EDFLOW: Event Driven Optical Flow Camera With Keypoint Detection and Adaptive Block Matching

Min Liu and Tobi Delbruck, Fellow, IEEE

Abstract—Event cameras such as the Dynamic Vision Sensor (DVS) are useful because of their low latency, sparse output, and high dynamic range. In this paper, we propose a DVS+FPGA camera platform and use it to demonstrate the hardware implementation of event-based corner keypoint detection and adaptive block-matching optical flow. To adapt sample rate dynamically, events are accumulated in event slices using the area event count slice exposure method. The area event count is feedback controlled by the average optical flow matching distance. Corners are detected by streaks of accumulated events on event slice rings of radius 3 and 4 pixels. Corner detection takes about 6 clock cycles (16 MHz event rate at the 100MHz clock frequency) At the corners, flow vectors are computed in 100 clock cycles (1 MHz event rate). The multiscale block match size is 25×25 pixels and the flow vectors span up to 30-pixel match distance. The FPGA processes the sum-of-absolute distance block matching at 123 GOP/s, the equivalent of 1230 Op/clock cycle. EDFLOW is several times more accurate on MVSEC drone and driving optical flow benchmarking sequences than the previous best DVS FPGA optical flow implementation, and achieves similar accuracy to the CNN-based EV-Flownet, although it burns about 100 times less power. The EDFLOW design and benchmarking videos are available at https://sites.google.com/view/edflow21/home.

Index Terms—Dynamic vision sensor, FPGA, near-sensor processing.

I. INTRODUCTION

MEASURING Optical Flow (OF) has been a goal of computer vision since the 1960s. OF is widely used in a variety of applications including segmentation [1], angular velocity estimation [2], tracking [3], [4], obstacle avoidance and altitude control [5], [6], aircraft landing [7], [8], robot navigation [9], and visual odometry [10].

Popular frame-based OF methods like Lucas-Kanade (LK) [11] can work well with small displacement motion, but poorly with large displacements on edges or on complex textures. The fixed sample rate of frames makes it difficult to get small displacement image pairs when the scene is moving fast and slow movements result in tiny displacements that result in imprecise flow. Conventional methods also fail when the images are over or underexposed, and when the images are too blurred to extract good features.

Event cameras based on the Dynamic Vision Sensor (DVS) output asynchronous brightness change events [12]. Some event camera types can also concurrently output intensity samples [13]–[18]. The DVS output is a variable data-rate stream of timestamped pixel brightness change events. The DVS large dynamic range (120db), sub-millisecond time resolution, sparse output, and relatively low power consumption (≈10 mW at the die level, ≈1W at the USB camera level) make it an attractive alternative to conventional imagers for many moving object or moving camera applications. Since the DVS provides high dynamic range and activity-driven brightness change events, it makes sense to consider using it for computing OF.

The measurement of OF is more robust at the locations of corner-like features called keypoints than along edges, where the component of motion along the edges are not constrained. At keypoints, the OF can be unambiguously determined. Measurement of OF using keypoints measures OF only at these reliable keypoints. These same keypoints are used in semidense Visual Odometry (VOD) and Simultaneous Localization And Mapping (SLAM) pipelines, and OF can help the matching processing converge faster and more reliably.

A local gradient measurement such as that used by LK cannot account for nonlocal structures. For better OF estimation, video compression algorithms use Block Matching Optical Flow (BMOF) to match blocks of pixels between frames using a hierarchical coarse to fine search. The company LSI Logic in their DoMiNo™ and other companies used BMOF in mass production video compression architectures since the 1990s [19], [20]. BMOF is not popular for software OF because it requires many operations that must be computed serially on a CPU (See Sec. VI).

The key requirement for BMOF with DVS events is to collect DVS frames with high quality features. Our Adaptive Block Matching Optical Flow (ABMOF) algorithm was introduced to adaptively vary the DVS event slice exposure for good block matching quality. ABMOF benefits from DVS high temporal resolution, low latency, high dynamic range, and activity-driven computation.

Fig. 1 illustrates our EDFLOW camera processing steps. The EDFLOW camera, which we built in this work, outputs, along with the original brightness change events, a stream of keypoints and the OF of these keypoints. It consists of four main parts: DVS chip (a), keypoint detector (b-d), OF estimator (d-e), and feedback control of slice duration (g).
Keypoint events drive OF estimation at informative locations and thus reduce the data rate for OF estimation so that even high event rates can be processed in real time without skipping informative OF events.

Our method uses a novel algorithm called Slice-based FAST (SFAST) for corner detection and ABMOF [21] for OF estimation. Our main contributions are:

- A powerful EDFLOW event camera (Sec. V-A, Fig. 8) that extracts OF and keypoints for real-time applications. It combines several techniques in scheduling and Hardware Description Language (HDL) level (e.g., data partitioning, on-the-fly processing with minimal buffers, deep pipelining, parallelization at bit-/data-/instruction-/task-level) for efficient circuit design.
- A hardware corner detection method (Sec. IV-B) that is more accurate and more compatible with ABMOF than other methods.
- An open-source hardware implementation of combined corner detection and full multiscale adaptive event slice optical flow (Sec. IV-C), which together improve the accuracy of optical flow estimation (Sec. V-C) and increase the throughput (Sec. IV).

The rest of this paper is organized as follows. Sec. II reviews event-based OF and event-based corner detectors. Sec. III describes the algorithms and Sec. IV explains the hardware implementation details. Sec. V shows the experimental results. Sec. VI discusses our results and concludes the paper. Our Supplementary Material (SM) includes videos and additional details of the logic circuits and experimental results; readers are also invited to the EDFLOW21 project website.

II. RELATED WORK

A review of event-based OF and event-based corner detection is introduced in this section. For event-based OF, we introduce them from two aspects: software and hardware.

A. Software Event-Based OF

The earliest methods include [22]–[26] and are compared in [27]. They estimate the flow by adapting frame-based LK to event-based cameras, modeling cortical Direction Selective (DS) cells, or fitting a Local Plane (LP) [24], [28] to the Timestamp Image (TI) of the timestamps of the most recent event at each pixel. Several works have demonstrated interesting uses of averaged DVS OF for aircraft flight control [7], [8].

In recent years, motion compensation combined with optimization approaches have been widely used in event camera OF [2], [29]–[32] and have become the most popular approaches among all the non-learning-based event camera OF methods. These methods stem from the simple observation that if the event cloud is viewed in 3D spacetime, then there is a view angle where the events locally line up. This view angle represents a particular flow. The optimization procedure to locally align the event cloud can be carried out by a search procedure that maximizes the contrast of the resulting 2D image, and can be considered as a more robust iterative LP method. Since they requires multiple iterative optimizations and lots of memory to hold a cloud of events, they need an entire desktop computer for real time operation. However, optimized implementations such as Mitrokhin et al.’s BetterFlow [32] make it feasible to detect moving objects standing out from a moving background.

