. In: IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR) (2025)
15. Han, H., Zhai, W., Cao, Y., Li, B., Zha, Z.j.: Mate: Motion-augmented temporal consistency for event-based point tracking pp. 8340–8349 (October 2025)
16. Hidalgo-Carrió, J., Gallego, G., Scaramuzza, D.: Event-aided direct sparse odometry. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2022)

17. Hu, S., Kim, Y., Lim, H., Lee, A.J., Myung, H.: ecdt: Event clustering for simultaneous feature detection and tracking. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3808–3815. IEEE (2022)
18. Huang, T.W., Lin, D.L., Lin, C.X., Lin, Y.: Taskflow: A lightweight parallel and heterogeneous task graph computing system. IEEE Transactions on Parallel and Distributed Systems **33**(6), 1303–1320 (2022). https://doi.org/10.1109/TPDS.2021.3104255
19. Huang, Z., Sun, L., Zhao, C., Li, S., Su, S.: Eventpoint: Self-supervised interest point detection and description for event-based camera. In: 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). pp. 5385–5394 (2023). https://doi.org/10.1109/WACV56688.2023.00536
20. Ikura, M., Gentil, C.L., Müller, M.G., Schuler, F., Yamashita, A., Stürzl, W.: Rate: Real-time asynchronous feature tracking with event cameras. In: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 11662–11669 (2024). https://doi.org/10.1109/IROS58592.2024.10802050
21. Ilic, A., Pratas, F., Sousa, L.: Cache-aware roofline model: Upgrading the loft. IEEE Computer Architecture Letters **13**(1), 21–24 (2014). https://doi.org/10.1109/L-CA.2013.6
22. Klenk, S., Motzet, M., Koestler, L., Cremers, D.: Deep event visual odometry. In: International Conference on 3D Vision, 3DV 2024, Davos, Switzerland, March 18-21, 2024. pp. 739–749. IEEE (2024)
23. Kueng, B., Mueggler, E., Gallego, G., Scaramuzza, D.: Low-latency visual odometry using event-based feature tracks. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 16–23 (2016). https://doi.org/10.1109/IROS.2016.7758089
24. Li, R., Shi, D., Zhang, Y., Li, K., Li, R.: Fa-harris: A fast and asynchronous corner detector for event cameras. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 6223–6229 (2019). https://doi.org/10.1109/IROS40897.2019.8968491
25. Li, S., Zhou, Z., Xue, Z., Li, Y., Du, S., Gao, Y.: 3d feature tracking via event camera. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18974–18983 (2024). https://doi.org/10.1109/CVPR52733.2024.01795
26. Li, S., Zhou, Z., Xue, Z., Li, Y., Du, S., Gao, Y.: 3d feature tracking via event camera. In: CVPR. pp. 18974–18983 (2024)
27. Liu, M., Delbruck, T.: Adaptive Time-Slice Block-Matching Optical Flow Algorithm for Dynamic Vision Sensors. BMVC (Sep 2018), http://bmvc2018.org/contents/papers/0280.pdf
28. Manderscheid, J., Sironi, A., Bourdis, N., Migliore, D., Lepetit, V.: Speed invariant time surface for learning to detect corner points with event-based cameras. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10237–10246 (2019). https://doi.org/10.1109/CVPR.2019.01049
29. Messikommer, N., Fang, C., Gehrig, M., Cioffi, G., Scaramuzza, D.: Data-driven feature tracking for event cameras with and without frames. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 1–12 (2025). https://doi.org/10.1109/TPAMI.2025.3536016
30. Messikommer, N., Fang, C., Gehrig, M., Scaramuzza, D.: Data-driven feature tracking for event cameras. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5642–5651 (2023). https://doi.org/10.1109/CVPR52729.2023.00546

31. Mueggler, E., Bartolozzi, C., Scaramuzza, D.: Fast event-based corner detection. In: British Machine Vision Conference (BMVC) (2017)
32. Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., Scaramuzza, D.: The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. The International Journal of Robotics Research **36**(2), 142–149 (Feb 2017). https://doi.org/10.1177/0278364917691115, http://dx.doi.org/10.1177/0278364917691115
33. Oneapi-Src: Oneapi-src/onetbb: Oneapi threading building blocks (onetbb), https://github.com/oneapi-src/oneTBB
34. Rebecq, H., Gallego, G., Mueggler, E., Scaramuzza, D.: EMVS: Event-based multi-view stereo—3D reconstruction with an event camera in real-time. Int. J. Comput. Vis. **126**, 1394–1414 (Dec 2018). https://doi.org/10.1007/s11263-017-1050-6
35. Rebecq, H., Horstschaefer, T., Gallego, G., Scaramuzza, D.: Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. IEEE Robotics and Automation Letters **2**(2), 593–600 (2017). https://doi.org/10.1109/LRA.2016.2645143
36. Rebecq, H., Ranftl, R., Koltun, V., Scaramuzza, D.: Events-to-video: Bringing modern computer vision to event cameras. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3852–3861 (2019). https://doi.org/10.1109/CVPR.2019.00398
37. Seok, H., Lim, J.: Robust feature tracking in dvs event stream using bézier mapping. In: 2020 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1647–1656 (2020). https://doi.org/10.1109/WACV45572.2020.9093607
38. Shen, Y., Li, Y., Chen, S., Li, G., Huang, Z., Bao, H., Cui, Z., Zhang, G.: Blinktrack: Feature tracking over 80 fps via events and images. In: CVPR. pp. 9298–9308 (2025)
39. Valerdi, J.L., Bartolozzi, C., Glover, A.: Insights into batch selection for event-camera motion estimation. Sensors **23**(7) (2023). https://doi.org/10.3390/s23073699, https://www.mdpi.com/1424-8220/23/7/3699
40. Vasco, V., Glover, A., Bartolozzi, C.: Fast event-based harris corner detection exploiting the advantages of event-driven cameras. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4144–4149 (2016). https://doi.org/10.1109/IROS.2016.7759610
41. 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**(2), 994–1001 (2018)
42. Wang, Z., Molloy, T., van Goor, P., Mahony, R.: Asynchronous blob tracker for event cameras. IEEE Transactions on Robotics **40**, 4750–4767 (2024). https://doi.org/10.1109/TRO.2024.3454410
43. Zhou, X., Bei, C.: Backlight and dim space object detection based on a novel event camera. PeerJ Computer Science **10**, e2192 (2024)
44. Zhu, A.Z., Atanasov, N., Daniilidis, K.: Event-based feature tracking with probabilistic data association. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 4465–4470 (2017). https://doi.org/10.1109/ICRA.2017.7989517
45. Zhu, A.Z., Atanasov, N., Daniilidis, K.: Event-based visual inertial odometry. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5816–5824 (2017). https://doi.org/10.1109/CVPR.2017.616