

Real-Time Multi-Particle Tracking at 10 kHz: Leveraging Binary Frames for Event-Based Vision

Johannes Baßler¹; Paul Bäcker²; Michael Heizmann¹ 

¹Karlsruhe Institute of Technology, Institute of Industrial Information Technology (IIIT), Karlsruhe

²Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB), Karlsruhe

Abstract

While event cameras provide microsecond temporal resolution, purely asynchronous tracking algorithms remain strongly load-dependent, scaling poorly with both instantaneous event-rate bursts and the number of simultaneous particles. We present a deterministically scalable framework that converts this variable tracking workload into a bounded, GPU-parallel detection task by aggregating events into ultra-short 100 μ s binary frames. Validated in a physical free-fall sorting scenario, our architecture sustains a strict 10 kHz tracking loop on standard consumer hardware. It scales to 4,000 simultaneous trajectories within the 100 μ s deadline, achieving an effective $\mathcal{O}(1)$ core detection latency to guarantee the execution determinism required for industrial automation.

Keywords: Event-based vision, real-time multi-object tracking, binary frames, GPU acceleration, industrial automation

1 Introduction

The automated monitoring and sorting of rapidly moving objects – such as bulk materials traversing a free-fall section – poses severe challenges for classical machine vision. Standard area-scan cameras suffer from motion blur at exposure times required for adequate light collection, while operating them at ultra-high frame rates generates prohibitive volumes of redundant data. Neuromorphic event cameras overcome these limitations by reporting only asynchronous, pixel-level brightness changes. Their microsecond-scale temporal resolution and high dynamic range theoretically provide the perfect sensory foundation for high-speed tracking applications [5]. Despite their theoretical advantages, deploying event cameras in real-time industrial environments remains rare. The core bottleneck lies in the computational paradigm. Purely asynchronous tracking algorithms process the data stream sequentially on an event-by-event basis. Consequently, when dozens of objects simultaneously enter the field of view, the resulting burst of millions of events per second forces the data association complexity to explode [1, 2, 3]. This strict dependency on the visual scene dynamics makes guaranteeing a constant Worst-Case Execution Time (WCET) practically impossible on standard Von Neumann architectures [5].

To bypass this sequential bottleneck, recent literature frequently transitions back to synchronous processing by accumulating events into dense representations such as time surfaces or voxel grids. However, to extract meaningful geometric features for classical computer vision algorithms, these approaches often require extended accumulation windows spanning tens of milliseconds [6, 7]. In

highly dynamic scenarios, this prolonged aggregation reintroduces motion blur, effectively nullifying the defining temporal advantage of the event sensor. While natively asynchronous Spiking Neural Networks (SNNs) offer a promising alternative, they currently lack the mature inference hardware and unified toolchains necessary for reliable, low-latency integration into industrial pipelines [13, 12].

In this paper, we present a deterministic, frame-based architecture for real-time multi-particle tracking. Building upon the concept of efficient binary representations [11], we restrict the event accumulation window to an ultra-short duration of 100 μ s. Because this 10 kHz quantization threshold is comparable to the hardware's inherent timestamp jitter and refractory period [16, 17], virtually no physically relevant temporal information is lost. By leveraging a decoupled, multi-threaded architecture with zero-overhead GPU execution paradigms, our system transforms the unpredictable, event-rate-dependent tracking workload into a highly parallelizable task with effective $\mathcal{O}(1)$ latency scaling for the core detection. We demonstrate that robust, microsecond-scale multi-object tracking is achievable on standard consumer hardware, providing a viable blueprint for deploying event-based vision in strict real-time settings.

2 Related Work

Purely asynchronous tracking algorithms aim to preserve the extreme temporal resolution of the sensor by processing each incoming event individually as it occurs. Prominent methodologies in this domain include event-based clustering, such as the asynchronous mean-shift algorithm proposed by Barranco et al. [1], and the *AEB tracker* in-

troduced by Wang et al. [2]. The *AEB tracker*, for instance, associates individual events to spatio-temporal Gaussian models using nearest-neighbor classification, whereas feature-driven methods like *HASTE* (Alzugaray & Chli) [3] process single events via continuous multi-hypothesis evaluation. While this event-by-event paradigm minimizes latency for sparse scenes with a limited number of targets, it exhibits inherent scalability constraints. As detailed by Gallego et al. [5, Sec. 4.1 and 4.2], the purely asynchronous event-by-event processing paradigm limits parallelizability, as the computational complexity of data association intrinsically scales with the dynamically varying event rate. Alternatively, Spiking Neural Networks (SNNs) offer a natively asynchronous parallel processing paradigm [4]. However, simulating the irregular firing patterns of SNNs on standard GPUs causes severe control-flow divergence [12], while dedicated neuromorphic processors currently lack the unified compiler toolchains and hardware maturity required for strict industrial integration [13].