1https://sites.google.com/view/edflow21/homesites.google.com/view/edflow21/home

2The TI [22] was rediscovered and called the Surface of Active Events (SAE) or Time Surface [14].
Other event-based OF approaches were also developed. The relation between segmentation and OF is like chicken and egg: Better segmentation improves OF accuracy and more accurate OF could improve segmentation. In [31], they analyzed the influence of different contrast maximum reward functions on the aperture problem to jointly estimate the pixel-level segmentation and OF. [33] estimated the OF and intensity image from a single blurred DAVIS APS image and event stream, similar to [34]: Both use variational methods to jointly estimate OF and intensity. However, [33] exploits single-frame motion blur so they only require one single APS image. Nevertheless, both of them are very slow, e.g. less than 1 frame/second on desktop CPU. Akolkar [28] proposed a method based on [24] that measures LP flow over several aperture scales to maximize the mean value of the normal OF over all scales, thus mitigating the aperture problem.

Low [35] is another work based on LP fitting. It proposes using Prim’s algorithm to find the optimal event sets for plane fitting to improve the accuracy. They simplify the original LP fitting [24] by imposing more constraints on the incoming event and make the algorithm non-iterative. Guidelines for potential implementations on hardware are also reported in this work. However, it still suffers from limitations of local plane fitting for computing normal flow for edges and difficulties with dense texture, which produces a dense point cloud.

Almatrafi’s Davis-OF et al. [36] adapts the classical optical flow equation to a log optical flow equation. The spatial derivative is computed from the Dynamic and Active pixel Vision Sensor (DAVIS) Active Pixel Sensor (APS) frame. The event timestamps between two frames interpolate the temporal derivative. The final optical flow is thus calculated by solving the log optical flow estimation. This method cleverly combines the high spatial accuracy of APS and high temporal resolution of events, but it can only sample at the APS frame rate and fails when the limited-dynamic-range APS frames saturate.

Almatrafi et al. [37] proposes an interesting frame-based method that uses a ‘distance surface’ to represent the distance of a pixel to the closest event edge; this surface can nicely integrate multiple events (if the input is strongly denoised). Combining this distance metric and the classical optical flow equation, they derive a new equation which they call distance optical flow equation. They then use the classic Horn and Schunck method to calculate regularized optical flow on this surface. On their new IMU-based camera rotation DAVIS flow dataset called DVSMOTION20, accuracy of flow is slightly better than the authors’ fine-tuned CNN-based EV-FlowNet from [38]. Cost is dominated by the slow Horn-Schunk algorithm that requires in Matlab 700ms for each computed motion field; the distance transform requires 35ms per frame.

Deep learning is also explored in event-based OF [38]–[43]. These Deep Neural Network (DNN) learning based methods are more accurate than the hand-crafted approaches, but they are much more computationally intensive. For example, [44] reported that their most capable Convolutional Neural Network (CNN) runs at 40 Frames Per Second (FPS) on an NVIDIA 1080Ti Graphics Processing Unit (GPU) whose power consumption is about 200W. Many of these DNNs have high levels of sparsity that could be exploited in upcoming accelerators, but the papers usually do not provide the operations per input sample or the level of sparsity.

B. Hardware Event-Based OF

The above methods are implemented on CPU or GPU, consuming at least tens of watts and large areas of expensive silicon. Computing OF quickly, close to the sensor maintains the efficiency and low-latency advantages of event cameras. Hardware implementations of BMOF for standard video have been claimed to compute more accurate OF than LK methods when the hierarchical matching results are regularized [51], but most implement LK (e.g. [52]) or other nearest-neighbor estimation. See [51] for a useful summary of hardware block matching flow architectures for conventional video.

Neuromorphic vision chips over 1986-2000 implemented various models of DS cells from biology [53]–[57]. In 2018, [48] implemented a spiking neural network that modeled Barlow-Levick DS neurons using a DVS and the IBM TrueNorth system, but software computes OF from the output spikes.

Table I compares EDFLOW with other DVS OF implementations that directly output the flow vectors, including our first BMOF implementation [49]. The method [24] of fitting a Local Plane (LP) to the timestamp image (TI) was implemented in FPGA by Aung et al. [46]. This work cleverly uses lookup tables to avoid matrix inverse and divides the 5×5 TI into 9 subplanes for parallel 3×3 plane fitting with timestamp outlier and density checking. They report that they can process up to 2.75M events per second. However, the accuracy is also limited by the 3×3 planes and only a 5×5 area around each event, which is much smaller than the 43×43 total search area of EDFLOW. Huang et al. [47] used the Celex camera and added a part called Pixel Rendering Module (PRM) to the existing DVS pixel circuit. The PRM can quickly output the DVS event timestamp and logarithmic intensity for these 5 pixels. Thus they easily can compute local gradients and OF on the attached FPGA, however, they only provide a couple of figures showing simple flow and patent applications [58], [59]. The PRM requires 5 times the sensor output bandwidth, and estimating the gradient from only 4 neighbours is sensitive to noise. Most recently, Stumpf et al. [45] posted hardware Aperture Robust Multiscale flow (hARMS), an FPGA realization of [28]. This LP-based method selects the maximum flow vector from multiple scales to mitigate the aperture problem. hARMS uses a fixed-size window of past events that must be adjusted to the scene dynamics.

C. Event-Based Corner Detectors

When we estimate the OF locally on an edge, the flow along the edge is ambiguous. Keypoints (essentially corners) are used to help solve this problem, by restricting the flow computation to parts of the scene with less local flow ambiguity. Here we give a brief review on event-based corner detection. Clady [60] in 2015 reported the first DVS corner detector. It detected intersections of local planes fitted to
TABLE I
EDFLOW Compared With Prior DVS OF Hardware Implementations

| HW implementation | Algorithm | Hardware | Latency (us/event) | OF radius | Denoising | Keypoints |
|-------------------|-----------|----------|-------------------|-----------|-----------|-----------|
| Stumpp et al. [45] et al. | Multiaperture LP | DVS→Field Programmable Gate Array (FPGA) | 0.8–10 | NA | × | × |
| Aung et al. [46] | LP | DVS→FPGA | 0.36 | 2 | × | × |
| Huang et al. [47] | Gradient-based | DVS→FPGA | NA | 1 | × | × |
| Haessig et al. [48] | DS² | DVS→TrueNorth→CPU | NA | 1 | × | × |
| Our prior BMOF [49] | BMOF³ | USB DVS→FPGA | 0.22 | 4¹ | × | × |
| This work | EFAST→BMOF | DVS→FPGA | 1 (0.06-0.1)² | 21 | ✓ | (✓) |

¹ The radius of neighbors considered for each OF output.
² Aung [46] denotes with a local spatio-temporal correlation check [22] and imposes a refractory period before LP OF. Liu [49] denotes with keypoint detector and minimum event density check in OF.
³ Liu [49] used only binary event slices, 9x9 blocks, and Binary Hamming Distance (BHD) with 1-pixel search distance.
⁴ Uses 4 Barlow-Levick DS cells [50]; OF computed by CPU.
⁵ Only EDFLOW offers keypoints; keypoints are optional.

