ASYNCHRONOUS HIGH-SPEED TRACKING OF ASTRONOMICAL OBJECTS USING NEUROMORPHIC CAMERA FOR EDGE COMPUTING

Satyapreet Singh Yadav[*], *Ajay Vikram P*[*], *Adithya MD*[*], *Chandra Sekhar Seelamantula*[†], *Chetan Singh Thakur*[*]

[*]Department of Electronic Systems Engineering, Indian Institute of Science, Bangalore, India
[†]Department of Electrical Engineering, Indian Institute of Science, Bangalore, India
Emails: {satyapreets, css, csthakur}@iisc.ac.in, ajayvikramp@gmail.com, amdadithya@gmail.com

ABSTRACT

Neuromorphic event-based cameras provide low latency, high dynamic range, and sparse output, enabling efficient tracking of high-speed objects at low data rates. Although widely applied in robotics for moderate-speed scenarios, their use in Space Situational Awareness (SSA) remains nascent. We present a real-time, frame-free, multi-target clustering and tracking framework that operates directly on asynchronous events using an Extended Kalman Filter (EKF). Its event-driven design naturally suppresses background noise and hot pixels, achieving near-constant computational complexity, making it well-suited for resource-constrained SSA edge platforms. Our system consists of a Prophesee Gen4 (IMX636) event sensor connected to a Raspberry Pi 4 via USB, running our algorithm on-device in real time. We present an event-driven EKF that can track blobs moving at $> 17{,}000$ px/s with per-event EKF latency of $< 0.7\,\mu$s. The end-to-end pipeline (clustering+EKF) runs in $\sim 1\,\mu$s/event, achieving < 12 px localization error and clustering F1-score of > 0.9. We extensively evaluated the system through LED-matrix multi-target simulations and telescope-based outdoor trials, including tracking a fast-moving Starlink satellite and a slow-drifting star field. The results demonstrate its potential for SSA applications.

***Index Terms*—** Neuromorphic event-based cameras, Space situational awareness, Multi-target tracking, Asynchronous event processing, Edge computing, Extended Kalman Filter

1. INTRODUCTION

Tracking satellites and space debris is central to Space Situational Awareness (SSA), enabling timely collision-avoidance manoeuvres [1, 2]. Operationally, satellites are commanded through ground-station contacts. Since uplinks are confined to line-of-sight visibility windows [3], successive passes are separated by gaps of minutes, adding huge latency to satellite control. With >1 million fragments travelling at >7 km/s evading routine tracking [4], collision risk is rising, motivating onboard autonomy that processes sensor data in situ and executes real-time manoeuvre decisions [5, 6, 7]. This shift demands high-speed sensors paired with edge-computing systems within space platforms' compute, power, and form-factor limits.

Cameras are widely used for real-time visual tracking. Still, the space environment with low illumination, bright source glare, and rapidly approaching targets stresses conventional frame-based sensors (Fig. 1c). These systems suffer motion blur, limited dynamic range (< 60 dB) [8, 9, 10], and high, redundant data rates (1080p/30 fps/10-bit $\sim$ 1.7 Gbps) that burden onboard processing . In contrast, neuromorphic event-based sensors deliver μs-scale temporal resolution, ≥ 120 dB dynamic range [11, 12, 13], and sparse asynchronous output (Fig. 1a; 1 Mev/s $\sim$ 16 Mbps), eliminating motion blur (Fig. 1b) and enabling rapid detection/tracking of fast objects [14].

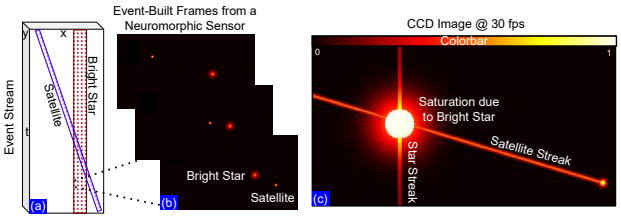

Fig. 1. (a) Neuromorphic sensor emits sparse events (x, y, p, t). (b) Its high dynamic range resists bright-star glare and localises a fast satellite. (c) A charge-coupled device (CCD) displays motion-blur streaks due to a fixed-frame rate and saturates near a bright star due to limited dynamic range, breaking satellite tracking.