To overcome the limitations of sequential processing, a widely adopted alternative is to aggregate asynchronous events into synchronous, dense representations. Prominent examples include event frames, Time Surfaces [6], and spatio-temporal voxel grids [7]. This transformation mitigates the data association bottleneck by enabling the direct application of mature, highly parallelizable computer vision algorithms and standard tracking-by-detection pipelines. However, a critical trade-off emerges: to extract meaningful spatial features, these representations require a sufficient spatial density of events to form recognizable structures and continuous edges. To achieve this spatial completeness, conventional methods typically rely on extended accumulation windows – often ranging from 10 ms to 50 ms depending on the scene dynamics [8, 10, 7]. In high-speed applications, such extended temporal windows inevitably accumulate overlapping particle positions, reintroducing severe intra-frame motion blur and thereby compromising the μs -scale temporal precision of event-based sensors [9, 11]. At the same time, a second trade-off arises at the system level: as the effective frame rate increases, many existing tracking pipelines are no longer fast enough to process the generated data within the available real-time budget. Thus, beyond representational fidelity, practical high-speed event-based tracking is also constrained by the real-time barrier imposed by the computational throughput of the tracking algorithm itself [5]. Our approach bridges these paradigms by explicitly prioritizing execution determinism without sacrificing temporal fidelity. Building upon the concept of efficient binary frames [11], we utilize an ultra-short accumulation window of 100 μs . By processing these frames at

10 kHz, we transform the unpredictable workload of asynchronous tracking into a deterministically parallelizable GPU task, yielding a predictable worst-case execution time (WCET) governed by fixed per-frame operations rather than the instantaneous event rate.

3 System Architecture and Methodology

At a target processing frequency of 10 kHz, the temporal budget (100 μs) for data ingestion, memory transfers, and system state updates is limited. Implementing a sequential, single-threaded processing pipeline at this frequency is highly impractical. The accumulation of blocking sensor I/O operations, host-to-device memory transfer latencies, and CPU-side CUDA driver overheads – which typically consume 5 μs to 15 μs per individual kernel launch [14] – rapidly consumes a prohibitive fraction of the available 100 μs deadline. To physically realize this frame rate and maintain execution determinism without data loss, we propose a decoupled, multi-threaded architecture supported by zero-overhead GPU execution paradigms. An overview of the system architecture is illustrated in Fig. 1.

3.1 Asynchronous Multi-Threaded Pipeline

To achieve a constant WCET and prevent the highly variable event rate from stalling the system, the architecture is divided into three asynchronous stages communicating via lock-free ring buffers:

- **Sensor I/O (Thread A):** The camera callback continuously accumulates incoming events into a bit-packed memory structure (*FastBit-Buffer*). Exactly every 100 μs , the buffer is swapped and passed to the execution queue, establishing the deterministic frame rate.
- **Detection (Thread B):** This host-side thread controls the GPU detection pipeline, launching the dense detection and centroid refinement kernels and forwarding the resulting spatial coordinates to the tracking queue.
- **Tracking (Thread C):** Operating entirely asynchronously on the CPU, the tracker consumes the detection coordinates, updates the physical state of the particles, and manages the lifecycle of the trajectories.

This pipelined approach isolates the sensor I/O from the downstream algorithmic workload. Thread A still scales with the incoming event count, but its callback is reduced to a pre-allocated bit update per event, avoiding clustering, data association, and dynamic allocation. Because the constant factor of these operations is negligibly small, the thread effortlessly absorbs extreme event bursts without exceeding its fractional time budget. This lock-free decoupling ensures that downstream computations