The TI. Vasco [61] reported eHarris in 2016. It adapted the frame-based Sobel-filter Harris detector to binary event patches. Muggler [62] in 2017 reported Event-Based time surface FAST (EFAST). It detects corners on the timestamp image with frame-based Features from Accelerated Segment Test (FAST). In 2019, Manderscheid [63] proposed a method called Speed Invariant Learned Corners (SILC). A random forest classifier detects corners. By using SILC, they decreased the reprojection error by 2X compared to using the normal TI. In 2021, Li [64] posted a Smallest Unvalue Segment Assimilating Nucleus (SUSAN)-based corner detector. It also provides a useful summary of software denoising methods.

However, the above methods were software developments and use transcendental functions and floating point serial CPU computation, making them difficult to realize in efficient FPGA logic circuits. Because of its simple form, our corner detection development started with EFAST [21]. We found that a simpler method provides better OF accuracy and report it here. Details of EFAST are explained in the SM.

III. INTRODUCTION TO ABMOF AND SFAST

Fig. 2 is a top-level diagram of EDFLOW. The DVS events first flow into the Slice-based FAST (SFAST) keypoint detector, and then flow is computed on these corner events by Adaptive Block Matching Optical Flow (ABMOF). EDFLOW uses event slices to accumulate DVS events. Event slices are equivalent to event frames in [14]. Each pixel in an event slice holds a count of accumulated events since the slice was last cleared. By analogy with photography, we define the exposure time of an event slice as the event accumulation time between the start timestamp and the end timestamp.

ABMOF [21] is a semidense method that computes the OF at points where DVS events signal that the brightness has changed. ABMOF computes OF using the past two slices using a multiscale Block Matching (BM) Coarse to Fine (CTF) search inherited from video compression [65].

ABMOF exploits the activity-driven DVS output to dynamically control its sample rate. The area event count method [21, Sec. 2.2.1] acts as a feed-forward slice exposure time controller and the average block matching distance $d_{avg}$ acts as a feedback controller on the slice memory circular buffer rotation event count number $k$ [21, Sec. 2.2.2] (Sec. IV-C).

Together, they dynamically control the slice exposure time and the interslice time interval $\delta_t$ to optimize the slice feature quality and BM search range. These capabilities makes ABMOF robust to dynamic scenes with varying motion speeds and scene structure; see Sec. V-B for experimental results. For more details of the algorithm, we refer readers to [21].

For detecting keypoint corners, we use a novel method called SFAST, which is adapted from FAST. FAST is a popular corner detection method in frame-based computer vision because of its robustness and speed [66]. To use FAST on event-based data, we accumulate an event count image in the event slice and treat it as an intensity image. Because SFAST and ABMOF are block-based methods, we can use the same event slice generated by ABMOF to detect corners. However, to simplify the design and increase the bandwidth, we use a separate copy of the event slice as a dedicated memory for SFAST. Since SFAST only operates on the coarse scale, this extra copy needs only a small amount of memory. The steps following the conclusion of the exposure

![Fig. 2. Top level diagram of EDFLOW SFAST corner detection and ABMOF implementation, including sizes of slice memories.](image-url)
of an event count image are similar to FAST, but we improved the robustness for the relatively low precision of event count images, whose 4-bit counts have limited dynamic range are quite quantized compared to conventional 8-bit gray scale images.

Fig. 3 illustrates SFAST. It shows the slice array of accumulated event counts neighboring the current event. Since DVS event ON and OFF polarity is ignored, the event count is a 4-bit unsigned saturating value. The red pixel in the center is the current event. To detect a corner, SFAST checks both the inner circle and outer circle centered on the current event. The cyan streaks on the inner circle and the outer circle mean the minimum event count of these streak pixels is sufficiently larger than the maximum of the rest of the count values in the same circle. The gap between the minimum of cyan streaks and the maximum of the rest of the values is called KPthr. Depending on the SFAST logical condition of a streak on one, either, or both of the circles, the current event is called a corner event. The logical condition sets the discrimination of SFAST and is a user choice; for the results presented in Sec. V-C, we used only a streak on outer circle. The gap between the minimum of cyan streaks and the maximum of the rest of the values is called KPthr. Here the length of the streak or streak_length can range from 2 to 7 for inner circle and 3 to 11 for outer circle.

SFAST and ABMOF parameters that are used in this paper are summarized in Table II.

IV. HARDWARE IMPLEMENTATION

Fig. 2 shows the top level EDFLOW diagram. It has 4 major blocks: DVS, SFAST, ABMOF, and multiscale RAMs. SFAST accepts the normal DVS event stream as input, writes the SFAST and ABMOF slices and output events that have been detected as corners. ABMOF estimates OF on this corner stream and outputs the final OF stream. Sec. IV-A describes the event slice accumulation and block search strategy, Sec IV-B describes implementation of SFAST, and Sec. IV-C describes the implementation of ABMOF.

Crucially, to enable processing high event rates observed in moving camera applications, parallelism in the SFAST keypoint logic design reduces it to require only 5–6 clock cycles. If a keypoint is detected, then massive parallelism in the Sum of Absolute Differences (SAD) computations hold the BMOF computation to only 100 cycles. Therefore, the combined design can measure keypoints up to event rates above 16 MHz, and can compute flow at 1 MHz.

A. Multiscale Slice Event Accumulation

Events are accumulated in the current slice $t$, while slices $t - \delta_1$ and $t - \delta_2$ are used for corners and OF. For simpler logic design and concurrent access, SFAST duplicates the coarse slice RAM. The cost is low since it is the coarse scale. A spare slice of each scale is cleared by a dedicated state machine while the $t$ slice accumulates events. ABMOF requires all 3 scales, but SFAST only uses one coarse scale. There are thus a total of 16 slice memory blocks as shown in Fig. 2.

