# MER-Tracker: Towards High-Speed 3D Point Tracking via Multi-View Event-RGB Hybrid Cameras

Supplementary Material

## 8. Additional Technical Details

### 8.1. Working Principle of Event Camera

Event camera is a type of bio-inspired sensor that can asynchronously record intensity changes. In contrast to conventional cameras that are restricted to sequentially produce frames at a fixed frame rate, event cameras asynchronously trigger events in each pixel when their intensity change exceeds a constant threshold, featuring properties such as low latency and high dynamic range. Formally, let $\mathbf{I}_{xy}(t)$ denote the instantaneous intensity at pixel coordinate $(x, y)$ at time $t$, and $\mathbf{L}_{xy}(t)$ denotes its logarithm. An event $p = \pm 1$ will be triggered whenever the change of $\mathbf{L}_{xy}(t)$ surpasses the threshold $c$, where the polarity represents the direction (increase or decrease) of changes. Let $\delta_{t_0}(t)$ be the impulse function at time $t_0$ with a unit integral, the event can therefore be expressed as a continuous-time signal $\mathbf{e}_{xy}(t) = p\,\delta_{t_0}(t)$, where $t_0$ signifies the time at which the event occurs. Then, the proportional intensity change during a time interval $[s, t]$ can be computed as the integral of events that occurred between times $s$ and $t$, expressed as $\mathbf{E}_{xy}(t) = \int_s^t \mathbf{e}_{xy}(h)dh$. Given that each pixel can be treated separately in the event camera, the subscripts can be omitted:

$$\mathbf{E}(t) = \int_s^t \mathbf{e}(h)dh. \tag{10}$$

We can then represent the logarithmic intensity change as: $\mathbf{L}(t) - \mathbf{L}(s) = c\,\mathbf{E}(t)$, rewrite as $\mathbf{L}(t) = \mathbf{L}(s) + c\,\mathbf{E}(t)$, and subsequently obtain the actual intensity change:

$$\mathbf{I}(t) = \mathbf{I}(s) \cdot \exp(c\,\mathbf{E}(t)). \tag{11}$$

Therefore, when an image $\mathbf{I}(s)$ is captured at time $s$, and the event stream is recorded during the time interval $[s, t]$, the image $\mathbf{I}(t)$ can be obtained by warping $\mathbf{I}(s)$ using Eq. 11.

### 8.2. Dual-modal 2D Feature Extractor

The optical camera provides discrete images $\mathcal{I}$ at a low frame rate, while the event camera records a continuous asynchronous event stream $\mathcal{E}$. Therefore, we first perform temporal alignment between the two modalities, followed by separate 2D feature extraction for each.

Firstly, we convert the event streams into an event voxel grid for subsequent processing and temporal alignment. For a time sequence of $(t_{start}, t_{end})$, the asynchronous event streams during this period could be converted into the event voxel grid for subsequent processing and temporal alignment. The event grid $V$ has a shape of $\mathbb{R}^{B\times W\times H}$, where $W \times H$ is the spatial dimension, and $B$ is the number of temporal bins ($B$=5 in the following experiments). The bin position $t_j^*$ for event timestamp $t_j$ could be computed by:

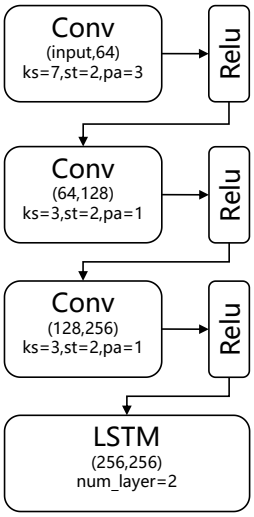


Figure 6. The network of event stream extraction. We build a three-layer convolution network and a long short-term memory network. $ks$ means the kernel size, $s$ means the stride, $pa$ means the padding parameter.

$$t_j^* = \frac{B-1}{t_{end} - t_{start}}(t_j - t_{start}). \tag{12}$$

Thus, the voxel grid $V$ for each temporal bin $b$ at the spatial coordinates $(x, y)$ could be calculated by:

$$V(b, x, y) = \sum_{x_j=x, y_j=y} p_j \cdot max(0, 1 - |b - t_j^*|), \tag{13}$$

where $j = \{j | t_{start} < t_j < t_{end}\}$ means using all events during $(t_{start}, t_{end})$. This conversion evenly populates the events in a sliding window into $B$ consecutive and non-overlapping bins, where each event contributes to its two adjacent bins.

