EventHands: Real-Time Neural 3D Hand Pose Estimation from an Event Stream –Supplementary Document–

Viktor Rudnev[1] Vladislav Golyanik[1] Jiayi Wang[1] Hans-Peter Seidel[1]
Franziska Mueller[2] Mohamed Elgharib[1] Christian Theobalt[1]
[1]MPI for Informatics, SIC [2]Google Inc.

In this document, we provide more insights on our *EventHands* method and additional experimental results. First, Sec. 1 discusses details about the experiments and further results. Subsequently, we explain the parameters of the Kalman filter in detail (Sec. 2) and the architecture choice for our neural network (Sec. 3). Sec. 4 contains further details about our simulator and the dataset, including the value ranges used for augmentation. The supplementary video can be found at https://4dqv.mpi-inf.mpg.de/EventHands/.

1. Experimental Details

1.1. Baseline Event Representations

Here we describe the baseline event representations used for the ablation study.

Event Occurence Image (EOI). We define $\mathcal{EOI} \in \mathbb{R}^{W\times H\times 2}$ to be the event occurence image which is initialised with zeros at the beginning. Then, for each event in the current window $\mathcal{E} = \{(t_i, x_i, y_i, p_i)\}_{i=1}^{N_k}$, we update the event occurence image by the following assignment:

$$\mathcal{EOI}(x_i, y_i, p_i) = 1. \quad (1)$$

Thus, $\mathcal{EOI}(x_i, y_i, p_i)$ indicates whether an event with polarity p_i has occurred in the window, but it does not consider the temporal event information.

Single-Channel Event Count Image (ECI-S). The single-channel event count image $\mathcal{ECI}$-$\mathcal{S} \in \mathbb{R}^{W\times H}$ counts the number of events that occurred at a given pixel, irrespective of their polarity

$$\mathcal{ECI}\text{-}\mathcal{S}(x, y) = \left|\{e_i \in \mathcal{E} \mid (x, y) = (x_i, y_i)\}\right|, \quad (2)$$

where x_i, y_i is the position of event $e_i \in \mathcal{E}$.

Event Count Image (ECI). Similar to $\mathcal{ECI}$-$\mathcal{S}$, the event count image $\mathcal{ECI} \in \mathbb{R}^{W\times H\times 2}$ also counts the number of events in each pixel, however, it contains one channel for each polarity

$$\mathcal{ECI}(x, y, p) = |\{e_i \in \mathcal{E} \,|\, (x, y) = (x_i, y_i) \wedge p = p_i\}|, \quad (3)$$

where x_i, y_i is the position of event $e_i \in \mathcal{E}$ and p_i is its polarity.

1.2. Slow-Motion Settings

Although the event stream representation is best suited for fast hands, our approach can be adapted, without retraining the model, to also handle slow or stationary hand motions which generate only a small number of events.

Usually, we generate a new LNES with a duration of 100ms every 1ms. In the slow motion setting, there might not always be enough new events to generate a new LNES. When there are fewer than 10 new events since the last generated LNES, we delay generating a new LNES until at least 10 new events have happened. In the case of stationary hands, noise events could eventually accumulate to generate a new LNES frame and cause a random prediction. We detect this degenerate case by checking the average amount of event information in the last 16 LNES frames. The pixel values inside LNES are time stamps and range from 0 (oldest event) to 1 (newest event). We can hence calculate the total amount of event information in each LNES by summing over all LNES pixel values, which gives more weight to more recent events in each LNES. If the average amount of event information over the last 16 LNES is less than 300, we assume the hand is stationary and repeat the last prediction. Lastly, we use an additional Kalman filter with the slow setting (see Section 5.3) to detect when faster motions occur. If its residual error is $\geq$0.7, we switch the main Kalman filter to the fast setting, otherwise we switch it to the slow setting. All values are selected empirically. We show results of this adaptation to slow hands in the supplementary video from 7:30 to 7:45.

1.3. Additional Results

1.3.1 Ablation Studies

In Fig. 1, we show the PCK curves corresponding to the AUC values reported in Table 1 in the main paper. We compare different settings of our method (no filtering, no aug-

mentation) and different event representations on real test data. The proposed method achieves the best result. Please refer to Section 6.2 in the main paper for a more detailed discussion.

1.3.2 Performance of Depth-based Methods

Most commodity depth cameras rely on structured light or time-of-flight techniques to estimate the depth. However, for fast motion scenarios targeted by our method, these techniques produce depth estimates that are severely corrupted with many missing depth values. As shown in Fig. 2, depth-based state-of-the-art methods such as Moon *et al.* [3] cannot handle such artefacts and hence produce erroneous pose estimates.