The slice memory controller increments the slice pixel values in the current slice at all 3 scales. For scales 1 and 2 (medium and coarse), the $x$ and $y$ addresses are right shifted by 1 or 2 bits for subsampling. Thus, the medium and coarse scale pixels sum events from $2 \times 2$ and $4 \times 4$ DVS pixels. DVS ON and OFF event polarity is rectified (i.e. ignored) in this design based on our earlier study [21]. When a new event arrives, the event count of the corresponding (shifted) addresses on all scales increases by 1. To make the blocks have about the same size in the DVS pixel address space among 3 scales, different block dimensions $b_c$, $b_m$, $b_f$ are used, listed in Table II. SFAST only checks for corners on the coarse scale, because this scale accumulates the most events and provides the most robust, large-scale corners.

B. SFAST Hardware Design

Fig. 4 shows the workflow of the hardware SFAST. The input of the system is the event stream. In the case that both circles are checked, it first checks the inner circle. If this event
Fig. 4. The workflow of the hardware SFAST corner detector illustrating the BOTH circles check. See Sec. IV-B for explanation.

The SFAST corner check consists of a slice memory controller, a sorting unit, and a contiguous index detector. The first part accumulates incoming events to the slices and reads the SFAST circle data from the slice. The second and third part are designed as two Processing Elements (PE): StreakRanker and StreakDetector. The principle and circuit of these two PEs are explained in this section. Since the only difference between inner circle and outer circle checking is the circle radius, we designed a general circuit to save resources.

First, StreakRanker sorts the accumulated counts to a list of event count ranks; i.e., each entry of the list is the rank of event counts, with the largest rank corresponding to the pixel with the largest event count. Then a second circuit StreakDetector detects streaks of the pixels with the largest event counts by detecting the longest contiguous set of ranks starting from the largest rank. This contiguous set represents a potential streak: If all event counts for this set are sufficiently larger by KPthr compared with the other pixels, then it is regarded as a streak.

Fig. 5 shows an example to illustrate the detection of the streak on the inner circle of Fig. 3. The data list on the top is the unrolled circle data and is read from the coarse event slice $t - \delta_1$. The inner circle has 8 values and every value has 4 bits, so it is efficiently packed as a 32-bit value. Fig. 5 shows that the second data of the result is 7, which means that the corresponding entry event count 12 is the maximum value in the input list. The schematic of StreakRanker is shown in our SM Fig. S5.

Second, StreakDetector finds the maximum length valid streak. All streaks to be checked for the inner circle are shown inside the dashed rectangle in Fig. 5. The check is best explained by an example: Suppose the check is for 3 contiguous largest counts on the 8-pixel circle. The ranks of these 8 pixel event counts have been computed by StreakRanker. The check ensures that all ranks of these 3 pixels are larger than 5. Similarly, a check for 4-pixel long streaks only needs to ensure all ranks are larger than 4. These checks require only comparators and are done in parallel. Among these valid streaks, StreakDetector returns the maximum streak length. If no valid streaks are found, StreakDetector returns 0. The schematic of StreakDetector is shown in our SM Fig. S6.

The circuits of StreakRanker and StreakDetector shown in Fig. 5 check only one data per clock cycle. Therefore, 8 cycles are required for inner circle and 12 cycles are required for outer circle checking. If both stages are processed, 20 cycles are required. To decrease the latency, we used 4 copies to parallelize the check, to process 4 data per cycle and so the check requires at most 5 clock cycles. Most events are not corners, and thus require only 6 clock cycles.

C. ABMOF Hardware Design

Fig. 6 is the architecture of the ABMOF hardware block. It consists of 7 Processing Element (PE). Each of the 7 PEs is a small computation unit with a specific function. The memories store the $t$, $t - \delta_1$, and $t - \delta_2$ event slices, each at 3 spatial scales. The First In First Out memory (FIFO) buffer data to synchronize between PEs with different data rates. All PEs use the same 100 MHz global clock.
PE 1 receives DVS events over the Address Event Protocol (AER) interface, extracts the x and y addresses and timestamps, and also filters out events that are too close to the boundary for OF computation.

PE 2 rotates the slice memory circular buffers using the area event count method, using feedback from PE 7 that controls the area event number $k$ [21].

PE 3 is the top level block matching Coarse to Fine (CTF) search controller, and also the memory controller. It updates the current event slices at each scale for every incoming event. It outputs the data in two FIFOs column by column in a specific order illustrated in Fig. 7.

PEs 4-6 matches blocks by computing the Sum of Absolute Differences (SAD) values to find the minimum SAD, as illustrated in Fig. 7. The SAD calculation is divided into PEs ColSad, RowSummer and FindStreamMin.

Fig. 6 shows how the CTF search starts at the corner event location in slice $t$. At each scale the Reference Block (RB) from slice $t - \delta_1$ centered on the previous scale best match location is compared with Target Block (TB) in slice $t - \delta_2$ over the Search Area (SA). The block size varies over the scale to keep nearly the same 25x25 block size in the DVS pixel array. Each search is an exhaustive full search over the entire 7x7 SA. (Many implementations use a sparser search strategy [67] but we found, like [51], that a sparser search strategy such as Diamond Search [68] as we use in software has too many conditional branches, which makes it difficult and inefficient to implement in hardware; see SM Sec. G.) Each coarser best match result is used as an initial value of the next finer scale search. The OF is a $(\delta_x, \delta_y)$ displacement vector which is the scaled sum of the displacements at each scale. The length of $(\delta_x, \delta_y)$ is the match distance $d$. The maximum search distance is $\pm 21$ pixels in x and y directions.

By careful partitioning of the slice block memories (SM Sec. C-B), the entire BMOF computation requires only about 100 clock cycles, which is 1 us at our 100MHz clock frequency (Sec. D in our SM).

Finally, PE 7 encapsulate the event addresses, timestamps, polarity and the OF result in one data and sends it out. Before sending the final output, it checks for sufficient BM feature density. If the final minimum SAD value is larger than the threshold $SAD_{\text{max}}$, or the non-zero area of RB or TB is too sparse ($B_{\text{sparsity,max}}$), or there is insufficient RB and TB non-zero overlap ($SAD_{\text{overlap,min}}$), then the OF is flagged invalid (SM Sec. E). PE 7 also computes the feedback control of $k$ (Sec. V-B). It calculates the actual average block match distance $d_{\text{avg}}$. If $d_{\text{avg}} > d_{\text{targ}}$, $k$ is reduced by factor $1/k_{\text{step}}$, otherwise it is increased by factor $k_{\text{step}}$.