Prior Work. Although neuromorphic cameras can deliver events with μs precision, many existing approaches first bin events into fixed temporal windows and then apply deep learning methods for tracking [15, 16, 17]. This binning step reduces temporal resolution and limits the detection of high-speed objects. At the same time, the reliance on data-driven models makes performance highly dependent on the availability of large, representative training datasets, which are currently impractical for SSA due to limited or non-existent data availability. At long ranges (>1 km), such objects typically appear as blobs [18]. While asynchronous clustering and tracking are demonstrated in [19], the method targets relatively slow platforms and reports optical-flow throughput, but does not provide ground-truth tracking-error evaluation or an on-device, real-time hardware implementation. Hardware implementations exist, but frame-based designs sacrifice temporal precision [20, 21]; frameless systems often assume slow, near-linear motion [22, 23] or cap concurrency at ≤ 4 targets [24].

Contributions. In this work, we present an asynchronous event-based clustering and tracking framework with near-constant per-event complexity, combining a lightweight Constant-Turn EKF (CT-EKF) tracker with constant-neighbour search algorithm. We implement the pipeline in C/C++ on Raspberry Pi (RPi) 4 to track high-speed astronomical objects. We evaluate it on an indoor LED-matrix testbed that simulates fast-moving blobs and report tracking error, clustering scores, and end-to-end latency. The system supports multi-target tracking. Our method delivers good tracking accuracy and responsiveness, making it well-suited for autonomous SSA in both ground- and space-based deployments.

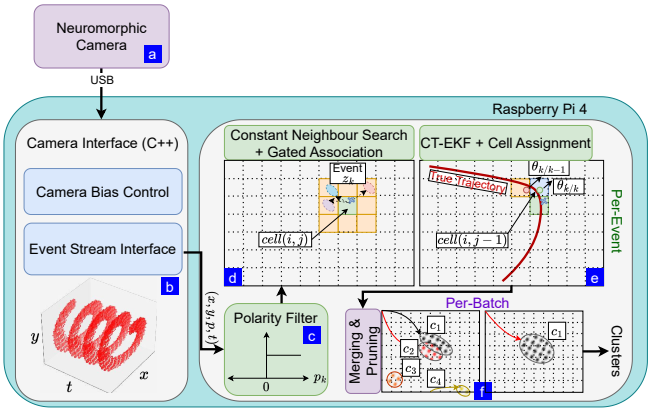

Fig. 2. Proposed Architecture running on RPi 4. (a) A neuromorphic camera streams events over USB; (b) a C++ interface configures biases and ingests the stream. (c) A polarity filter selects positive events. (d) For an event in cell (i, j) (green), constant-neighbour search restricts candidates to clusters in adjacent cells (peach) and applies gated Euclidean association. (e) The cluster's CT-EKF is updated. If its centroid crosses a cell boundary (blue), the cluster is re-indexed to the new cell. Steps (c–e) run per event. (f) After each batch of events, nearby clusters are merged, stale or out-of-field of view (FoV) clusters are pruned. Steps (c-f) are implemented in C.

2. METHODOLOGY

Distant targets (stars, most satellites) are unresolved when observed from Earth or far-away distances and appear as compact point spread functions in our images. Their apparent motion generates brightness changes that trigger asynchronous events when the sensor's log-intensity crosses a preset threshold. Under our bias settings (on = 140, off = 190), negative-polarity activity exhibits longer persistence and latency [25]. Thus, as depicted in Fig 2c, we restrict processing to only positive events, which lowers the per-update rate without affecting detection. Each positive event updates a CT-EKF. The CT model captures non-linear apparent curvature arising from the (a) relative observer–target geometry (b) earth rotation and motion jitter over short observation intervals.

2.1. Constant-Turn EKF (CT-EKF)

We track each target with a n_s-dimensional state with $n_s = 5$

$$\mathbf{x}_k = \begin{bmatrix} p_{x,k} & p_{y,k} & v_k & \theta_k & \omega_k \end{bmatrix}^\top$$

, with image-plane position $(p_{x,k}, p_{y,k})$ in pixels, speed v_k (px/s), heading angle θ_k (rad), and turn-rate ω_k (rad/s) at the k-th event. With $\Delta t = t_k - t_{k-1}$, we define the discrete-time CT-EKF prediction step as:

$$\hat{\mathbf{x}}_{k|k-1} = \begin{bmatrix} p_{x,k-1} + v_{k-1} \int_0^{\Delta t} \cos(\theta_{k-1} + \omega_{k-1} s)\, ds \\ p_{y,k-1} + v_{k-1} \int_0^{\Delta t} \sin(\theta_{k-1} + \omega_{k-1} s)\, ds \\ v_{k-1} \\ \theta_{k-1} + \omega_{k-1} \Delta t \\ \omega_{k-1} \end{bmatrix} \tag{1}$$