1.3.3 Qualitative Results

Fig. 3 shows more qualitative results for different subjects that we captured with the DAVIS240C event camera (*EventHands* uses event stream only). Furthermore, we provide results of a network trained with the arm entering the field of view from the bottom in Fig. 4. In this experiment, we use additional 55 hours of generated event stream data for training.

1.3.4 Low-Light Performance

We also annotated a 7 second part of the recorded low-light event stream. The annotations were done the same way as described in Section 6.1, with 236 frames and 1645 keypoints annotated in total. For visual results of our method on the extended material, please refer to the supplementary video (at 07:45).

2. Temporal Filtering

We use a Kalman filter [1] with constant velocity assumption to post-process our raw predictions $\theta \in \mathbb{R}^{12}$. The corresponding state vector $S \in \mathbb{R}^{24}$ is given by

$$S = \begin{bmatrix} \theta_1 & \dot{\theta}_1 & \dots & \theta_{12} & \dot{\theta}_{12} \end{bmatrix}^T, \tag{4}$$

where $\dot{\theta}_i$ is the velocity of i-th parameter θ_i. We model changes in velocities $\dot{\theta}_i$ as independent Gaussian white noise (*i.e.,* temporally uncorrelated). For a given process noise variance σ_P^2, the *discrete white noise covariance matrix operator* produces a block-diagonal covariance matrix

$$\omega(\sigma_P^2) = \sigma_P^2 \begin{pmatrix} W_1 & & \\ & \ddots & \\ & & W_{12} \end{pmatrix}. \tag{5}$$

This matrix models uncertainty in updating both the position θ and velocity $\dot{\theta}$ in the state vector S. In Eq. (5), W_i is the process noise covariance matrix of $[\theta_i, \dot{\theta}_i]$:

$$W_i = \begin{pmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 \\ \frac{1}{2}\Delta t^3 & \Delta t^2 \end{pmatrix}, \tag{6}$$

where Δt is the temporal step size.

3. Choosing Network Architecture

Besides ResNet-18, we examined other base models including VGG-{11,13,16,19} (with batch normalisation) [5], MobileNet v2 [4], ShuffleNet v2 [2], Inception v3 [6], MnasNet [7] and ResNet-34. Out of these models, only MobileNet v2, ResNet-34 and VGG networks produced validation losses comparable to ResNet-18. However, the smallest examined VGG network and ResNet-34 were unable to handle real-time processing at 1000 Hz, which was one of our main goals. Furthermore, while the inference time of MobileNet v2 was faster than that of ResNet-18, we select ResNet-18 as the base model as it has higher prediction accuracy and enables 1000 frames per second.

4. Simulator and Dataset Details

This section provides more details on the implementation of our GPU-based event simulator and the format used for storing our dataset.

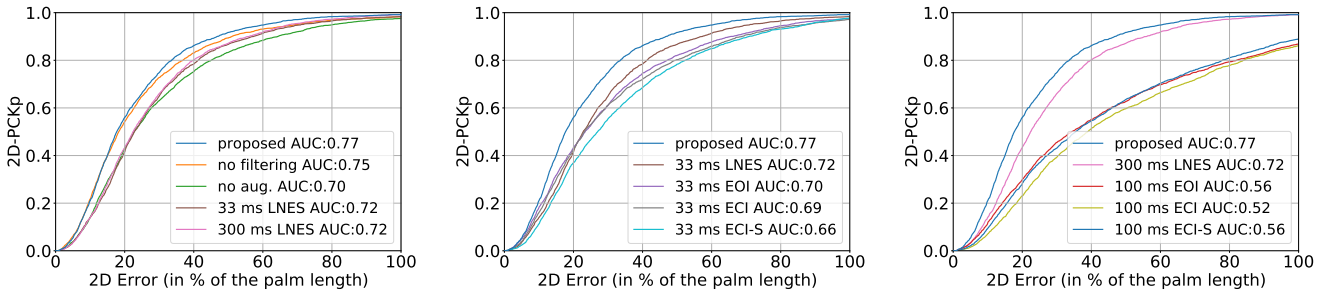
4.1. Implementation Details

The simulator is developed in C++ using

- CUDA, cuBLAS, cuRAND — for computing MANO pose-corrective mesh offsets (that reduce skinning artefacts) on GPU and event camera simulation,
- OpenGL — for posing the skinned and corrected MANO mesh as well as rendering the scene,
- xtensor — for computing MANO shape template mesh and textures on CPU and for loading MANO data, and
- SDL2 — for image loading and OpenGL context management.