V. EXPERIMENTS

To evaluate performance, we designed and built the DAVIS346Zynq EDFLOW platform in Fig. 8. We also implemented EFAST to compare its performance with SFAST. Details of EFAST and its hardware implementation are in our SM. This section provides accuracy (Sec. V-C) and throughput/power (Sec. V-D) comparisons, along with experiments demonstrating automatic slice exposure control (Sec. V-B). Videos showing EDFLOW output are included in the SM. Our experiments used the Table II parameter default values, only we used the diamond search block matching in software accuracy measurements of EDFLOW because the full search done by our hardware is prohibitively slow in software and the accuracy difference is not significant.

A. DAVIS346Zynq EDFLOW Camera

Existing event cameras have insufficient logic resources, so we designed a new powerful camera platform we call DAVIS346Zynq to implement EDFLOW (Fig 8). The 6-layer board holds a DAVIS346 sensor chip [69] and the most powerful System on Chip (SoC) FPGA of the Xilinx Zynq7000 family (XC7Z100, costing about $500). The XC7Z100 has a Kintex-7 FPGA on the Programmable Logic (PL) and a dual-core 800MHz ARM Cortex-A9 on the micro-processor Processing System (PS). Using this powerful SoC, we started from the logic developed in our SeeBetter and Visualise projects [70]–[72]. This logic handles AER handshaking, timestamp generation, and USB interfacing for the original DAVIS346 camera. Most hardware blocks besides ABMOF and SFAST on the PL are implemented with High Level Synthesis ( HLS) 3 (See Sec. B in our SM). The PS runs a bare-metal program that lets us program registers over the serial port (SM Sec. I). The firmware can send DVS events to the EDFLOW logic from the Secure Digital (SD) card

3https://github.com/SensorsINI/EDFLOWgithub.com/SensorsINI/EDFLOW source code
Fig. 7. The minimum block SAD computation of the HW ABMOF. It consists of PEs 3–6. Two gray rectangles here represent the DVS event slices \( t - \delta_1 \) and \( t - \delta_2 \). The blue Search Area (SA) and red Reference Block (RB) are centered on the current event. Here we choose \( b = 3 \) and \( r = 1 \) for illustration (the actual values are \( b = 7 \) and \( r = 3 \)). The first FIFO data is the right column and the last data is the left column. PE 5 RowSummer sums rows of the PE 4 ColSad output to result in final block SAD result. In this example, every 3 columns in the same row generate a block SAD. PE 5 FindStreamMin reads the block SAD value column by column and return the minimum SAD in each column.

Fig. 8. The prototype DAVIS346Zynq EDFLOW event camera platform. It consists of three parts: DAVIS as the input (only DVS output is currently used), FPGA as the EDFLOW processing, and USB as the output for testing. DVS events along with keypoints and OF are transmitted to the host computer over Universal Serial Bus (USB). We verified they are identical to output from the software ABMOF algorithm.

B. Adaptive Slice Exposure Control

As discussed in Secs. III and IV-C, ABMOF controls the slice exposure using area event count number \( k \), which is itself under feedback control to center the average BMOF matching distance \( d_{avg} \) in the range of possible match distances.

Using adaptive \( k \) is useful when speed varies over time. Fig. 9(a) shows an experiment with a rotating dot covering a speed range of more than a factor of 100, from \( \approx 50 \) px/s to 7000 px/s. Fig. 9(b) shows ABMOF quantities. The slice duration \( \delta_1 \) started at about 30 ms (30 FPS) and ended at less than 1 ms (1kFPS). As a result of feedback, \( k \) started from 1k events and ended at 0.3k events as the dot sped up. The feedback stabilized the average matching distance \( d_{avg} \) around \( d_{targ} \approx 7 \) px.

Fig. 9(c) plots \( k \) and \( \delta_1 \) versus the dot speed. The feedforward area event count makes \( \delta_1 \) vary inversely with dot speed over most of the speed range. The \( k \) feedback makes \( k \) decrease approximately as the square root of dot speed, i.e., it makes \( \delta_1 \) decrease with speed more than it would with fixed \( k \).

Fig. 9(d) shows that using adaptive \( k \) improves the accuracy of flow. We measured the Ground Truth (GT) speed of the dot using a tracker (green curve). Using a fixed \( k = 600 \) works well for low speed (red curve), but when the dot moves quickly, the flow results are much less accurate than when \( k \) adapts \( d_{avg} \) towards \( d_{targ} \) (blue curve).

In summary, when the input scene speed varies over time, the adaptive \( k \) stabilizes the BMOF vectors in the middle of their dynamic range.

C. Flow Accuracy Comparisons

We evaluated the EDFLOW accuracy by comparing ABMOF and ABMOF combined with corner detection methods EFAST (ABMOF+EFAST) and SFAST (ABMOF+SFAST). For comparison with accuracy that can be obtained from a costly CNN, we include our own EV-FlowNet [38] accuracy measurements. We also adapted a software version of the Local Plane (LP) method from [27] to compare with accuracy from the DVS flow camera Aung et al. [46]. The Huang et al. [47] flow camera relies on intensity samples, and Haessig et al. [48] is computed mostly in software. The multiaperture LP code used in hARMS [45] is not published in [45] or [28]. We used exactly the same algorithm parameters for all sequences.

To quantify the accuracy, we use Average Endpoint Error (AEE), Average Relative Endpoint Error (AREE), Average Angular Error (AAE), and event density as the metrics. AEE and AAE are common metrics for OF evaluation [74]. Since event cameras do not have fixed frame/sample rates, we measure AEE in px/s rather than px/frame as is conventional for frame datasets. The test sequences have flow magnitude ranging from 30–500px/s. We report AREE, the average of

---

4https://github.com/SensorsINI/jaer/blob/master/src/ch/unizh/ini/jaer/projects/minliu/PatchMatchFlow.java ABMOF PatchMatchFlow.java source code in jaerproject.org
the Relative Endpoint Error (REE), which is the magnitude of the flow vector divided by the ground truth flow vector magnitude, because it is more interpretable than the AEE. However, we note that a REE can be much larger than 100% if the GT flow is slow and the measured flow is fast, but not vice-versa. We included AAE to show how well direction of flow is measured, and (implicitly) how well flow ambiguities along edges are resolved by using keypoints. We define the event density as the fraction of events that are processed for OF.