with process noise $\boldsymbol{w}_k \sim \mathcal{N}(\mathbf{0}, Q_k)$ and

$$Q_k = \text{diag}\left(q_p,\ q_p,\ q_v,\ q_\theta,\ q_\omega\right) \Delta t. \tag{2}$$

where q_p, q_v, q_θ, q_ω are variance rates with units [px^2/s], [px^2/s^3], [rad^2/s], and [rad^2/s^3] respectively. Linearization of the CT dynamics about state $\mathbf{x}_{k-1}$ with $|\omega_{k-1}\Delta t| \to 0$ yields

$$F_k \triangleq \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Big|_{\mathbf{x}_{k-1}} = \begin{bmatrix} 1 & 0 & \cos\theta_{k-1}\,\Delta t & -v_{k-1}\sin\theta_{k-1}\,\Delta t & 0 \\ 0 & 1 & \sin\theta_{k-1}\,\Delta t & v_{k-1}\cos\theta_{k-1}\,\Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The prediction and update steps of the CT–EKF, using the measurement model $H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$, measurement noise $R = \text{diag}(r_p, r_p)$ with r_p being the position measurement noise variance in px^2, state error covariance P, innovation covariance S, and Kalman gain K, are summarized in Algorithm 1.

Algorithm 1: Constant–Turn EKF (per event, k)

Input: Previous estimate $(\hat{\mathbf{x}}_{k-1|k-1},\ P_{k-1|k-1})$, time difference Δt, event with location $\mathbf{z}_k = \begin{bmatrix} x_k,\ y_k \end{bmatrix}^\top$, process rates $q_p, q_v, q_\theta, q_\omega$, measurement noise R, wrap operator $\text{wrap}(\cdot)$

Output: Updated estimate $(\hat{\mathbf{x}}_{k|k},\ P_{k|k})$

State/Model: state $\mathbf{x}_k$; Measurement model H

1. Prediction

$\hat{\mathbf{x}}_{k|k-1} \leftarrow f(\hat{\mathbf{x}}_{k-1|k-1},\ \Delta t)$,

$\hat{\theta}_{k|k-1} \leftarrow \text{wrap}(\hat{\theta}_{k|k-1})$ // Eq. (1)

$P_{k|k-1} \leftarrow F_k\, P_{k-1|k-1}\, F_k^\top + Q_k$ // Eqs. (3), (2)

2. Measurement update

$\hat{\mathbf{z}}_{k|k-1} \leftarrow H\, \hat{\mathbf{x}}_{k|k-1}$

$\mathbf{y}_k \leftarrow \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1}$ // innovation

$S_k \leftarrow H\, P_{k|k-1}\, H^\top + R$ // innovation covariance

$K_k \leftarrow P_{k|k-1}\, H^\top\, S_k^{-1}$ // Kalman gain

$\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} + K_k\, \mathbf{y}_k$; $\hat{\theta}_{k|k} \leftarrow \text{wrap}(\hat{\theta}_{k|k})$

$P_{k|k} \leftarrow (I - K_k H)\, P_{k|k-1}\, (I - K_k H)^\top + K_k R K_k^\top$

2.2. Asynchronous Clustering and Tracking

Each positive event $e_k = (x_k, y_k, t_k)$ updates a dynamic set of clusters $\mathcal{C}$. A cluster $c \in \mathcal{C}$ stores cluster id c, centroid $\mathbf{c}_k^c = [c_{x,k}, c_{y,k}]^\top$, previous centroid $\mathbf{c}_{k-1}^c = [c_{x,k-1}, c_{y,k-1}]^\top$, velocity $\mathbf{v}^c = [v_x, v_y]^\top$, count of associated events n_c, time of most recent associated event t_{last}, time of first associated event t_{init}, time of cluster birth t_{birth}, EKF state, active flag $a_c \in \{0, 1\}$, missed counter m_c, gate threshold d_c. All cluster fields are updated asynchronously according to Algorithm 2.

Constant-Cost Neighbor Search. To make per–event computational cost independent of the active clusters M, we partition the image into square cells of side $s = 3\, d_{\text{base}}$, where d_{base} is a fixed distance threshold used to initialize each cluster's gate threshold d_c. Each cell stores up to K cluster pointers (by centroid). For an event in cell (i, j), we search only neighboring cells within (Fig. 2d)

$$r = \left\lceil \frac{2 \max_{c \in \text{cell}(i,j)} d_c}{s} \right\rceil \quad (\text{often } r = 1), \tag{4}$$

yielding at most $(2r + 1)^2 K$ candidates (e.g., $\leq 9K$), independent of M. Hence association time is $O(1)$ in M with predictable latency. When a centroid crosses a cell boundary, we remove its pointer from the old cell and insert it into the new cell (Fig. 2e).

