ABMOF: A Novel Optical Flow Algorithm for Dynamic Vision Sensors

Min Liu, *Member, IEEE,* and Tobi Delbruck, *Fellow, IEEE*

Abstract—Dynamic Vision Sensors (**DVS**), which output asynchronous log intensity change events, have potential applications in high-speed robotics, autonomous cars and drones. The precise event timing, sparse output, and wide dynamic range of the events are well suited for optical flow, but conventional optical flow (**OF**) algorithms are not well matched to the event stream data. This paper proposes an event-driven OF algorithm called adaptive block-matching optical flow (**ABMOF**). ABMOF uses time slices of accumulated DVS events. The time slices are adaptively rotated based on the input events and OF results. Compared with other methods such as gradient-based OF, ABMOF can efficiently be implemented in compact logic circuits. Results show that ABMOF achieves comparable accuracy to conventional standards such as Lucas-Kanade (**LK**). The main contributions of our paper are new adaptive time-slice rotation methods that ensure the generated slices have sufficient features for matching,including a feedback mechanism that controls the generated slices to have average slice displacement within the block search range. An LK method using our adapted slices is also implemented. The ABMOF accuracy is compared with this LK method on natural scene data including sparse and dense texture, high dynamic range, and fast motion exceeding 30,000 pixels per second. The paper dataset and source code are available from http://sensors.ini.uzh.ch/databases.html.

Index Terms—dynamic vision sensor, optical flow, event-based sensor, block matching, neuromorphic.

SUPPLEMENTARY MATERIAL

- Video and ABMOF18 dataset.
- ch.unizh.ini.jaer.projects.minliu source code

1 INTRODUCTION

COMPUTING optical flow (OF) is a fundamental problem of computer vision. There are a variety of algorithms for frame-based cameras. The most widely used method in computer vision is probably the efficient and highly optimized LK method [1] from OpenCV [2]. It is a sparse method that locally applies brightness constancy around detected feature points. Most recent developments have been used deep convolutional neural networks to compute dense flow; they achieve high accuracy but are extremely expensive, for example FlowNet2.0 [3] runs HD video at 8 FPS using several hundred watts of PC+GPU power.

Although LK is widely used, it fails when the images are over or underexposed, and when the images are too blurred to extract good features, and when these features have too much displacement between frames, as we demonstrate in Sec. 3. These scenarios arise in high speed vision, for example in fast drone flight, or under low lighting conditions, where the frame exposure increases, or under natural lighting conditions, where extreme lighting variations, glare, and lens flare are common.

The adaptive block matching optical flow (**ABMOF**) OF method proposes to address these problems. It is a semi-dense method that computes flow at points where brightness changes. These brightness changes come from a Dynamic Vision Sensor (**DVS**) silicon retina. The DVS is a new type of camera [4], [5], [6], [7], [8], [9] that provides sparse asynchronous data output, high dynamic range, high time resolution and low latency. Its output is a variable data-rate stream of timestamped pixel brightness change events. The DVS requires new algorithms to take advantage of these brightness change events. The DVS has been proposed for high-level vision algorithms such as localization [10], navigation [11], [12], landing [13] [14], visual odometry [15], [16], [17], and simultaneous localization and mapping [18]. Previous low-level algorithms for DVS OF are reviewed in Sec. 1.2.

The ABMOF algorithm originates from video compression motion estimation. The core of the algorithm is an event-driven computation of block matching optical flow that operates on variable duration time slice images of accumulated DVS events. ABMOF is directly targeted for efficient multiplier-free parallel hardware implementation using simple logic circuits. Our original BMOF [19] does not work well on complex scenes, but we here describe five improvements that form ABMOF:

- A new *AreaEventCount* slice rotation method correctly rotates slices that vary in density of features.
- A new feedback control mechanism for adapting slice duration achieves a target average match distance, increasing speed range and usability.
- A new adaptive event-skipping mechanism does not discard any sensor data but only computes OF when pipeline allows it.
- Using new multiscale time slices matches longer distances and lower spatial frequencies.

• *M. Liu and T. Delbruck are with Sensors Group, affiliated with the Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland: http://sensors.ini.uzh.ch. Corresponding author E-mail: minliu@ini.uzh.ch.*
This research is funded by Swiss National Center of Competence in Research Robotics (NCCR Robotics). We thank iniLabs for providing technical support, and L. Jira and T. Zaluska for help with recordings.

