Supplementary Material for

EVDodge: Embodied AI For High-Speed Dodging On A Quadrotor Using Event Cameras

Nitin J. Sanket[1], Chethan M. Parameshwara[1], Chahat Deep Singh[1], Ashwin V. Kuruttukulam[1], Cornelia Fermüller[1], Davide Scaramuzza[2], Yiannis Aloimonos[1]

S.I. Event Frame $\mathcal{E}$

A traditional grayscale (global-shutter) camera records frames at a fixed frame rate by integrating the number of photons for the chosen shutter time. This is done for all pixels synchronously. In contrast, an event camera only records the polarity of logarithmic brightness changes *asynchronously at each pixel*. If the brightness at time t of a pixel at location $\mathbf{x}$ is given by $I_{t,\mathbf{x}}$, an event is triggered when:

$$\| \log\left(I_{t+1,\mathbf{x}}\right) - \log\left(I_{t,\mathbf{x}}\right) \|_1 \geq \tau$$

Here τ is a threshold which will determine if an event is triggered or not. τ is set at the driver level as a combination of multiple parameters. Each triggered event outputs the following data:

$$\mathbf{e} = \{\mathbf{x}, t, p\}$$

where $p = \pm 1$ denotes the sign of the brightness change. The event data unlike an image can vary in data rate and are generally output as a vector of four numbers per triggered event. The data rate is small when the amount of motion and/or scene contrast are small and large when either the motion or the scene contrast is large. This can be beneficial for asynchronous operation, however on a low power digital processor like the one on-board a micro-quadrotor, this can be appalling. To maintain a near constant computational bottle-neck for event processing, we create event frames denoted as $\mathcal{E}$. An event frame is essentially a collection events triggered in a spatio-temporal window starting at t_0 and a temporal depth of δt. The event frame $\mathcal{E}$ is formed as follows.

$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_+ = \sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = +1\right)$$

$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_- = \sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = -1\right)$$

$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_\tau = \left(\sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = \pm 1\right)\right)^{-1} \mathbb{E}\left(t - t_0\right)$$

Here $\mathbb{1}$ is an indicator function which has a value of 1 for an event triggered with polarity of p. For $\mathcal{E}_+$ the value of p is +1. Here $\mathbb{E}$ is the expectation/averaging operator. Finally $\mathcal{E}_+$, $\mathcal{E}_-$ and $\mathcal{E}_\tau$ are normalized such that minimum and maximum values are scaled between $[0, 1]$. Essentially $\mathcal{E}_+/\mathcal{E}_-$ captures the per-pixel average number of positive/negative event triggers in the spatio-temportal window spanned between t_0 and $t_0 + \delta t$. $\mathcal{E}_\tau$ captures the average trigger time per pixel. This event frame representation is inspired by previous works [1] [2]. The event frame $\mathcal{E}$ is composed by depthwise stacking $\mathcal{E}_+$, $\mathcal{E}_-$ and $\mathcal{E}_\tau$, i.e., $\mathcal{E} = \{\mathcal{E}_+, \mathcal{E}_-, \mathcal{E}_\tau\}$. Using event frames has some pragmatic advantages as compared to processing event by event on the raw event stream.

- The control command can be produced within a constant time bound as event frames are produced at a near constant rate.
- The spatial relationships between event triggers are preserved along with polarity and timing information which is exploited by convolutional neural networks employed in this paper.
- Event frames can be produced in *linear time* in the number of event triggers.

S.II. Loss Functions

In this Section, we present the mathematical formulations for variants of the loss functions used in this work.