Algorithm 2: Asynchronous clustering and tracking

Input: Batch size N; grid $\mathcal{G}$ (cell size s, cap K); cluster set $\mathcal{C}$; thresholds $d_{\text{base}}, d_{\text{merge}}, m_{\max}, t_{\min}, v_{\min}, n_{\min}$; CT-EKF, global event counter $n_{\text{evt}} \leftarrow 0$

Output: Updated $\mathcal{C}$

Seeding: if $\mathcal{C} = \varnothing$ ***or*** $\forall\, c \in \mathcal{C} :\ \|\mathbf{y}_k^c\|_2 > d_c$ **then**
  create c at (x_k, y_k); $\mathbf{c}_k^c \leftarrow \mathbf{c}_{k-1}^c \leftarrow [x_k, y_k]^\top$; $(t_{\text{birth}}, t_{\text{init}}, t_{\text{last}}) \leftarrow t_k$; $n_c \leftarrow 1$; $d_c \leftarrow d_{\text{base}}$; $\hat{\mathbf{x}}_{k|k-1}^c \leftarrow [x_k\ y_k\ 0\ 0\ 0]^\top$; insert c into $\mathcal{G}$ with rest of cluster fields been zero.

Batch initialization: foreach $c \in \mathcal{C}$ **do**
  $\mathbf{c}_{k-1}^c \leftarrow \mathbf{c}_k^c$; $t_{\text{init}} \leftarrow t_{\text{last}}$; $a_c \leftarrow 0$;

Per-event processing for $e_k = (x_k, y_k, t_k)$**:**
  $\texttt{cell}(i,j) \leftarrow (\lfloor x_k/s \rfloor, \lfloor y_k/s \rfloor)$; compute r via Eq. (4); gather candidates $\mathcal{N}$ from the $(2r{+}1) \times (2r{+}1)$ neighborhood (fixed scan order); *assigned*$\leftarrow$ **false**
  foreach $c \in \mathcal{N}$ **do**
    predict $(\hat{\mathbf{x}}_{k|k-1}^c, P_{k|k-1}^c)$ via Eq. (1); $\hat{\mathbf{z}}_{k|k-1}^c \leftarrow H\hat{\mathbf{x}}_{k|k-1}^c$, $\mathbf{y}_k^c \leftarrow [x_k\ y_k]^\top - \hat{\mathbf{z}}_{k|k-1}^c$, $r^c \leftarrow \|\mathbf{y}_k^c\|_2$;
    if $r^c \le d_c$ **then**
      // Winning cluster update
      EKF update (Alg. 1) with $\mathbf{z}_k = [x_k\ y_k]^\top$; $\mathbf{c}_k^c \leftarrow H\hat{\mathbf{x}}_{k|k}^c$; $t_{\text{last}} \leftarrow t_k$; $n_c \leftarrow n_c{+}1$; $m_c \leftarrow 0$; $a_c \leftarrow 1$;
      if $\texttt{cell}(\mathbf{c}_k^c) \neq \texttt{cell}(\mathbf{c}_{k-1}^c)$ **then**
        move pointer to new cell
      assigned$\leftarrow$ **true**; **break**;

if *assigned* ***is false*** **then**
  apply the seeding clause above

$n_{\text{evt}} \leftarrow n_{\text{evt}} + 1$

if $n_{\text{evt}} = N$ **then**
  // Periodic maintenance: merge, prune, rebuild
  Merge: apply Eqs. (5), (6); **Prune:** apply Eq. (7); **Cluster Re-assignment:** apply Eq. (8);
  $n_{\text{evt}} \leftarrow 0$;

Cluster Merging. Extended or bright targets can spawn multiple nearby clusters, while small targets may split under jitter. To suppress fragmentation, we merge spatially proximal clusters after receiving a batch of N events (Fig. 2f) if they satisfy the **Merge condition:** For clusters $c_i, c_j \in \mathcal{C}$ with centroids $\mathbf{c}^{c_i}, \mathbf{c}^{c_j} \in \mathbb{R}^2$, merge if

$$\left\|\mathbf{c}^{c_i} - \mathbf{c}^{c_j}\right\|_2 < d_{\text{merge}}. \tag{5}$$