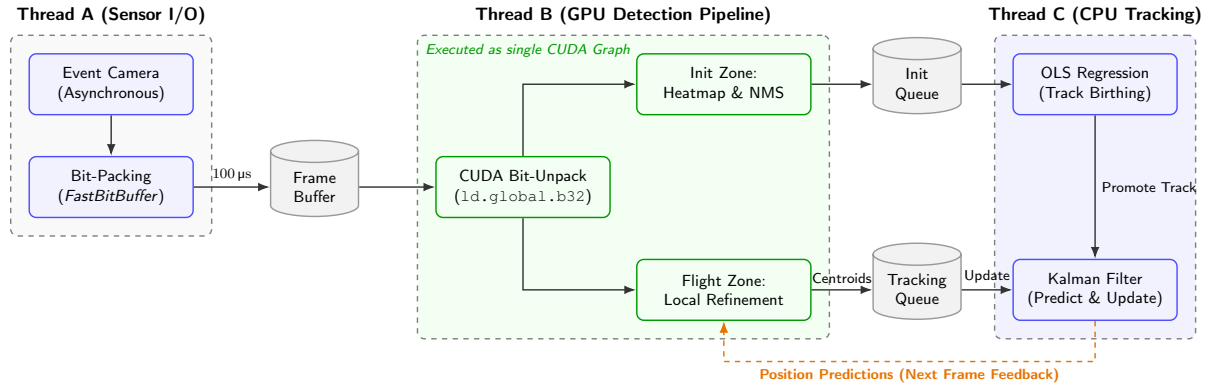


Figure 1: Overview of the decoupled, multi-threaded system architecture. The pipeline converts asynchronous events into deterministic 10 kHz binary frames (Thread A). The dense GPU detection and centroid refinement (Thread B) is fully encapsulated within a single CUDA Graph to eliminate kernel launch overhead. Thread C performs localized trajectory tracking by continuously feeding Kalman predictions back to the GPU, entirely bypassing global $\mathcal{O}(M \times N)$ data association common in conventional event-based tracking.

– such as updating the state of dozens of trajectories in Thread C – never exert backpressure on the camera SDK. The sensor interface is never blocked, guaranteeing a deterministic 10 kHz frame generation loop.

3.2 GPU Detection Pipeline

The object detection is performed solely on dense representations. However, transferring and processing full-resolution matrices at 10 kHz requires aggressive optimization.

Bit-Unpacking and Memory Access: To bypass PCIe bandwidth limitations, the frames are transferred to the GPU in their natively compressed, bit-packed format (1 bit per pixel). To minimize runtime overhead, we implement a custom CUDA unpacking kernel that reconstructs the binary frame directly in device memory. During centroid refinement, the local search window is scanned using 4-byte loads to reduce memory overhead. These loads are expressed via fixed-size `memcpy` operations rather than pointer casts, avoiding strict-aliasing violations while still allowing NVCC to emit a single 32-bit global memory load (`ld.global.b32`) instead of four separate 8-bit loads. This reduces the instruction count during local refinement and improves memory access efficiency.

Activity Heatmap and NMS: Following the frame generation, an activity heatmap is computed by aggregating spatial neighborhoods. To minimize computational overhead, this dense global search is not performed over the entire sensor plane. Instead, it is confined to a designated spatial initialization region used exclusively for spawning new trajectories (further detailed in Section 3.3). Within this

bounded region, we employ a highly parallel Non-Maximum Suppression (NMS) kernel to extract local maxima of event activity, followed by a sub-pixel centroid refinement. Because the matrix dimensions of this initialization zone are fixed, the computational cost of this dense operation remains $\mathcal{O}(1)$ relative to the scene’s activity and the incoming event rate.

CUDA Graph Execution: The most critical bottleneck at a 100 μ s time budget is the CPU-side CUDA driver overhead. A standard sequential dispatch of multiple dependent kernels (e.g., Unpack, Heatmap, NMS, and Refinement) along with their memory synchronizations incurs cumulative launch latencies and inter-kernel gaps that easily reach 20 μ s to 40 μ s, as the CPU struggles to keep the GPU queues fed [15]. At this scale, the dispatch overhead dominates the actual execution time, consuming nearly half the available frame budget before computation concludes. To eliminate this bottleneck, the entire detection sequence is pre-compiled into a *CUDA Graph*. This topology captures all kernel parameters and execution dependencies, allowing the entire pipeline to be dispatched with a single API call. By replacing the sequential driver interactions with a single graph launch, the dispatch overhead is reduced to a few microseconds.

3.3 Hybrid Trajectory Tracking

To maintain the strict 100 μ s execution cycle for dozens of concurrent particles, the CPU tracking module (Thread C) avoids standard global data association entirely. Instead, the system employs a tightly coupled, hybrid tracking architecture that conceptually divides the field of view. It applies rigorous multi-frame regression to securely initialize new targets, while established tracks in flight uti-

lize a continuous feedback loop between the CPU's Kalman filter predictions and the GPU's localized measurement extraction.