Then, for discrete RGB frames, we extract dense appearance features using a convolutional neural network $\phi(I_t) \in \mathbb{R}^{d\times \frac{H}{k}\times \frac{W}{k}}$, where $k$ is set to 4 to downscale the input resolution for computational efficiency. Following [20, 34, 52], we also compute the feature maps at $S$ = 4 different scales, i.e., $\phi(I_t^s) \in \mathbb{R}^{d\times \frac{H}{k2^{s-1}}\times \frac{W}{k2^{s-1}}}, s = 1, ..., S$.

Thirdly, as for continuous event streams, they exhibit (i) strong temporal ordering dependencies and (ii) low texture content per time step. Therefore, simply applying a CNN to a single-frame event is inadequate for capturing detailed features. Therefore, we build a CNN-LSTM network to extract temporally dependent, fine-grained 2D feature maps $\psi(E_m(t_{start}, t_{end}))$ from the event-camera voxel grid:

$$\psi(t_{start}, t_{end}) = \mathrm{CT}(V(t_{start}, t_{end}). \tag{14}$$

Specifically, the CNN-LSTM consists of a three-layer convolution network and a long short-term memory network, as shown in Fig.6. Note that the event features are projected to match the RGB feature dimension.

# 9. Additional Implementation Details

## 9.1. Pipeline Clarification

Here, we provide a detailed description of the full pipeline, with explicit tensor shapes and data flow through the network. We also report the exact dimensions of each tensor during actual execution on the FMV-Kubric Dataset in Table 4, so that readers can more easily follow the pipeline.

**1) Event Input**. The event stream $E_m$ is a set of asynchronous spatio-temporal events captured by the $m$-th event camera with shape $(N, 4)$. $p \in \{+1, -1\}$ is the event polarity indicating whether the intensity at the corresponding pixel increases or decreases.

**2) Event 2D Extraction**. Via voxel grids, $E_m$ is discretized into the shape of $(V, T, B, H, W)$ and then sent to a CNN-LSTM extractor, resulting in a 2D feature map of shape $(V, T, c, h, w)$.

**3) Event 3D Lifting**. The depth maps for event cameras are projected from RGB depth via camera calibration. using these depths, 2D event features are lifted to 3D and merged.

**4) Anchor Sampling**. Anchors are representative key points sampled from the merged RGB–event 3D feature points using farthest point sampling (FPS). As in PointNet++ [53], FPS iteratively selects points by maximizing the minimum distance between sampled points, yielding a balanced spatial distribution while preserving the original point-cloud geometry and structure.

**5) Trajectory Prediction**. 3D anchor features are fed into the Transformer to predict 3D trajectories, producing outputs of shape $(T, P, 3)$.

Table 4. Pipeline hyperparameter details for FMV-Kubric dataset.

| Type | DVS events | Voxel grid | 2D features | 3D traj. |
|---|---|---|---|---|
| Symbol | $(N, 4)$ | $(V, T, B, H, W)$ | $(V, T, c, h, w)$ | $(T, P, 3)$ |
| Value | time-varying | (4,24,5,480,640) | (4,24,256,120,160) | (24,512,3) |

# 10. Additional Experiments

## 10.1. More Ablation Studies

To further validate the effectiveness of our method design, we conduct additional ablation studies.

The first is why we choose CNN–LSTM encoders rather than CNN or other encoders. We replace the CNN-LSTM encoder in our network with CNN (Convolutional Neural Network), GRU (Gated Recurrent Unit), and TCN (Temporal Convolutional Network), as shown in Table 5.

Table 5. Ablation study on encoder architectures.

| Enc. | CNN | +GRU | +TCN | **+LSTM(Ours)** |
|---|---|---|---|---|
| AJ↑ | 70.8 | 71.3 | 71.8 | **72.3** |

The second is the choice of voxel-window size $B$, which is used in converting the event streams into event voxel grids. As shown in Table 6, our voxel-window size (Equal 5) is best.

Table 6. Ablation study on voxel-window size $B$.

| $B$ | 1 | 3 | **5(Ours)** | 10 |
|---|---|---|---|---|
| AJ↑ | 70.5 | 71.7 | **72.3** | 72.0 |

The third is whether our LoRA fine-tuning strategy is effective. We compare with other LoRA fine-tuning strategies in Table 7.