Let the survivor be the cluster with the larger event count, $c_{\text{surv}} = \arg\max(n_{c_i}, n_{c_j})$, and the other, $c_{\text{dead}} = \{c_i, c_j\} \setminus \{c_{\text{surv}}\}$. We perform one EKF update on c_{surv} using the dead centroid as a pseudo-measurement to re-centre on the merged evidence. We then delete c_{dead} and enlarge the survivor's gate:

$$d_c^{surv} \leftarrow \min\Big(1.2\, \max\left(d_c^{(surv)}, d_c^{(dead)}\right),\ 4\, d_{\text{base}}\Big). \tag{6}$$

Cluster Pruning. We prune clusters after a batch of N events (Fig. 2f) if they satisfy any of the following conditions.

$$\text{age}_c = t_{\text{last}} - t_{\text{birth}} > t_{\min} \text{ and } \|\mathbf{v}^c\|_2 < v_{\min}, \tag{7a}$$

$$n_c < n_{\min}, \quad m_c > m_{\max}, \quad \mathbf{c}^c \notin [0, W) \times [0, H). \tag{7b}$$

Condition (7a) removes hot-pixel clusters since they show negligible velocity after a certain age; Condition (7b) prunes low-support noise clusters with few events per batch; deletes clusters which become stale with too many consecutive misses; and drops clusters that moved outside the FoV.

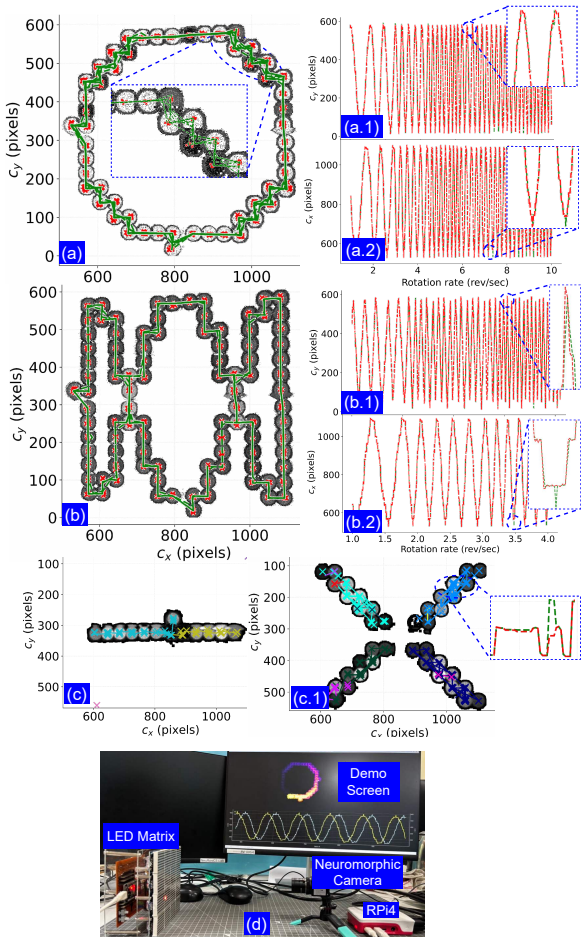

Fig. 3. LED-matrix Simulation Results: Panels (a–c) summarise the circular, Lissajous, and multi-object collision simulations, with insets (a.1–b.1/a.2–b.2) zooming speed ramps and cluster assignments. Panel c shows the instant impact between two bodies, while panel c.1 captures the post-collision debris ejecting radially outward with tracking close to ground truth. (d) Our real-time experimental setup with an LED matrix, a neuromorphic camera and an RPi4. **Overlays:** ground truth green; CT–EKF (ours) red; events gray.

Cluster Re-Assignment. Merging and pruning operations delete clusters, re-centre survivor clusters, and enlarge their d_c, invalidating cell pointers and the per–cell gate maxima used to set r. Hence, after each pruning/merging phase, we re-assign the clusters to cells. For each surviving cluster with current centroid $\mathbf{c}^c = [c_x, c_y]^\top$, we compute its cell and its index

$$(i,j) \leftarrow \big(\lfloor c_x/s \rfloor,\ \lfloor c_y/s \rfloor\big), \qquad \texttt{idx} \leftarrow i + j\, n_{\text{cols}}, \tag{8}$$

then insert c into cell $\texttt{idx}$ and update the cluster's gate d_c via Eq. (6) in case of merging operation and recompute r via Eq. (4). This flushes deletions, relocates moved survivors, and restores the constant candidate bound $(2r{+}1)^2 K$.

3. EXPERIMENTAL EVALUATION

We evaluate our asynchronous clustering and tracking algorithm using (i) a controlled LED-matrix testbed with programmable ground

truth and (ii) night-sky recordings using an 8-inch telescope.