An outlier metric was introduced in the KITTI benchmark 2015 [75], which defined it as a flow vector result with an endpoint error larger than 3 pixels or 5% of the GT magnitude. It was adapted for the EV-FlowNet in [38], but contrary to the statement in the paper, the EV-FlowNet code actually only checks for 3 pixel errors over each effective frame. For the outdoor day sequences, the frame rate is 45 Hz, which is equivalent to an error greater than 135 px/s. Unfortunately, this metric does not really make sense, because this huge error is larger than most of the sequences flow magnitude. Additionally, this metric is misleading when the algorithm produces a systematic error, such as the underestimation of true flow by the normal-flow LP method. To better understand the distribution of errors, we show a histogram visualization of flow in Fig. 11.

Fig. 10 shows the sequence slider_hdr_far from [76]. We used this sequence because it is a natural forest scene with a distribution of angle and contrast. The figure compares EDFLOW variants with LP used in [46] and EV-FlowNet [38]. The flow is rendered using barbs to better show direction and magnitude. All EDFLOW variants measure the true OF even for most of the slanted features, in contrast to the normal flow measured by LP. EV-FlowNet also reports true flow direction.

Table III shows that EDFLOW achieves accurate measurement of the slider_hdr_far flow magnitude and direction, with accuracy 2X better than EV-FlowNet and 10X better than the LP method.

The slider_hdr_far sequence has uniform flow, so the distributions should show a single sharp peak. Fig. 11 shows that EDFLOW correctly has zero variation for the vertical velocity component and has a tight but bimodal horizontal velocity distribution, probably from search distance quantization in BM. EV-FlowNet shows variations on both x and y directions but the flow is still concentrated. By contrast, the normal flow of LP results much broader distributions. All methods underestimate the flow speed, at least slightly.

ABMOF+SFAST produces the most accurate EDFLOW flow, probably because the keypoints result in more reliable matching, but SFAST produces more concentrated keypoints than EFAST with more measurements of flow of each of the keypoints, which could increase VOD accuracy. The EFAST corners in Figs. 10 have many more “isolated corners” than the SFAST corners. Many of these are the failure cases we show in Fig. S3 of our SM.

Table III further compares accuracy on more complex indoor drone flying and outdoor daytime and nighttime driving scenes from MVSEC [77]. MVSEC has two main scenes: indoor flying and outdoor driving. It provides OT GT indirectly based on the measured depth and camera pose. For indoor flying sequences, the whole background is static and only the drone is moving, so the OF GT is accurate. However, for outdoor driving sequences, since OF GT is obtained from GPS camera ego motion, IMU camera rotation, and LIDAR depth sensor, the converted OF GT is sometimes incorrect, particularly when the depth sensor incorrectly reports infinite depth (e.g. in trees and lampposts), for moving cars and pedestrians, and for hood reflections. Thus the outdoor driving sequences have much larger GT error. For benchmarking,
we followed [38] by cropping out the lower 70 pixel car
hood region and selecting parts of sequences that do not
have moving objects, but the accuracy values should only
be compared between methods; the absolute values are not
reliable indications of overall flow accuracy.

Fig. 12 shows the qualitative result on outdoor_night2. Both
ABMOF variants reproduce the expanding GT flow field,
as does EV-FlowNet. Like [38], we also could not make the
LP method work well with this data, probably because the
event timings are quite noisy. Even under the bright lighting
conditions in indoor_flying and outdoor_day, the LP method
has poor AAE because it computes normal flow and it works
poorly for dense texture. Careful inspection shows that the GT
flow angles closely match EDFLOW and EV-FlowNet flow,
but the flow speed GT overall is not so accurate; probably
mostly because of missing depth values for some objects like
lampposts and distant buildings.

Overall, Table III shows that the best methods still only
achieve about 40% AREE and 10° AAE on complex scenes.
EDFLOW is 3X to 10X more accurate for speed and angle
of flow than its closest hardware-matched LP implementa-
tion [46]. For the flying and driving sequences, it achieves
accuracy very similar to EV-FlowNet for both AREE and
AAE. The good AAE accuracy of EDFLOW, in particular
ABMOF+SFAST, shows that its keypoint detection helps
produce accurate flow results, although with the limitation
of much lower flow result density.

D. Throughput, Power, and Resources

Table IV lists the processing time (computational latency)
of ABMOF and its variants compared with [46]. The dense

Table III

| Sequence        | AREE (%) | AEE (px/s) | AAE (°) | % event density |
|-----------------|----------|------------|---------|-----------------|
| slider_hdr_far  |          |            |         |                 |
| LP Aung et al.  | 76.54    | 71.56      | 49.09   | 8.49            |
| EV-FlowNet      | 20.52    | 17.61      | 6.38    | 100             |
| ABMOF           | 9.91     | 9.2        | 1.45    | 56.46           |
| ABMOF+SFAST     | 8.57     | 8.0        | 0.99    | 4.57            |
| ABMOF+SFAST     | 7.99     | 7.2        | 0.44    | 5.81            |
| indoor_flying   |          |            |         |                 |
| LP Aung et al.  | 95.64    | 72.56      | 69.35   | 89.15           |
| EV-FlowNet      | 63.61    | 35.18      | 32.55   | 100             |
| ABMOF           | 52.69    | 31.33      | 16.57   | 67.88           |
| ABMOF+SFAST     | 50.79    | 28.38      | 16.10   | 1.07            |
| ABMOF+SFAST     | 37.52    | 23.02      | 12.15   | 3.42            |
| outdoor_day1    |          |            |         |                 |
| LP Aung et al.  | 122.10   | 57.38      | 69.41   | 88.56           |
| EV-FlowNet      | 65.54    | 21.62      | 22.88   | 100             |
| ABMOF           | 90.13    | 37.18      | 33.76   | 66.75           |
| ABMOF+SFAST     | 85.78    | 37.14      | 30.97   | 2.82            |
| ABMOF+SFAST     | 69.40    | 27.55      | 23.30   | 6.93            |
| outdoor_night2  |          |            |         |                 |
| LP Aung et al.  | 134.69   | 69.90      | 70.44   | 44.32           |
| EV-FlowNet      | 44.68    | 22.97      | 14.20   | 100             |
| ABMOF           | 50.18    | 30.71      | 15.47   | 63.06           |
| ABMOF+SFAST     | 41.36    | 32.93      | 15.20   | 4.38            |
| ABMOF+SFAST     | 36.06    | 24.20      | 9.91    | 6.13            |

1 EDFLOW measures px/s, as does LP [46]. To convert our AEE in px/s to pixel translation values reported in other papers, divide by the
frame rate (which is not always reported). For all MVSEC sequences, it is critical to account for possible different starting times of the GT flow
and the recording.
2 To model [46] accuracy, we measured error in jAIR using the LocalPlanes method [27]; we used the RobustLP method with plane size of 5x5,
refractory period of 0 ms, maxDet=50 ms, thr1=3e-3, thr2=3e-3, thr3=3e-3, as closely as possible as reported in [46]. The LP method usually
underestimates the true flow because of its normal flow measurement.
3] EV-FlowNet only outputs 256x256 pixels as shown in Fig. 12. We used outdoor_night2 because outdoor_night1 has many other moving
cars and substantial camera shake. We computed px/s from EV-FlowNet pixel displacement output using the EV-FlowNet effective frame rate.
As in [38], we evaluated EV-FlowNet using event volumes with the DAVIS APS frame rate of each sequence (For indoor_flying, 60 Hz; for outdoor_day, 45 Hz; for outdoor_night, 10 Hz). We used the reported EV-FlowNet 2e9 with 2 Residual blocks variant.