Robust Initialization via Spatial Regression: In industrial use cases like bulk material sorting, particles often follow a predictable primary motion vector, e.g., falling due to gravity, and enter the sensor's field of view at a specific boundary. Leveraging this *a priori* knowledge, we restrict track birthing to a narrow, designated spatial initialization zone spanning the entry edge (e.g., the top rows of the sensor, adapting to all edges if the entry vector is unknown). Detections in this zone are not immediately promoted to active tracks. Instead, they are aggregated into a temporal cache. A candidate is promoted to an active trajectory only if its recent history passes an Ordinary Least Squares (OLS) motion-consistency gate, requiring a minimum number of consecutive observations and a low residual Root Mean Square (RMS) error when fitting the cached detections as a linear trajectory over time. This physically constrained regression filters out spurious noise clusters without scanning the entire sensor plane for new targets.

Deterministic Flight Tracking: Once a track is established and leaves the initialization zone, global data association (e.g., Hungarian matching) is bypassed entirely. The success of this bypass relies directly on the high sampling rate. Because $\tau = 100 \mu\text{s}$ is sufficiently small relative to the particle's macroscopic velocity, the inter-frame spatial displacement of a particle's centroid is reduced to a fraction of a pixel. Instead of matching independent detections, the tracker utilizes a tightly coupled feedback loop. The CPU-based discrete Kalman filter predicts the particle's next position. These coordinates are directly fed into a secondary, localized GPU refinement kernel. By searching only within a highly constrained bounding box around the Kalman prediction, the GPU extracts the exact sub-pixel measurement and feeds it back to the CPU. This physically localized extraction handles even rapid non-linear trajectory deviations robustly and eliminates the $\mathcal{O}(M \times N)$ complexity of global association.

4 System Evaluation

This section evaluates the proposed architecture in a physical free-fall sorting setup, focusing on real-time determinism, per-thread latency, and scalability under both measured and synthetic loads.

4.1 Experimental Setup and Application Scope

To evaluate the real-time capabilities and hardware scalability of the proposed architecture, we consider

the highly dynamic use case of industrial bulk material sorting. The experimental setup consists of a vertical free-fall section where objects traverse the field of view primarily along the vertical sensor axis (denoted as the y -axis, with $y \in [0, 1279]$). To maximize the observable fall distance, the event camera is mounted rotated by 90° relative to its default landscape orientation, i.e., in a portrait-style configuration. Consequently, the sensor's long image dimension is aligned with gravity and defines the effective vertical tracking axis used throughout this section. The scene is uniformly illuminated by an LED backlight to ensure consistent contrast changes during the particles' flight.

Data acquisition is performed using an event camera (Prophesee EVK4 @ default settings) operating at its native 1280×720 resolution. The spatial initialization zone for spawning new trajectories is tightly constrained to a narrow horizontal strip spanning $y \in [200, 240]$ pixels. Once successfully initialized via the spatial regression gate, the particles' flight tracks are continuously refined until they cross the $y = 1000$ boundary. Beyond this exit zone, the particles enter the physical separation stage (e.g., a pneumatic valve array), rendering further tracking unnecessary for this application, although extended tracking would be computationally trivial.

The computational timing evaluation is based on over seven minutes of continuous event stream recordings acquired within a controlled physical laboratory setup. This extensive dataset encompasses a diverse spectrum of falling bulk materials, including small stones, wheat grains, coffee beans, fabric scraps, and dried tea leaves, generating thousands of unique object trajectories. These recordings are utilized exclusively to benchmark the system's hardware latencies, execution determinism, and scalability. While formal tracking accuracy metrics (e.g., identity switches or MOTA scores) fall outside the scope of this system architecture paper, the qualitative tracking robustness is effectively demonstrated. Even notoriously challenging targets exhibiting chaotic aerodynamics – such as irregularly shaped, fluttering dried tea leaves – are continuously tracked from entry to exit without fragmentation (see Fig. 5).

At the target 10 kHz frame rate ($\tau = 100 \mu\text{s}$), the average inter-frame displacement of a falling particle is approximately 0.5 pixels. This minimal inter-frame motion constitutes the foundational kinematic property that allows the system to bypass global data association entirely.

All computational benchmarks were executed on a standard consumer workstation equipped with an Intel Core i7-12700K CPU and an NVIDIA RTX 3060 GPU under a conventional Debian Linux environment. No real-time kernel, CPU isolation, or specialized hardware was used, so the reported latencies

reflect the determinism achieved by the software architecture itself.