The two flavors of the heuristic Contrast function $\mathcal{C}$ used in Section II-A of the main paper is inspired by [3] are given below (denoted by $\mathcal{C}_1$ and $\mathcal{C}_2$).

$$\mathcal{C}_1\left(\mathcal{E}\right) = \mathbb{E}\left(\|\text{Var}\left(\nabla\mathcal{E}\right)\|_1\right)$$
$$\text{Var}\left(\mathcal{E}\right) = \mathbb{E}\left(\mathcal{E}\left(\mathbf{x}\right) - \mathbb{E}\left(\mathcal{E}\right)\right)$$
$$\mathcal{C}_2\left(\mathcal{E}\right) = \mathbb{E}\left(\|\nabla\mathcal{E}\|_1\right)$$

where $\nabla = \begin{bmatrix}\nabla_x & \nabla_y\end{bmatrix}^T$ is the 2D gradient operator (sobel in our case), Var is the variance operator and $\mathbf{x}$ denotes the pixel location. *The key difference from [3] is that we use the variance operator on the gradients instead on raw values as empirically this gave us better results and was more stable during training.*

Mathematical formulations of the different variants of the distance function $\mathcal{D}$ used in Sections II-A, II-B and II-C of the main paper which measures the similarity between two

Nitin J. Sanket and Chethan M. Parameshwara contributed equally to this work. (Corresponding author: Nitin J. Sanket.)

[1]Perception and Robotics Group, University of Maryland Institute for Advanced Computer Studies, University of Maryland, College Park.

[2]Robotics and Perception Group, Dep. of Informatics, University of Zurich, and Dep. of Neuroinformatics, University of Zurich and ETH Zurich.

event frames are given below (denoted as $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$).

$$\mathcal{D}_1\left(\mathcal{E}_1, \mathcal{E}_2\right) = \mathbb{E}\left(\|\mathcal{E}_1 - \mathcal{E}_2\|_1\right)$$

$$\mathcal{D}_2\left(\mathcal{E}_1, \mathcal{E}_2 | \, \alpha, \epsilon\right) = \mathbb{E}\left(\left(\left(\mathcal{E}_1 - \mathcal{E}_2\right)^2 + \epsilon^2\right)^{\alpha}\right)$$

$$\mathcal{D}_3\left(\mathcal{E}_1, \mathcal{E}_2 | \, \alpha, \epsilon, c\right) = \frac{b}{d}\left(\left(\frac{\left(\left(\mathcal{E}_1 - \mathcal{E}_2\right)/c\right)^2}{b} + 1\right)^{d/2} - 1\right)$$

$$b = \|2 - \hat{\alpha}\|_1 + \epsilon; \qquad d = \begin{cases} \hat{\alpha} + \epsilon & \text{if } \hat{\alpha} \geq 0 \\ \hat{\alpha} - \epsilon & \text{if } \hat{\alpha} < 0 \end{cases}$$

$$\hat{\alpha}_i = \left(2 - 2\epsilon_\alpha\right) \frac{e^{\alpha_i}}{e^{\alpha_i} + 1} \quad \forall i$$

Here, $\mathcal{D}_1$ is the generic l_1 photometric loss [4] commonly used for traditional images, $\mathcal{D}_2$ is the Chabonnier loss [5] commonly used for optical flow estimation for traditional images and $\mathcal{D}_3$ is the robust loss function presented in [6]. In $\mathcal{D}_3$, the value of α is output from the network (Refer to S.III for architecture details).

S.III. Network Details

In this Section, we will present the information on network architecture and training details.

The network architecture is shown in Fig. S1. Notice the simplicity in our network owing our performance to the approach of stacking multiple shallow networks to obtain good performance. *It must be noted that using advanced architectures might lead to better performance.* We leave this as an avenue for future work.

EVDeblurNet was trained for 200 epochs with a learning rate of 10^{-3} for 200 epochs with a batch size of 256 for losses using $\mathcal{D}_1$ and $\mathcal{D}_2$ and with a batch size of 32 for losses using $\mathcal{D}_3$. Also, the loss part associated with the contrast is scaled by a factor of 2.0 and the loss part associated with the distance is scaled by a factor of 1.0. This is equivalent to setting $\lambda = 0.33$.

EVSegNet, EVFlowNet, and EVSegFlowNet were trained for 50 epochs with a learning rate of 10^{-4} and a batch size of 256. EVHomographyNet was trained for 200 epochs with learning rate 10^{-4} and a batch size of 256.

For all the networks, the event frames $\mathcal{E}$ were normalized by dividing each pixel value by 255 and then subtracting by 0.5 and finally scaling by 2.0 to bound all values between $[-1, 1]$.