5 The processing time per event for the EV-FlowNet [38] CNN is not a valid
metric because it uses a constant FPS event volume input representation. Its
processing cost is constant; the wall clock processing time per frame is 40 ms
using the smaller EV-FlowNet network variant on an NVIDIA 1050 GPU
with a power consumption of 75W only for the GPU. For a CNN that uses
“constant count” event voxel grid input [14, Sec. 3.1, Fig. 3], then we could
assume e.g. 10k events per input and compute that the time per event would
be 40ms/10kev=4us/ev.
Fig. 10. Snapshots of 20 ms of OF from EDFLOW variants compared with the LP method used in [46] and EV-FlowNet [38] on slider_hdr_far sequence from [76]. The APS frame is overexposed for the DAVIS camera but still makes high quality events. Color wheel shows OF direction. Vector length indicates speed. Red squares in EFAST and SFAST show detected keypoints.

Fig. 11. The flow distributions of ABMOF + SFAST, LP, EV-FlowNet, and ground truth (GT) on slider_hdr_far.

ABMOF can process OF using only 1 us/event, or at rate of 1 MHz. It is about the same speed as [46]. The sparser ABMOF+EFAST and ABMOF+SFAST are more than 10X faster and can process event rates up to 16.6 MHz.

This speed is made possible by hardware parallelism in the keypoint detection and SAD computations. Keypoint detection requires on average only about 6 clock cycles. Accounting for the operations only in ABMOF, EDFLOW achieves 123 GOp/s at a clock frequency of only 100 MHz, \( i.e. \) the equivalent of 1230 Op/clock cycle.\(^6\)

Xilinx Vivado reports that the EDFLOW logic modules consume about 1 W (Fig. 13). EDFLOW is thus about 75X more energy efficient for ABMOF and more than 450X more energy efficient for ABMOF+SFAST than the CNN-based EV-FlowNet running on a 75 W laptop GPU.

Table V summarizes EDFLOW FPGA resources. EDFLOW uses about the same logic as [51], and more logic and memory than [46], but only a fraction of the whole FPGA. We see that the largest usage is of Block RAMs (BRAM) (25\%) and Digital Signal Processing units (DSP) (33\%). The adders in the DSPs are used for the SAD computations.

### VI. Discussion and Conclusion

Calculating OF only on “corner events” improves OF accuracy, with the drawback of lower OF density, but with advantage of processing at least 10X higher event rates. Our results show that SFAST achieves more accurate OF than EFAST. The use of keypoints is optional, and EDFLOW includes the semidense ABMOF mode where all events that can be processed without overflowing the pipeline are processed, and with OF vectors labeled as keypoint OF events.

Compared with previous DVS OF methods, ABMOF uses a multiscale coarse-to-fine BMOF method that matches complex patterns to achieve three to ten times better accuracy than the best previous hardware DVS OF implementation [46]. Comparing the ABMOF variants with CNN-based OF, ABMOF is hundreds of times more power efficient and at least ten times quicker, and our Table III shows it can be as accurate as the CNN, but produces sparser output.

\(^6\)One SAD computation requires 3 math operations: subtract, absolute and accumulate. The total number of SADs to be computed is \((25 \times 25 + 13 \times 13 + 7 \times 7) \times 49 = 41307\) SAD. Thus, the total operations are \(41307 \times 3 = 123921\) Op. All of them are finished within 100 cycles.
Fig. 12. EDFLOW ABMOF and ABMOF+SFAST compared with LP [24], [27], [46] and EV-FlowNet [38] on outdoor_night2 from [77] at \( t = 197.6 \) s.

Fig. 13. The composition of dynamic power consumption in the Zynq XC7Z100 SoC. Total power is 2.96W and the EDFLOW logic modules burn 1.03W. Wall plug power measured from power strip is 10.1W and includes the power pucks, the FPGA power board, USB, DAVIS346, Video Graphics Adaptor (VGA) driver, and other components. Total power varies about 400mW depending on activity.

Readers can compare videos of flow estimation on the MVSEC sequences in our supplementary material and EDFLOW21 website.

Although EDFLOW does not make use of the precise DVS event timing in the way that LP or volumetric methods do, [12, Fig. 10] shows that DVS event timing jitter can exceed 1 ms, which makes methods like LP work poorly. However, EDFLOW retains the high dynamic range and activity-driven computation initiated by the DVS brightness change detection. All the OF computations done by EDFLOW are initiated by events, and EDFLOW’s event slice circular memory buffer rotation—and hence its sample rate—is determined by the event rate.

EDFLOW uses only a handful of multipliers and mostly 4-bit arithmetic. The FPGA implementation of ABMOF+SFAST uses 855 KB of BRAM and 669 48-bit DSPs to (inefficiently) implement the SAD accumulators. The actual total memory usage is less than 300 KB. EDFLOW would be straightforward to implement as an Application Specific Integrated Circuit (ASIC) block, where the logic and memory blocks could be more tightly integrated and the memory could be optimized. The main difficulty is using HLS for the ASIC since HLS is not popular for ASIC design (SM, Sec. J).

The inclusion of EDFLOW in an event camera offloads low-level keypoint detection and optic flow to camera logic circuits and could enable a continuous VOD pipeline within the camera.

**ACKNOWLEDGMENT**

The authors would like to thank inivation for their help with logic for interfacing to the DVS, and Garrick Orchard, Alex Zhu, Shih-Chii Liu, Yuhuang Hu, and their reviewers for helpful comments. They would also like to thank Cornelia Fermuller for suggesting and Levi Burner for building the `evimo_flow.py` script to extract GT OF from the EV-IMO2 dataset [79], although these comparisons were done during review and will only appear on the EDFLOW21 website.