LED-Matrix Testbed. We built a testbed comprising a matrix of 32×32 LEDs controlled by an ESP32 microcontroller. (Fig. 3d). The ESP32 running at 240 MHz streams per-pixel updates, providing deterministic cadence and intensity control. Trajectories are generated in Python at μs resolution and flashed as LED buffers. Single-target motion includes (i) circles and (ii) Lissajous curves, $x(t) = A\cos(at+\phi_x)$, $y(t) = B\sin(bt+\phi_y)$, with programmable kinematics. Across patterns, we sweep the turn-rate ω to simulate slow-fast motions. For multi-target studies, we simulated a two-body approach and collision, followed by the radial ejection of four faster fragments emulating debris.

Telescope Trials: Night–Sky Observations. A Prophesee Gen4 [26] was mounted at the focal plane of an 8-inch telescope; the effective FoV is $\approx 8'' \times 10''$. To induce drift without tracking artefacts, sidereal tracking was disabled so stars traversed the sensor at a near-constant rate. We recorded (i) *slow drift* with the tube fixed and (ii) *fast drift* via deliberate slews, stressing association across slow–fast transitions. Multiple Starlink passes (Fig. 4b), were also captured, and raw event streams were saved for offline playback and analysis.

4. RESULTS

We evaluated the asynchronous clustering and CT–EKF tracker running in real time on an RPi 4 at 1.5 GHz, implemented primarily in single-precision. We report (i) mean tracking error, (ii) mean per-event processing time from event arrival to completed association (including the EKF update), and (iii) clustering F1-score after matching predicted to reference clusters. As shown in Fig. 3(a–c) and Table 1, across all runs the tracker stayed close to ground truth: the mean tracking error was < 12 px for both single- and multi-target simulations over low-to-high rotation rates. The mean processing latency was ≈ 1.1 μs/event, with the EKF update averaging ≈ 0.7 μs/event.

Table 1. Real-time results of our implementation on RPi4.

Hyperparameters: $q_p = 0.5$, $q_v = 10^{-3}$, $q_\theta = 10^{-3}$, $q_\omega = 10^{-3}$, $r_p = 20.0$, $d_{base} = 20.0$ $d_{merge} = 40.0$, $n_{min} = 10$, $m_{max} = 40$, $N = 2000$, $K = 16$, $t_{min} = 0.001$, $v_{min} = 1.0$.

Testcase	Mean Tracking Err (px)	Mean Event Rate (Mev/s)	Mean Proc. Latency (μs)	Cluster F1 Score	Rot. Rate (rev/s)
Circular	10.71	0.5	1.145	0.99	1–10
Lissajous (a=1, b=3)	11.77	0.8	1.133	0.99	1–4
Multi-object collision	6.47	0.9	1.3	0.90	1

Abbreviations— px: pixel; Mev/s: mega-events per second; rev/s: revolutions per second.

Table 2. Comparison with prior work on asynchronous clustering & tracking algorithms for circular trajectory experiment.

Method	Algo	ω (rev/s)	Trackable speed (px/s)	H/W
jAER [27, 19]	Async. clustering + Linear Tracking	< 6	> 9000	No
jAER [27, 24]	(10 μs latency)	1.67	–	FPGA
Prophesse [28, 19]		< 6	> 2000	No
AB_tracker [19]	Async. Clustering + EKF (100 KHz)	< 6	> 11000	No
Ours	Async. Clustering + EKF (1.1μs latency)	< 10	> 17000	RPi4

Table 2 benchmarks the circular trajectory experiment against prior work: our implementation tracks $> 17{,}000$ px/s with ≈ 1.1 μs processing latency; while AB_TRACKER with no hardware implementation updates at > 100 kHz (~ 10 μs) and was tested up to 11,000 px/s. Other trackers are linear, operating at lower speeds with comparatively high latency.

For the telescope trials, we used d_{base}=40, N=3500, n_{min}=100. We captured multiple passes of Starlink, of which three representative passes are shown in Fig. 4(a,b). Pseudo-ground truth was obtained by performing clustering and tracking on frames by accumulating events over 2 ms duration (green track). Our implementation maintained a stable trajectory (red track) matching the pseudo-ground truth with a tracking error of < 5 px. We next imaged a stellar field containing bright and faint stars (Fig. 4c). With cluster-merge logic enabled, the CT-EKF maintained stable tracks for both bright (red track) and faint (purple track) sources across slow and fast-drift segments, without loss of track continuity, outperforming the frame-based baseline (green track), which lost tracking intermittently. Noise-induced, stale clusters visible in Fig 4c get pruned as they age; thus, no stale clusters are visible in Fig. 4d.