All mesh operations that happen every frame, are performed entirely on GPU using GPU memory only. Only two CPU-GPU memory transfers are needed per frame to obtain the current pose vector and the event stream output.

This and other optimisations allow fast simulation of the full SMPL+H body model at 240×180 resolution, which is the resolution of the DAVIS240C event camera that we use for the experiments. On a single NVIDIA GeForce GTX1070, two instances of the simulator can be launched simultaneously to obtain events at rates of around 2000 simulated time steps per second. Considering that we use a time step equal to $1/1000$ of a second, that means we can simulate data at twice the real-time speed with 1 ms temporal resolution.



(a) Removing filtering leads to a comparably small quantitative decrease in performance whereas removing augmentation has a significant impact on real test data. LNES works well with varying temporal window sizes with the proposed 100ms window achieving the best accuracy.

(b) With a temporal window size of 33ms, there is less variation in the performance of the different event representations. Our 33ms LNES still improves over the other event representations while being less accurate than the proposed 100ms LNES.

(c) When increasing the window size to 100ms, the difference between LNES and the other representations increases because the latter do not keep any temporal information within the window. While their performance at 100ms is already significantly degraded, LNES still works with very long windows like 300ms.

Figure 1. Quantitative ablation studies on real data. We plot the percentage of keypoints with an error lower than a given threshold. The PCK curves correspond to the AUC values reported in Table 1 in the main paper.

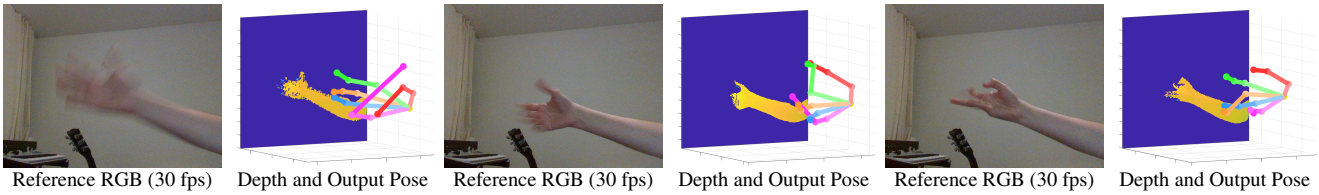


Figure 2. Fast moving hands lead to corrupted depth maps on which depth-based methods, like Moon *et al.* [3], produce large errors. We show the blurry RGB image for reference only as well as the input depth and the predicted 3D hand pose.

4.2. Event Camera Calibration

To reduce the domain gap between the simulated and real event data, we used the event threshold value and the noise event rate of our DAVIS240C event camera.

To calibrate the event threshold C, we shot several sequences by moving an object (a checkerboard) monotonously from one side to the other with different speeds. We captured both the events $\{(t_i, u_i, p_i)\}_{i=1}^{N_{\text{events}}}$ and instant intensity images $\{\Omega_j\}_{j=1}^{N_{\text{images}}}$ simultaneously. By moving monotonously from one side to the other side, we eliminate cases when the event stream contains events that cannot be explained by the intensity images, *e.g.,* events that cancel themselves between two consecutive intensity images. To estimate the event threshold from the captured data, we use the following observation. According to our camera model, if the camera emits N events in total, then the intensity images would have the total log-intensity change of $\approx NC$. Thus, C can be estimated by dividing the total log-intensity change by the total number of events N.

Hence, we counted the total intensity change of the instant intensity images Δ_{total} as

$$\Delta_{\text{total}} = \sum_{i=1}^{N_{\text{images}}-1} |\log(\max\{\Omega_{t+1}, \varepsilon\}) - \log(\max\{\Omega_t, \varepsilon\})|, \tag{7}$$

where $\varepsilon = 10$ is a constant added for numerical stability. Then, we estimate C as

$$C \approx \Delta_{\text{total}} / N_{\text{events}}. \tag{8}$$

For our event camera, we obtain $C = 0.5$–0.55.

To estimate the *noise event rate*, we shot the static background and count the number of positive and negative recorded events. For our DAVIS240C, we estimate the noise to be $\approx$2500 positive and $\approx$100 negative events per second.