4.2 Sensor I/O and Event Burst Immunity

The primary responsibility of Thread A is the continuous and deterministic ingestion of asynchronous events from the camera driver into the binary representations. The architectural details of this process are described in Section 3. Event-based sensors frequently produce temporal data bursts when dense object clusters traverse the field of view. To guarantee real-time operation, the pipeline must process these peak loads without inducing backpressure on the camera interface.

Figure 2 shows the callback-side processing latency of Thread A as a function of the incoming event rate, measured on the physical bulk-material datasets described above. Across the observed range up to approximately 25.6 Mevents/s, the CPU event-ingestion and bit-buffer update time increases only mildly and remains well below the 100 μ s frame deadline. The median latency rises from about 4 μ s at low activity to about 12 μ s in high-support bins. The 99th percentile reaches 27.0 μ s for well-supported bins and up to 36.4 μ s in rare cases. This metric excludes the separate frame-swap step and the asynchronous PCIe transfer ($\approx 22 \mu$ s), which do not compromise the pipeline's timing constraints. Overall, the measurements indicate substantial headroom before Thread A approaches the real-time limit.

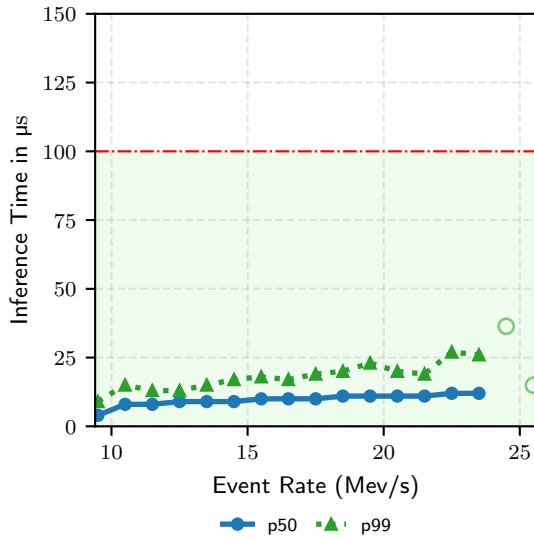


Figure 2: Callback processing latency of Thread A ($t_{a,us}$) versus incoming event rate, aggregated into 1 Mevents/s bins with median and 99th percentile latencies. The horizontal line marks the 100 μ s real-time deadline; circles denote low-support bins (< 100 frames) and their corresponding maximum values.

4.3 Compute Overheads and Tracking Scalability (Thread B)

Thread B controls the GPU-side detection pipeline, including activity-based detection and centroid calculation. The resulting particle sub-pixel coordinates are subsequently propagated to Thread C for the actual Kalman filter updates. Since the GPU kernels themselves execute within a small fraction of the available time budget, the measured end-to-end latency of Thread B is primarily determined by host-side orchestration overheads, including operating-system scheduling and CPU synchronization.

The empirical distribution of the end-to-end execution time for Thread B demonstrates this characteristic distinctly. The median total latency is 79.0 μ s, with the 99th percentile bounded at exactly 100.0 μ s (see Fig. 4). However, when isolating the raw GPU execution time for the detection and refinement branches from the total latency, the tracking computation accounts for merely 26.7 % of the time. The remaining 73.3 %, averaging 59.0 μ s per frame, is entirely consumed by operating system jitter, memory mapped I/O, and thread synchronization overheads.

To buffer against these non-deterministic scheduling spikes, the architecture uses a deep host-side ring buffer between Thread A (producer) and Thread B (inference worker), while Thread B processes the newest available frame to avoid backpressure. Analysis of the raw execution logs reveals that extreme system interrupts occasionally exceed the permissible timing constraints, forcing a frame drop. This scenario occurs in approximately one out of every 4,300 evaluated physical frames, equating to a drop rate of roughly 0.23 ‰. Because the Kalman filter algorithm in Thread C is inherently designed to predict trajectories across missing measurements, these isolated drops do not compromise the tracking integrity or the real-time reliability of the sorting process. Nevertheless, deploying the system on a real-time patched kernel with isolated CPU cores could theoretically eliminate these rare interruptions entirely. For the synthetic scalability analysis, we assumed $N_{init} = 0.1 N_{flight}$ in order to preserve a realistic workload ratio between the initialization-detection branch and the flight-refinement branch. This choice is motivated by the sorting geometry: only a small subset of particles is expected to occupy the narrow initialization zone at any given time, whereas most visible particles have already transitioned to active flight tracking. The synthetic regime is therefore intended as a scalability stress test rather than a physically realistic sorting density.