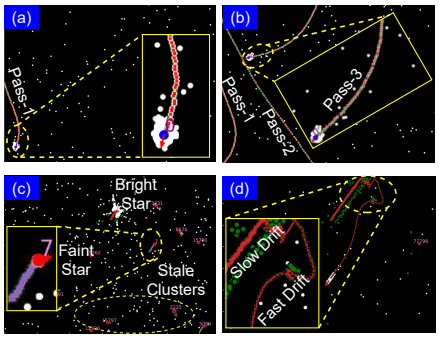

Fig. 4. Telescope Trials: (a–b) three Starlink passes. Frames are 2 ms event accumulations. (c) stellar field with bright and faint stars with slow drift (telescope stationary); (d) fast, curved motion during deliberate slews. Our tracker maintains lock across slow→fast manoeuvres. **Overlay:** CT–EKF red, frame-based tracking green, events white.

Computational Complexity. With a square cell of size s, we get $C_x = \lceil \frac{W}{s} \rceil \times C_y = \lceil \frac{H}{s} \rceil$ cells, each with cell cap K, each event probes only $B = (2r+1)^2$ neighboring cells. With $n_s = 5$ dimensional CT-EKF state, the per-event processing latency is $T_{event} = O(BK) + O(n_s^3) \approx$ const, independent of active clusters M and $W \times H$ as long as per-cell occupancy $\leq K$.

Memory. Each track stores $O(n_s + n_s^2)$ values (e.g., 30 for n_s=5) along with small metadata. The grid holds $O(C_x C_y)$ cells with $O(M)$ total pointers for M active clusters. Thus, memory scales linearly with M.

5. CONCLUSIONS

We introduced a real-time, plug-and-play, high-speed asynchronous multi-target edge computing platform for SSA that operates directly on events, with a CT–EKF tracker delivering near-constant per-event computational complexity on an RPi 4. The system localizes fast targets (>17000 px/s) with sub-μs EKF latency (~ 0.7 μs/event). The algorithm simultaneously tracks bright and faint stars across slow/fast drift regimes and their transitions without losing tracking. Our EKF-based formulation supports high event rates; however, the current prototype targets sparse SSA regimes and operates reliably at rates <1 MeV/s. In denser fields, event rates >1 Mev/s can lead to event drops and degraded performance due to the current single-threaded implementation. Parallelizing the clustering–EKF steps and FPGA offloading are natural next steps to raise the supported event-rate ceiling. The demo of our implementation is available here: Demo

6. ACKNOWLEDGMENT

The authors thank the organisers of the Bangalore Neuromorphic Engineering Workshop 2025 (BNEW25), where this idea originated and early prototypes were discussed.

7. REFERENCES

[1] B. Wang, S. Li, J. Mu, X. Hao, W. Zhu, and J. Hu, "Research advancements in key technologies for space-based situational awareness," *Space: Science & Technology*, 2022.

[2] J. N. Pelton, "Tracking of orbital debris and avoidance of satellite collisions," in *Handbook of Satellite Applications*. Springer, 2016, pp. 1–13.

[3] R. Goldsmith, K. Phillips, T. Jones, E. Trachtman, P. McGaugh, and H. Walden, "Global data relay for leo spacecraft using inmarsat's bgan service," in *Proc. 29th AIAA Int. Communications Satellite Systems Conf. (ICSSC)*, 2011.

[4] European Space Agency, "Annual space environment report," ESA Space Debris Office, Tech. Rep., 2025.

[5] K. L. Hobbs and E. M. Feron, "A taxonomy for aerospace collision avoidance with implications for automation in space traffic management," in *Proc. AIAA SciTech 2020 Forum*, 2020.

[6] E.-H. Kim, H.-D. Kim, and H.-J. Kim, "Optimal solution of collision avoidance maneuver with multiple space debris," *Journal of Space Operations*, vol. 9, no. 3, pp. 20–31, 2012.

[7] N. Bourriez, A. Loizeau, and A. F. Abdin, "Spacecraft autonomous decision-planning for collision avoidance: A reinforcement learning approach," 2023, arXiv:2310.18966.

[8] J. Sharma, G. H. Stokes, C. von Braun, G. Zollinger, and A. J. Wiseman, "Toward operational space-based space surveillance," *Lincoln Laboratory Journal*, vol. 13, no. 2, pp. 309–334, 2002.

[9] K. Watanabe-Brouillette, O. Daigle, Y. Gosselin, E. Beaulieu, and S. Thibault, "Emccd for future sda applications," in *Sensors and Systems for Space Applications XV*, ser. Proc. SPIE, vol. 12121. SPIE, 2022, pp. 141–152.