4.3. Dataset Format

The generated dataset consists of two files, *i.e.,* the event stream and the metadata stream. The event stream file format is tailored for the frame-by-frame event stream simulation. It consists of blocks of four bytes: two bytes for x coordinate, one byte for y coordinate and one byte for polarity p. At the start, the timestamp is considered to be zero. A

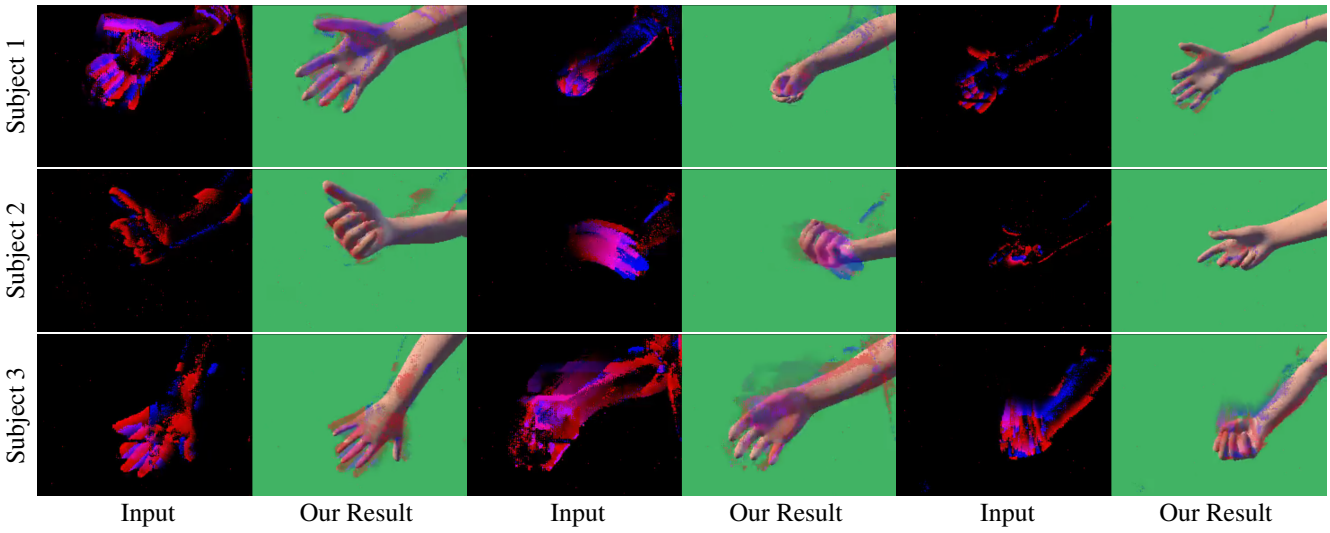


Figure 3. Additional results of *EventHands* on real event sequences captured with different subjects.

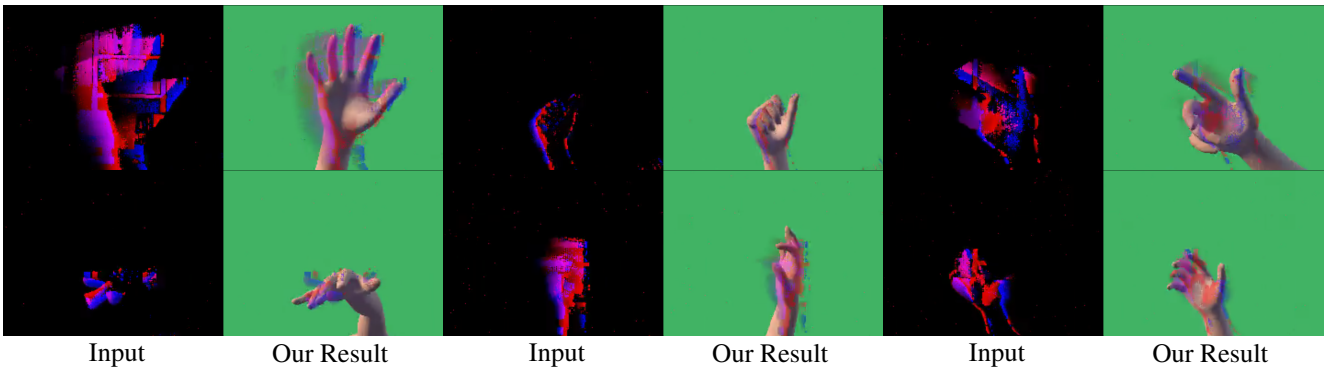


Figure 4. Results of *EventHands* on real data where the hand is entering the frame from the bottom.

new frame is indicated by the polarity value $p = 255$, which signals that the timestamp should be incremented by one time step. We use the time step of $1/1000$ of a second. The file starts with a four-byte integer that specifies the number N of metadata fields per each frame. Then, the stream starts and it consists of $8N+2$ byte blocks. The block contains N eight-byte double-precision reals and two-byte magic. We use $N = 12$ for six MANO articulation coefficients, three components of the hand root translation vector, and three components of the hand root rotation vector.