The computational headroom of the tracking implementation becomes apparent under synthetic load testing. While the physical sorting scenarios exhibit a maximum of approximately 70 simultaneous parti-

cles, the synthetic benchmark evaluated the pipeline up to 10,000 parallel flight trajectories. The end-to-end latency increases approximately linearly at a rate of roughly $0.006\ \mu\text{s}$ per particle. The system successfully maintained the $100\ \mu\text{s}$ real-time deadline for up to 4,000 simultaneously tracked objects, a volume that vastly exceeds the physical limits of the pneumatic ejection array. This is illustrated in Fig. 3.

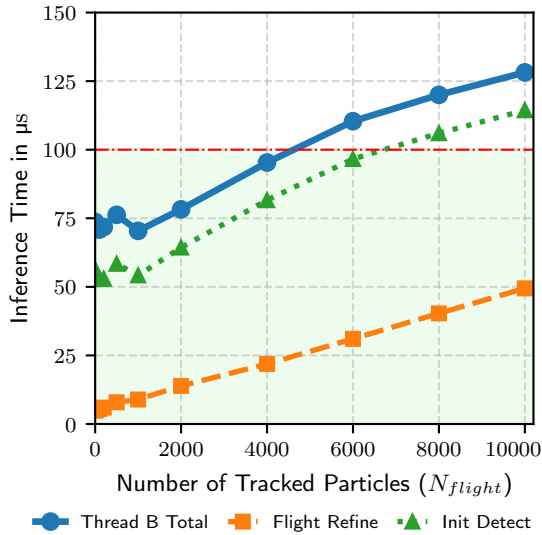


Figure 3: Mean inference time of Thread B for the physical measurements and the synthetic scalability benchmark. The curves show total Thread B latency, flight-track refinement, and initialization detection. In the synthetic benchmark, $N_{init} = 0.1 N_{flight}$ was assumed. The horizontal line marks the $100\ \mu\text{s}$ real-time deadline.

4.4 Downstream Processing and System Modularity

Thread C receives the tracking coordinates from Thread B, where the asynchronous event stream has been fully reduced to dense geometric information. The empirical execution latency of Thread C (see Fig. 4) confirms that trajectory management operates within the real-time constraints, with rare OS scheduling spikes occasionally exceeding the $100\ \mu\text{s}$ deadline without compromising overall system performance. For the evaluated sorting task, a discrete Kalman filter predicts and maintains particle trajectories from this structured coordinate data.

The optimization of this tracking backend extends beyond the scope of this work. The system architecture is designed with modularity in mind. The Kalman filter module can be substituted with alternative downstream algorithms tailored to different use cases. This decoupled design allows for flexible application logic integration without affecting the deterministic event ingestion front-end.

4.5 Overall System Performance and Algorithmic Scalability

The fundamental advantage of the proposed architecture is the decoupling of computational load from visual scene dynamics: by converting the asynchronous stream into fixed-size binary frames, the core detection achieves an effective $\mathcal{O}(1)$ runtime. While Thread A still scales marginally with the raw event volume and flight refinement scales linearly with the number of active tracks, these slopes are sufficiently flat to maintain robust real-time operation.

Figure 4 illustrates the combined latency distributions of the three processing stages, confirming that the end-to-end processing time remains bounded below the real-time deadline with rare OS jitter exceptions.

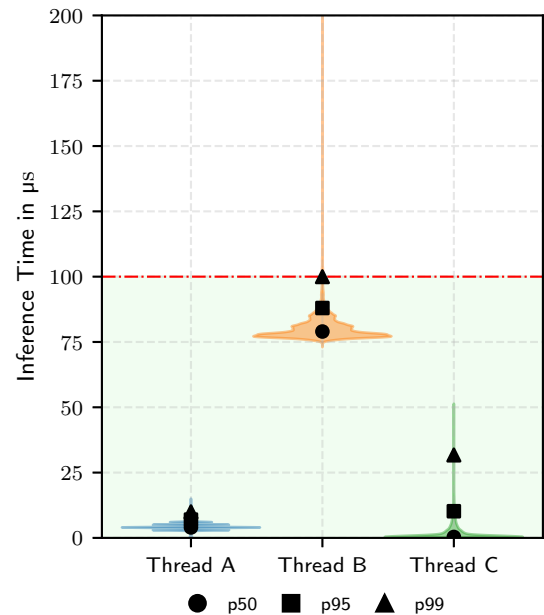


Figure 4: Per-frame inference-time distributions of Threads A, B, and C. Internal markers denote the median, 95th percentile, and 99th percentile, and the horizontal line marks the $100\ \mu\text{s}$ real-time deadline.