For networks using $\mathcal{D}_3$ loss, the values of α are output by the networks last n channels. Here n denotes the number of input channels per image (3 in our case of event frame). $N = 2n$ when using $\mathcal{D}_3$ loss.

S.IV. Multi Moving Object Event Dataset

Extensive and growing research on visual (inertial) odometry or SLAM have lead to the development of a large number of datasets. Recent adaption of deep learning to solve these aforementioned problems have fostered the development of large scale datasets (large amount of data). However, most of these datasets are built with the fundamental assumption of static scenes in mind and as a manifestation of which moving or dynamic objects are often not included in these datasets [7]–[9].

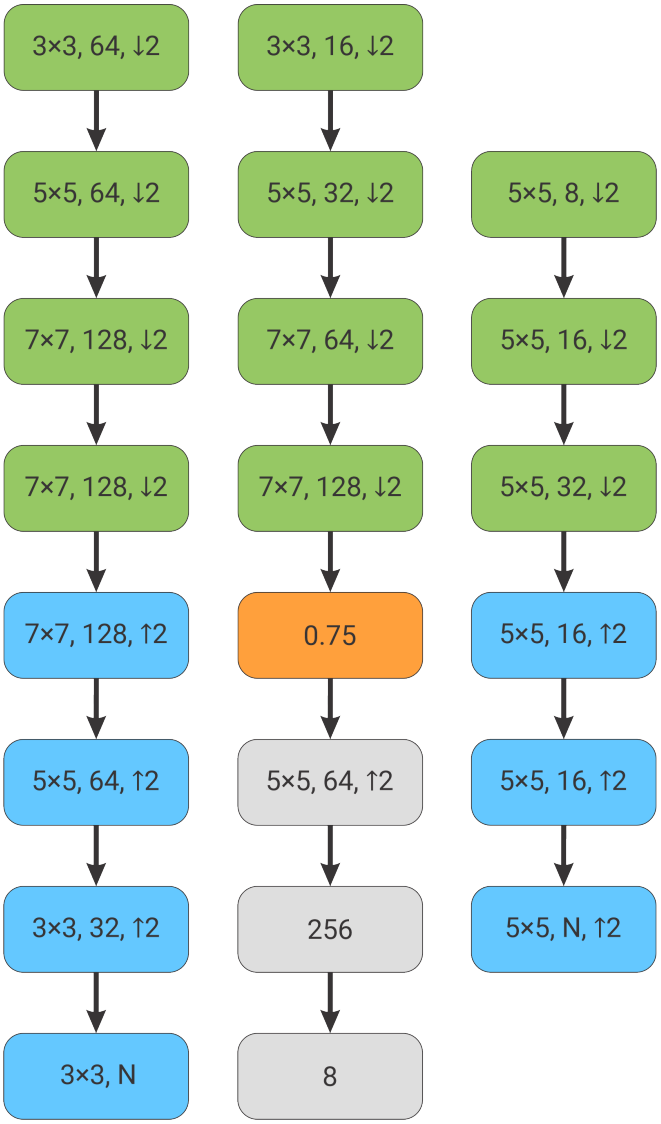

Figure S1. Network Architectures used in the proposed pipeline. Left: EVDeblurNet, Middle: EVHomographyNet and Right: EVSegFlowNet. Green blocks show the convolutional layer with batch normalization and ReLU activation, cyan blocks show deconvolutional layer with batch normalization and ReLU activation and orange blocks show dropout layers. The numbers inside convolutional and deconvolutional layers show kernel size, number of filters and stride factor. The number inside dropout layer shows the dropout fraction. N is 3 and 6 respectively for EVDeblurNet when using losses $\mathcal{D}_1/\mathcal{D}_2$ and $\mathcal{D}_3$. N is 2 and 5 respectively for EVSegFlowNet when using losses $\mathcal{D}_1/\mathcal{D}_2$ and $\mathcal{D}_3$.