**REFERENCES**

[1] T. Stoffregen and L. Kleeman, “Simultaneous optical flow and segmentation (SOFAS) using dynamic vision sensor,” 2018, arXiv:1805.12326.

[2] G. Gallego and D. Scaramuzza, “Accurate angular velocity estimation with an event camera,” IEEE Robot. Autom. Lett., vol. 2, no. 2, pp. 632–639, Apr. 2017, doi: 10.1109/LRA.2016.2647639.

[3] A. Z. Zhu, N. Atanasov, and K. Daniilidis, “Event-based feature tracking with probabilistic data association,” in Proc. IEEE Int. Conf. Robot. Automat. (ICRA), May 2017, pp. 4465–4470, doi: 10.1109/ICRA.2017.7989517.

[4] A. Hadviger, I. Markovic, and I. Petrovic, “Stereo dense depth tracking based on optical flow using frames and events,” in Advanced Robotics. 2020, pp. 1–12, doi: 10.1080/01691864.2020.1821770.

[5] K. McGuire, G. de Croon, C. D. Wagter, K. Tuyls, and H. Kappen, “Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone,” IEEE Robot. Autom. Lett., vol. 2, no. 2, pp. 1070–1076, Apr. 2017, doi: 10.1109/LRA.2017.2658940.

[6] A. Beyeler, J.-C. Zufferey, and D. Floreano, “Vision-based control of near-obstacle flight,” Auton. Robots, vol. 27, no. 3, p. 201, Aug. 2009, doi: 10.1007/s10514-009-9139-6.

[7] G. C. H. E. de Croon, H. W. Ho, C. De Wagter, E. van Kampen, B. Remes, and Q. P. Chu, “Optic-flow based slope estimation for autonomous landing,” Int. J. Micro Air Vehicles, vol. 5, no. 4, pp. 287–297, Dec. 2013, doi: 10.1260/1756-8293.5.4.287.
[48] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman, and G. Orchard, “Spatio optical flow for event-based sensors using IBM’s TrueNorth neurosynaptic system,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 4, pp. 860–870, Aug. 2018, doi: 10.1109/TBCAS.2018.2834558.

[49] M. Liu and T. Delbrück, “Block-matching optical flow for dynamic vision sensors: Algorithm and FPGA implementation,” in *Proc. IEEE Trans. Circuits Syst. Video Technol.* (ISCAS), May 2017, pp. 1–4, doi: 10.1109/ISCAS.2017.8050795.

[50] H. B. Barlow and W. R. Levick, “The mechanism of directionally selective units in rabbit’s retina,” *J. Physiol.*, vol. 178, no. 3, pp. 477–504, Jun. 1965, doi: 10.1113/jphysiol.1965.sp007638.

[51] K. Seyid, A. Richaud, R. Capoccia, and Y. Leblebici, “FPGA-based hardware implementation of real-time optical flow calculation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 1, pp. 206–216, Jan. 2018, doi: 10.1109/TCSVT.2016.2598703.

[52] J. E. Tanner, “Integrated optical motion detection,” Ph.D. dissertation, Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, 1986. [Online]. Available: http://resolver.caltech.edu/CaltechETD:etd-03102008-081506

[53] R. G. Benson and T. Delbrück, “Direction selective silicon retina that uses null inhibition,” in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, J. Moody, S. Hanson, and R. P. Lippmann, Eds., Burlington, MA, USA: Morgan-Kaufmann, 1992. [Online]. Available: https://papers.nips.cc/paper/1991/hash/4a20a3e6a4a915e12e-Abstract.html

[54] T. Delbrück, “Silicon retina with correlation-based, velocity-tuned pixels,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 4, no. 3, pp. 529–541, May 1993, doi: 10.1109/72.217194.

[55] R. R. Harrison and C. Koch, “A robust analog VLSI motion sensor based on the visual system of the fly,” *Auton. Robots*, vol. 7, no. 3, pp. 211–224, Nov. 1999, doi: 10.1023/A:1008961022887.

[56] S.-C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbrück, “Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time FPGA applications,” *IEEE Access*, vol. 7, pp. 134926–134942, 2019, doi: 10.1109/ACCESS.2019.2941282.

[57] S.-C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbrück, “Event-driven sensing for efficient perception: Vision and audition algorithms,” *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 29–37, Nov. 2019, doi: 10.1109/MSP.2019.2928127.

[58] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and S. Szeliski, “A database and evaluation methodology for optical flow,” *Int. J. Comput. Vis.*, vol. 92, no. 1, pp. 1–31, Mar. 2011, doi: 10.1007/s11263-010-0390-2.

[59] Menze and A. Geiger, “Object scene flow for autonomous vehicles,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* (CVPR), Jun. 2015, pp. 3061–3070, doi: 10.1109/CVPR.2015.7298925.

[60] E. Mueggler, H. Rebecq, G. Gallego, T. Delbrück, and D. Scaramuzza, “The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM,” *Int. J. Robot. Res.*, vol. 36, no. 2, pp. 142–149, Feb. 2017, doi: 10.1177/027836491661115.

[61] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfommer, V. Kumar, and K. Daniilidis, “The multivehicle stereo event camera dataset: An event camera dataset for 3D perception,” IEEE Robot. Autom. Lett., vol. 3, no. 3, pp. 2032–2039, Jul. 2018, doi: 10.1109/LRAL.2018.2800793.

[62] A. Zhu, (Apr. 2019). MUSEC Multivehicle Stereo Event Camera Dataset. Accessed: Jan. 6, 2022. [Online]. Available: https://daniillidis-group.github.io/mvsec/.

[63] A. Mitrokhin, C. Ye, C. Fermuller, Y. Alimoynos, and T. Delbrück, “EV-MO: Motion segmentation dataset and learning pipeline for event cameras,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 6105–6112, doi: 10.1109/IROS40897.2019.8968520.

---

**Min Liu** received the B.Sc. and M.Sc. degrees in electronic engineering from Beihang University, China, in 2011 and 2014, respectively, and the Ph.D. degree from ETH, in 2021. He is currently a Post-Doctoral Researcher at the Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland. He is working on event-based vision algorithms for embedded platform which requires low power consumption and has limited resource. His research interests include hardware algorithms and efficient computation.

**Tobi Delbrück** (Fellow, IEEE) received the degree in physics from the University of California in 1986 and the Ph.D. degree from Caltech in 1993. He is currently a Professor of physics and electrical engineering from the Institute of Neuroinformatics, University of Zurich and ETH Zurich, where he has been since 1998. The Sensors group, which he co-directs with SC Liu, currently focuses on neuromorphic sensory processing, control, and efficient hardware AI.