4.6 Temporal Fidelity and Information Retention

Despite transitioning from an asynchronous event stream to a synchronous processing pipeline, the proposed architecture discards virtually no relevant sensor information under the evaluated camera configuration. The $100\ \mu\text{s}$ accumulation window was chosen to be short relative to the effective per-pixel event rate observed at the vendor default settings. Neuromorphic sensors exhibit a bias-dependent refractory period, i.e., a recovery time before an individual pixel can register a subsequent brightness change. While aggressive bias settings may reduce this period below $100\ \mu\text{s}$ [16, 17], repeated events

from the same pixel within one accumulation window were empirically negligible in our evaluation: only 0.04‰ of all events were lost due to the binary representation.

Furthermore, while modern event cameras assign timestamps with nominal microsecond resolution, the true temporal accuracy is bounded by analog comparator delays and digital bus arbitration jitter. Recent hardware characterizations demonstrate that this inherent timestamp uncertainty consistently ranges from tens to hundreds of microseconds [17]. Therefore, precision beyond the 100 μ s quantization threshold is dominated by sensor noise rather than actual scene dynamics.

4.7 Qualitative Tracking Evaluation

While the preceding sections extensively quantify the real-time processing capabilities of the architecture, the utility of the system inherently depends on its tracking integrity. The application domain of industrial bulk sorting constitutes a highly controlled environment characterized by a static background and consistent object illumination. In addition, the sensor is deliberately mounted in a portrait-style orientation so that its long image axis follows the free-fall direction, maximizing the observable trajectory length within a single view. Consequently, the tracking module is intentionally specialized for these conditions.

Within this operational scope, the algorithmic pipeline demonstrates robust tracking capabilities despite complex object dynamics. Figure 5 visualizes the reconstructed flight paths of falling, dried tea leaves captured by the event camera. Due to their irregular geometry and aerodynamic properties, these particles exhibit chaotic rotational and translational movements during their free fall. Nevertheless, the combination of binary centroid detection and Kalman filtering successfully isolates and continuously tracks up to 18 simultaneous particles within this specific recording. The visual assessment confirms that the generated trajectories are smooth, continuous, and resistant to trajectory fragmentation. This reliable spatial representation provides the prerequisite information required for accurate downstream pneumatic ejection commands.

5 Conclusion

This paper demonstrates that event-based multi-particle tracking can be executed deterministically at 10 kHz on standard consumer hardware by converting asynchronous event streams into ultra-short binary frames. The proposed multi-threaded architecture separates event ingestion, GPU-side detection, and trajectory management, thereby preventing instantaneous event-rate bursts from directly propagating into the core detection workload. In the evaluated sorting scenario, the system maintains the 100 μ s processing deadline and scales to

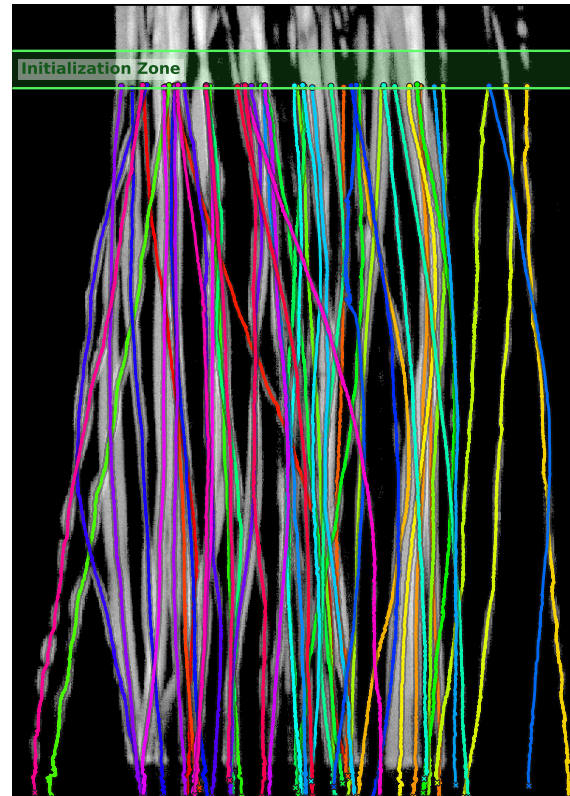


Figure 5: Composite trajectory map over time-surface-filtered event activity. Each trajectory is color-coded; the highlighted *Initialization Zone* marks the constrained region where new tracks are seeded and OLS-validated before promotion to active flight tracking. This plot covers a 7 s spanning recording of falling dried tea leaves, demonstrating the system’s ability to maintain continuous tracks despite complex aerodynamics.