To this end, we propose to use synthetic scenes for generating "unlimited" amount of training data with one or more moving objects in the scene. We accomplish this by adapting and proliferating the simulator presented in [8]. To incubate generalization to novel scenes and to utilize the algorithm trained on simulation directly in the real world, we create synthetic moving objects which vary significantly in their texture, shape and trajectory. We also choose random textures for the walls of the 3D room in which objects will

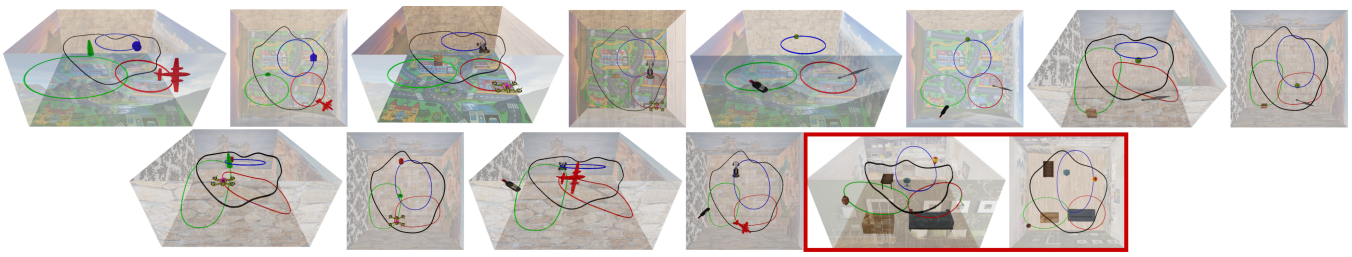

Figure S2. Various Scene setups used for generating data. Red box indicates the scene used for generating out of dataset testing data to evaluate generalization to novel scenes.

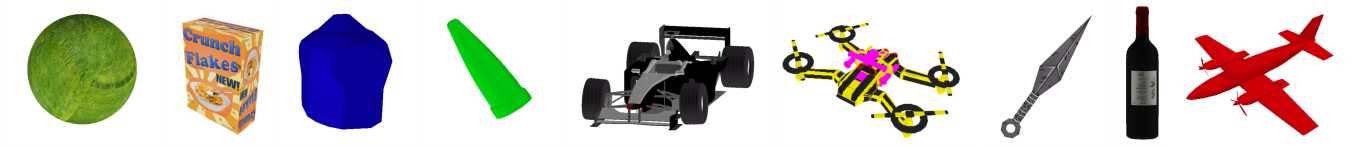

Figure S3. Moving objects used in our simulation environment. Left to right: ball, cereal box, tower, cone, car, drone, kunai, wine bottle and airplane. Notice the variation in texture, color and shape. *Note that the objects are not presented to scale for visual clarity.*

Figure S4. Random textures used in our simulation environment

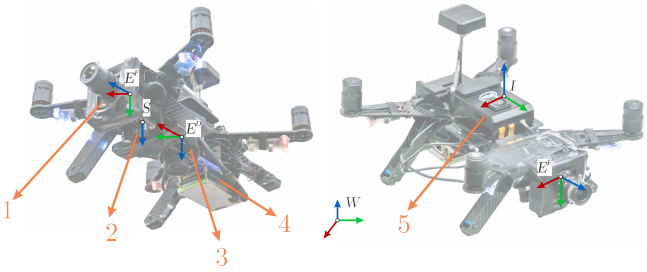

Figure S5. Representation of coordinate frames on the hardware platform used. (1) Front facing DAVIS 240C, (2) down facing sonar on PX4Flow, (3) down facing DAVIS 240B, (4) NVIDIA TX2 CPU+GPU, (5) Intel® Aero Compute board.

move about.

To generate data, we randomize wall textures, objects and object/camera trajectories to obtain seven unique configurations out of which one is exclusively used for test of generalization on more complex structures. Each configuration has a room with three objects moving as shown in Fig. S2. Images are rendered at 1000 frames per second at a resolution of 346×260 and a field of view of 90° for each configuration. Using these images, events are generated following the approach described in [8]. Later event frames $\mathcal{E}$ are generated with three different integration times δt of $\{1, 5, 10\}$ ms. Details about the room, lighting and objects are given next.