[10] L. Robinson and C. Frueh, "A ccd/cmos telescope digital twin for space situational awareness," *Advances in Space Research*, vol. 76, no. 5, pp. 3074–3097, 2025.

[11] S. S. Yadav, N. Roy, and C. S. Thakur, "Enhancing celestial imaging: High dynamic range with neuromorphic detectors," *iScience Notes*, vol. 9, no. 9, pp. 1–3, 2024.

[12] S. S. Yadav, B. Pradhan, K. R. Ajudiya, T. S. Kumar, N. Roy, A. Van Schaik, and C. S. Thakur, "Neuromorphic cameras in astronomy: Unveiling the future of celestial imaging beyond conventional limits," 2025, arXiv:2503.15883.

[13] A. Lakshmi, A. Chakraborty, and C. S. Thakur, "Neuromorphic vision: From sensors to event-based algorithms," *Wiley Interdiscip. Rev.: Data Min. Knowl. Discov.*, vol. 9, no. 4, p. e1310, 2019.

[14] G. Cohen, S. Afshar, B. Morreale, T. Bessell, A. Wabnitz, M. Rutten, and A. Van Schaik, "Event-based sensing for space situational awareness," *The Journal of the Astronautical Sciences*, vol. 66, no. 2, pp. 125–141, 2019.

[15] B. Ramesh, S. Zhang, Z. W. Lee, Z. Gao, G. Orchard, and C. Xiang, "Long-term object tracking with a moving event camera," in *Proc. British Machine Vision Conf. (BMVC)*, 2018, p. 241.

[16] H. Chen, D. Suter, Q. Wu, and H. Wang, "End-to-end learning of object motion estimation from retinal events for event-based object tracking," in *Proc. AAAI Conf. Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 10 534–10 541.

[17] N. Messikommer, C. Fang, M. Gehrig, and D. Scaramuzza, "Data-driven feature tracking for event cameras," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 5642–5651.

[18] J. Tao, Y. Cao, and M. Ding, "Sdebrisnet: A spatial–temporal saliency network for space debris detection," *Applied Sciences*, vol. 13, no. 8, p. 4955, 2023.

[19] Z. Wang, T. Molloy, P. Van Goor, and R. Mahony, "Asynchronous blob tracker for event cameras," *IEEE Trans. Robotics*, 2024.

[20] Y. Lu, K. Cui, Y. Shi, Z. Li, J. Li, W. Lu, Y. Zheng, and T. T.-H. Kim, "A memory-efficient high-speed event-based object tracking system," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[21] Y. Lu, Y. Shi, Z. Li, J. Li, and T. T.-H. Kim, "A real-time event-vision sensor-based object detection and tracking system for edge applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2024, pp. 1–3.

[22] M. Litzenberger, C. Posch, D. Bauer, A. N. Belbachir, P. Schön, B. Kohn, and H. Garn, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *Proc. IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop*, 2006, pp. 173–178.

[23] J. P. Rodríguez-Gomez, A. G. Eguíluz, J. R. Martínez-de Dios, and A. Ollero, "Asynchronous event-based clustering and tracking for intrusion monitoring in uas," pp. 8518–8524, 2020.

[24] A. Linares-Barranco, F. Gómez-Rodríguez, V. Villanueva, L. Longinotti, and T. Delbrück, "A usb 3.0 fpga event-based filtering and tracking framework for dynamic vision sensors," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2015, pp. 2417–2420.

[25] C. Cabriel, T. Monfort, C. G. Specht, and I. Izeddin, "Event-based vision sensor for fast and dense single-molecule localization microscopy," *Nature Photonics*, vol. 17, no. 12, pp. 1105–1113, 2023.

[26] Sony Semiconductor Solutions Corporation, "Imx636-aamr-c product brief," 2024, accessed: 2025-09-18. Event-based vision sensor (Gen4.1) co-developed with Prophesee; 1280×720. [Online]. Available: https://www.prophesee.ai/wp-content/uploads/2024/09/IMX636-AAMR-C-Product-Brief-2024.pdf

[27] SensorsINI, "jAER: Java tools for address-event representation (aer) neuromorphic processing," 2007, accessed: 2025-09-17. [Online]. Available: https://github.com/SensorsINI/jaer

[28] Prophesee, "Prophesee software tool for generic object tracking," 2021, accessed: 2025-09-17. [Online]. Available: https://docs.prophesee.ai/stable/samples/modules/analytics/tracking_generic_cpp.html