Table 7. Ablation study on fine-tuning strategies.

| Variant | AJ ↑ |
|---|---|
| No fine-tuning | 55.6 |
| LoRA (r=8, all layers) | 71.9 |
| LoRA (r=16, all layers)**(Ours)** | **72.3** |
| LoRA (r=16, last layer only) | 68.7 |

## 10.2. More Comparisons

Owing to the limited space in the main paper, only a small number of comparison methods were included. Here, we further expand the comparisons to include more state-of-the-art methods. Specifically, the compared methods can be grouped into two categories: (1) 3D point tracking methods based on RGB images with frame interpolation, and (2) other 3D point tracking methods based on Event-RGB inputs. For the first category, we additionally compare with recent trackers such as SpatialTracker[46], SpatialTracker-V2 [47], and TAPIP3D [52]. For the second category, since there is currently no event-based method that performs 3D point tracking, we compare against ETAP [13], an event-based 2D point tracker. Following the MVTracker protocol,

we run ETAP [13] per view with visible query points and lift the resulting 2D tracks to 3D using the same intrinsics and depth.

Table 8. Comparison with additional SOTA trackers.

| Method | AJ ↑ |
|---|---|
| SpatialTracker + Traj.Inter | 54.5 |
| SpatialTrackerV2 + Traj.Inter | 58.2 |
| TAPIP3D + Traj.Inter | 60.7 |
| ETAP | 61.2 |
| **Ours** | **72.3** |

### 10.3. V.S. High-FPS RGB-only Pipelines.

Since a major motivation is that event cameras are preferable to very high-frame-rate RGB cameras, a deeper analysis of this trade-off would be valuable. Thus, we also conduct a small-scale experiment on a subset of the FMV-Kubric dataset by modifying the data to replace the DVS streams with high-FPS RGB inputs, evaluating the resulting RGB-only pipeline using MVTracker. Moreover, we also provide a balanced discussion of hardware price, calibration complexity, and maintenance overhead to help clarify why event cameras are genuinely advantageous relative to widely available high-FPS RGB sensors.

Comparisons are reported together with the actual camera bandwidth, latency, and cost as [4] in Tab. 9, showing a moderate cost with competitive performance.

Table 9. Comparison with High-FPS RGB on a subset of FMV-Kubric dataset.

| Sensors | Methods | Bandwidth ↓ | Latency ↓ | Price ↓ | AJ ↑ |
|---|---|---|---|---|---|
| Low-FPS RGB | MVTracker | 15-20 Mb/s | ∼ 30ms | ∼ 500 $ | 58.6 |
| High-FPS RGB | MVTracker | >100 Mb/s | ∼ 50us | ∼ 10000 $ | 68.3 |
| **Ours** (Event + RGB) | Ours | 20-30 Mb/s | ∼ 25us | ∼ 6000 $ | 70.6 |

## 11. Limitations and Future Work

In this paper, we propose the first task for high-speed 3D point tracking using Event-RGB hybrid cameras. However, there still remains some limitations that could be addressed in future work. First, the method relies on a relatively accurate initialization point cloud and per-frame depth estimation. In the real-world dataset, we used VGGT as the annotation source; subsequently, we also experimented with the latest Depth Anything V3 [26] and found that the tracking accuracy could be further improved. Second, in this work, the depth of the event camera is obtained by transforming the RGB-camera depth through camera-pose-based geometric projection. This coordinate transformation inevitably introduces accuracy loss, which may lead to imprecise point tracking, as shown in Fig. 7. In the future, we plan to employ a beam splitter to achieve spatial synchronization between the RGB and DVS cameras. Another


Figure 7. A failure case of a shaking toy caused by inaccurate event-camera depth estimation.

limitation lies in the benchmark itself. Specifically, the current evaluation compares a rendered depth proxy against monocular depth predictions from Depth Anything, which only measures the consistency with the biases of another learned model rather than the true correctness of 3D point tracking. In future work, we plan to establish more realistic depth ground truth, for example, by using onboard LiDAR or computing depth from stereo high-speed cameras. In addition, we plan to expand our real-world datasets to larger scenes with more complex deformations (e.g., full human bodies in high-speed sports) where occlusion and depth ambiguity are heavier. Finally, our current system involves multiple camera groups, making the overall hardware setup overly complex and less favorable for reproducibility. We are therefore exploring how to reduce the number of devices while maintaining strong tracking performance, with the goal of extending the method to broader real-world application scenarios.