A. 3D room and moving objects

Each room is of size $10 \times 10 \times 5$ m and has random textures on all the walls. These random textures consist of different patterns, colors and shapes. These textures mimic those which occur in real-world indoor and outdoor environments such as skyscrapers, flowers, landscape, bricks, wood, stone and carpet. Each room contains seven light sources inside it for uniform illumination.

The camera is moved inside the 3D room on trajectories such that almost all possible combinations of rotation and translation are obtained. This is aimed at replicating the movement which could be encountered on a real quadrotor.

We have three Independently Moving Objects (IMOs) in each room. Each object is unique in color, shape, texture and size. The objects are chosen to range from simple shapes and textures to complex ones. The objects chosen are ball, cereal box, tower, cone, car, drone, kunai, wine bottle and airplane. The trajectories of the objects are chosen such that many different combinations of relative pose between the camera and the objects are encountered. Also, the objects are moving ten times faster than the camera simulating objects being thrown at a hovering or a slow moving (drifting) quadrotor. The wall textures and moving objects are shown in Figs. S3 and S4 respectively.

Figure S6. Different textured carpets laid on the ground during real experiments to aid robust homography estimation from EVHomographyNet.

B. Dataset for EVDeblurNet

To learn a simple deblur function, we obtaian data from a down facing camera looking at a planar texture and such that no moving objects appear in the frame. The textures for the floor are chosen to replicate the common floor patterns such as wooden flooring, stone, kid's play carpet, and tiles. A total of 15K event frames corresponding to five different textures and integration times δt of $\{1, 5, 10\}$ ms are obtained. Random crops of 128×128 are used to train the network.

C. Dataset for EVSegNet, EVFlowNet and EVSegFlowNet

In this dataset, the camera follows the same trajectory given in Section S.IV.B to capture the moving objects in the 3D environment. The camera is moving approximately at 0.005 m per frame and moving objects are ranging from 0.05 to 0.06 m per frame. There are some instances where moving objects collide with the camera. We specifically included these scenarios in the dataset so that the learning approaches could learn utilizing both small and large changes in object appearances between consecutive event frames. Average of two objects per frame is captured from the camera (minimum of 0 objects to maximum of 3 objects). Using the six scenarios, 70K event frames are obtained (including data from integration times of $\{1, 5, 10\}$ ms). Event frames from two random integration times per scenario are chosen for training. We obtain 60K images for training and 10K images for testing (we only use 1 ms data not used for training for testing due to mask alignment errors at higher integration times). During training, we use frame skips of one to four to faciliate variable baseline learning of flow and segmentation. We call this test set as "in dataset" testing because the test set though differs significantly in appearance due to integration times still contains the same objects and textures as the training set.

For measuring the amount of generalization of our approach, we created a more complex and completely different scenario for testing. This scenario contains immovable 3D objects such as table, chair and a box to 3D room with different textures (different per wall and different from training set textures). The textures on the wall are more realistic depicting a real indoor environment (Refer to the scenario in the red box in Fig. 2). We particularly designed such a scene to highlight that our network is mostly learning from contours and motion information of the objects which is agnostic to scene appearance. Here, we use integration times δt of $\{1, 2\}$ ms. We obtain 6K frames for "out of dataset" testing.

D. Dataset for EVHomographyNet

To train EVHomographyNet, we use the same training set from EVSegFlowNet with the major difference being that here only one frame is used at a time. A random patch of size 128×128 is obtained from the 346×260 frame. Then a random perturbation between $\pm\gamma$ is applied to each of the corners. This is used to obtain the homography warped event frame. This approach is exactly the same as given in [10], [11]. For testing "in dataset" the center crops of all the images used for training are chosen and random perturbations of different $\pm\gamma$ are applied. (Refer to Table I of the main paper).

For "out of dataset" testing a similar treatment is given to the out of dataset used for evaluating EVSegFlowNet.

S.V. Experimental Setup

The proposed framework was tested on a modified Intel® Aero Ready to Fly Drone. The Aero platform was selected for its rugged carbon fiber chassis and integrated flight controller running the PX4 flight stack.