We also implement a high-speed C++/Python loader for the proposed dataset format. It allows loading $\sim 8.6 \cdot 10^5$ simulated frames per second or $\sim 1.75 \cdot 10^8$ events per second when using storage capable of 1000 MB/s read speeds. With the fixed rate of 1000 simulated frames per second, this amounts to loading 860 simulated seconds per second. Thus, we are able to load a 45-hours-long dataset in just three minutes.

4.4. Simulation Parameters

We next describe how we augment the simulation for generating the event data. SMPL+H body shape β is drawn from $\mathcal{U}[-2, 2]$. Body position θ is drawn as follows. First, we sample $\xi \sim \mathcal{U}[-0.2, 0.2]$. Then $\theta = \xi \odot g + o$, where g is the gain vector, o is the offset vector and $\odot$ is the component-wise multiplication operator.

For the dataset in which the arm comes from the top and right edges, the gain is

$$g_i^{(1)} = \begin{cases} 100, & \text{if } i = 16 \cdot 3 + 9, \\ 40, & \text{if } i = 16 \cdot 3 + 10, \\ 10, & \text{if } i = 16 \cdot 3 + 11, \\ 40, & \text{if } i = 16 \cdot 3 + 15, \\ 40, & \text{if } i = 16 \cdot 3 + 16, \\ 40, & \text{if } i = 16 \cdot 3 + 17, \\ 1, & \text{otherwise,} \end{cases}$$

and the offset is

$$o_j^{(1)} = \begin{cases} 0.2, & \text{if } j = 13 \cdot 3 + 5, \\ 0.1, & \text{if } j = 16 \cdot 3 + 5, \\ 1.4\epsilon, & \text{if } j = 16 \cdot 3 + 9, \\ 0.5, & \text{if } j = 16 \cdot 3 + 11, \\ 0, & \text{otherwise,} \end{cases}$$

where ϵ is sampled randomly and is either -1 or 1 with equal probability. Global translation vector has (x, y) components sampled from $\mathcal{U}[-0.3, 0.3]$. The depth component z is taken from $\mathcal{U}[-0.09, 0.09]$.

For the dataset in which the arm comes from the bottom edge, the gain is

$$g_i^{(2)} = \begin{cases} 100, & \text{if } i = 16 \cdot 3 + 9, \\ 10, & \text{if } i = 16 \cdot 3 + 11, \\ 40, & \text{if } i = 16 \cdot 3 + 15, \\ 40, & \text{if } i = 16 \cdot 3 + 16, \\ 40, & \text{if } i = 16 \cdot 3 + 17, \\ 1, & \text{otherwise,} \end{cases}$$

and the offset is

$$o_j^{(2)} = \begin{cases} -2.3, & \text{if } j = 2, \\ 0.2, & \text{if } j = 13 \cdot 3 + 5, \\ 0.1, & \text{if } j = 16 \cdot 3 + 5, \\ 1.4\epsilon - 3.8, & \text{if } j = 16 \cdot 3 + 9, \\ 0.5, & \text{if } j = 16 \cdot 3 + 11, \\ 0, & \text{otherwise,} \end{cases}$$

where ϵ is also chosen randomly and is either -1 or 1 with equal probability. Global translation vector has x component drawn from $\mathcal{U}[-1.5, -0.9]$, y component taken from $\mathcal{U}[-0.52, 0.08]$ and depth component z taken from $\mathcal{U}[-0.09, 0.09]$.

The hand MANO articulation parameters are sampled from $\mathcal{U}[-2, 2]$, whereas hand texture PCA coefficients are chosen from $\mathcal{N}(0, 4I)$. Light directions are sampled uniformly from all possible directions and light intensities are drawn from $\mathcal{U}[0.9, 1.1]$. Finally, the background image is drawn randomly from the collected set of nine background images. The event generation threshold is drawn from $\mathcal{N}(0.5, 0.0004)$.

References

[1] Roger Labbe. Kalman and bayesian filters in python. https://elec3004.uqcloud.net/2015/tutes/Kalman_and_Bayesian_Filters_in_Python.pdf, 2014. online, accessed on the 11 Dec. 2020. 2

[2] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European conference on computer vision (ECCV)*, 2018. 2

[3] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 3

[4] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[6] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[7] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 2