4,000 simultaneously tracked particles in the synthetic stress test.

The evaluation confirms that the main remaining bottleneck is no longer the GPU-side detection computation itself, but host-side scheduling and synchronization overhead. At the same time, the current implementation is intentionally specialized for controlled optical sorting scenes, where static backgrounds, consistent illumination, and a known entry region enable simplified initialization and local refinement. Quantitative tracking-accuracy comparisons against asynchronous tracking algorithms are therefore left to future work.

Future work will focus on reducing operating-system jitter through real-time kernels and isolated CPU cores, and on extending the architecture to less constrained environments through robust background suppression and dynamic data association.

References

- [1] F. Barranco, C. Fermüller, and E. Ros, "Real-Time Clustering and Multi-Target Tracking Using Event-Based Sensors," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 5764–5769, doi: 10.1109/IROS.2018.8593380.
- [2] Z. Wang, T. Molloy, P. van Goor, and R. Mahony, "Asynchronous Blob Tracker for Event Cameras," *IEEE Transactions on Robotics*, vol. 40, pp. 4750–4767, 2024, doi: 10.1109/TRO.2024.3454410.
- [3] I. Alzugaray and M. Chli, "HASTE: multi-Hypothesis Asynchronous Speeded-up Tracking of Events," in *31st British Machine Vision Conference 2020, BMVC 2020*, Virtual Event, UK, September 7–10, 2020. BMVA Press, 2020.
- [4] S. Nagaraj et al., "DOTIE: Spiking Object Detection for Event Cameras," *arXiv preprint arXiv:2208.06648*, 2022. [Online]. Available: <https://arxiv.org/abs/2208.06648>
- [5] G. Gallego et al., "Event-based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 44, no. 1, pp. 154–180, Jan. 2022, doi: 10.1109/TPAMI.2020.3008413.
- [6] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "HOTS: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, pp. 1346–1359, 2017, doi: 10.1109/TPAMI.2016.2574707.
- [7] D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza, "End-to-End Learning of Representations for Asynchronous Event-Based Data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, South Korea, 2019, pp. 5633–5643.
- [8] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to Detect Objects with a 1 Megapixel Event Camera," in *Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16639–16652.
- [9] N. Messikommer, C. Fang, M. Gehrig, and D. Scaramuzza, "Data-Driven Feature Tracking for Event Cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Vancouver, BC, Canada, 2023, pp. 5642–5651.
- [10] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras," in *Robotics: Science and Systems (RSS)*, Pittsburgh, PA, USA, 2018.
- [11] J. Baßler and M. Heizmann, "Beyond the Sparsity Paradigm: Efficient Binary Frames for Event-Based Vision," *Proceedings of SPIE Optical Metrology*, Munich, Germany, 2025. [Online]. Available: <https://publikationen.bibliothek.kit.edu/1000188440>
- [12] M. Davies et al., "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021, doi: 10.1109/JPROC.2021.3067593.
- [13] C. D. Schuman et al., "Opportunities for Neuromorphic Computing Algorithms and Applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022, doi: 10.1038/s43588-021-00184-y.
- [14] NVIDIA Corporation, *CUDA C++ Best Practices Guide*, Version 12.8, 2024. [Online]. Available: <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/>
- [15] A. Kerr et al., "Getting Started with CUDA Graphs," *NVIDIA Technical Blog*, Sep. 2019. [Online]. Available: <https://developer.nvidia.com/blog/cuda-graphs/>
- [16] S. McMahon-Crabtree et al., "Event-based camera refractory period characterization and initial clock drift measurement," in *Applications of Machine Learning 2023*, vol. 12693. SPIE, 2023, p. 126930V, doi: 10.1117/12.2672301.
- [17] A. Manakov, H. Hermann and B. Jähne, "Characterisation metrics for performance comparison of area scan and event-based sensors," *tm - Technisches Messen*, De Gruyter, 2025, doi: 10.1515/teme-2025-0018.