For our experiments, we mounted a front facing DAVIS 240C event camera mated to a 3.3 - 10.5 mm $f/1.4$ lens set at 3.3 mm giving us a diagonal Field Of View (FOV) of 84.5°, a downfacing DAVIS 240B event camera mated to a 4.5 mm $f/1.4$ lens giving us a diagonal FOV of 67.4° and a down facing PX4Flow sensor for altitude measurements. Additionally, we obtain inertial measurments from the IMU on the flight controller. We also mounted an NVIDIA Jetson TX2 GPU to run all the perception and control algorithms on-board (Fig. S5). All the communications happen over serial port or USB. The takeoff weight of the flight setup including the battery is 1400 g with dimensions being $330 \times 290 \times 230$ mm. This gives us a maximum thrust to weight ratio of 1.35.

All the neural networks were prototyped on a PC running Ubuntu 16.04 with and Intel® Core i7 6850K 3.6GHz CPU,

an NVIDIA Titan-Xp GPU and 64GB of RAM in Python 2.7 using TensorFlow 1.12. The final code runs on-board the NVIDIA Jetson TX2 running Linux for Tegra® (L4T) 28.1. All the drivers for creating event frames and sensor fusion are written in C++ for efficiency and all the neural network codes run on the TX2's GPU in Python 2.7. We obtain a flight time of about 3 mins.

To enable robust homography estimation, we laid down carpets of different textures on the ground to obtain strong contours in event frames (Refer to Fig. S6).

S.VI. Compression Achieved by using EVSegFlowNet

Now, let's analyze the complexity of EVSegFlowNet as compared to a combination of separate segmentation and flow networks we call EVSegNet and EVFlowNet respectively. Let $\mathcal{O}$ denote the complexity measure as the minimum number of neurons to obtain a satisfactory generalization performance on a specific task. We also assume that for smaller and shallow networks complexity scales with number of neurons almost linearly. The complexity for a combination of EVSegNet and EVFlowNet is given by $\mathcal{O}(S)+\mathcal{O}(F)$. Let the complexity of segmentation and flow obtained by EVSegFlowNet be $\mathcal{O}(\widetilde{S})$, $\mathcal{O}(\widetilde{F})$ respectively. A compression/speedup is achieved when $\frac{\mathcal{O}(\widetilde{S})+\mathcal{O}(\widetilde{F})}{\mathcal{O}(S)+\mathcal{O}(F)} < 1$. Now, because we are only estimating flow for foreground pixels in EVSegFlowNet we have $\frac{\mathcal{O}(\widetilde{F})}{\mathcal{O}(F)} \approx \frac{\overline{\overline{\mathcal{F}}}}{\overline{\overline{\mathcal{B}}}}$. Also, as we mentioned before we get segmentation for free from EVSegFlowNet hence $\mathcal{O}(\widetilde{S}) \approx 0 \ll \mathcal{O}(S)$. This also implies that we achieved good compression/speedup by our formulation as $\frac{\mathcal{O}(\widetilde{S})+\mathcal{O}(\widetilde{F})}{\mathcal{O}(S)+\mathcal{O}(F)} \ll 1$.

References

[1] A. Mitrokhin, C. Fermüller, C. Parameshwara, and Y. Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, Oct 2018.

[2] Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018.

[3] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[4] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

[5] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[6] Jonathan T. Barron. A general and adaptive robust loss function. *CVPR*, 2019.

[7] Wenbin Li, Sajad Saeedi, John McCormac, Ronald Clark, Dimos Tzoumanikas, Qing Ye, Yuzhong Huang, Rui Tang, and Stefan Leutenegger. Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *British Machine Vision Conference (BMVC)*, 2018.

[8] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, October 2018.

[9] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, July 2018.

[10] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016.

[11] Ty Nguyen, Steven W Chen, Shreyas S Shivakumar, Camillo Jose Taylor, and Vijay Kumar. Unsupervised deep homography: A fast and robust homography estimation model. *IEEE Robotics and Automation Letters*, 3(3):2346–2353, 2018.