- Using a diamond search rather than full search improves search efficiency, e.g. by about 14X for search distances of 12 pixels.

Comparing both block matching and Lucas-Kanade methods on the adaptive slices shows improvement for both methods compared with the previous fixed slice durations.

1.1 DVS and DAVIS

DVS pixels independently react to brightness (log intensity) changes. If any pixel detects a brightness increase or decrease that exceeds a critical threshold amount, relative to the previously-memorized brightness, it generates an output event, and memorizes the new brightness value. Each event consists of a timestamp with microsecond resolution, an event address represented by x and y pixel location, and a polarity, where 1 means ON event increase and 0 means OFF event decrease. Comparing with a conventional camera, the DVS thus has worse spatial resolution but better temporal resolution. It also has lower latency and higher dynamic range.

The DAVIS [7] combines the DVS with conventional Active Pixel Sensor (**APS**) technology in the same pixel, using a shared photodiode. The DAVIS concurrently outputs DVS events and gray-scale image frames. While the DVS has high dynamic range (**DB**) (typically >120dB), the APS output has only limited DR of about 55dB. We use the APS output here to compare ABMOF with conventional LK on the frames.

1.2 Prior DVS Optical Flow

This section reviews previous DVS OF algorithms. They have been dominated by event-driven methods that operate on each DVS event.

[20] described an open-source algorithm (called **DS** in this paper) for time-of-flight DVS OF based on oriented edges detected by spatio-temporal coincidence. It works only for sharp edges and suffers from aperture problems since it is edge-based.

[21] adapted the frame-based LK algorithm (called **EBLK** in this paper) to DVS. It stores a fixed-queue length window of past events. For each new event, it computes the LK algorithm on a window of fixed time interval of a block of pixels surrounding the current event pixel. The gradient estimation precision is low due to quantization and small 5x5 window size. The small window size was used to limit the computation time in order to keep up with a high rate of events.

[22] proposed a contour-based method. In their work, they compared events-only method and events-frames-combined method. The difference is that by using events-only sensor, they need to reconstruct the contrast of the edge to localize the contour but it is not necessary for frames which have the absolute intensity. The optical flow estimation then is obtained from the contour width divided by the time interval.

[23] proposed a time-surface method (called **LP** in this paper) that combines the 2D events and timestamps into 3D space. Normal OF is obtained by robust iterative local plane fitting. It works well for sharp edges but fails with dense textures, thin lines, and natural scenes [24], [25] since these both produce complex structures that plane fitting does not model.

[26] proposed a more expensive phase-based method for high-frequency texture regions. They use normalized cross-correlation of to measure the pixel's timestamps' similarity and localize the contour. Once the contour is found, they use a Gabor filter to extract the local phase. The OF constraint that assumes the constancy of the spatio-temporal contours using the phase is formulated and is used to solve the normal OF flow.

[24] implemented and compared the DS, EBLK, and LP methods. It concluded that the existing algorithms were both computationally expensive and do not work well natural scenes and noisy sensor data. This paper also proposed an evaluation method and a provided a simple benchmark dataset with ground truth. The ground truth OF is obtained by constraining the camera motion to pure rotation and uses the camera's Inertial Measurement Unit (**IMU**) rate gyros to obtain global translational and rotational OF.

[27] proposed a frame-based variational algorithm that simultaneously estimates the optical flow, gradient map, and intensity reconstruction from DVS. Although the simultaneous constraints results in very regularized output, the results are not quantified, and the method is very expensive compared to others.

Though the main goal of [28] is for event-based feature tracking, it also proposed a pipeline to compute OF on corner points. They added another two assumptions: One is that events generated by the same point lie on a curve, and OF within a small spatial temporal window is constant. The OF problem is cast in an optimization framework and the expectation maximization (EM) algorithm computes the solution. It can run in real time with 15 features on a PC.

[25] recently reported the first CNN-based DAVIS OF architecture and published a useful dataset. It is trained by minimizing photometric loss from the DAVIS APS frames. It achieves the best published accuracy, but burns 50W to run at 25 Hz frame rate on a laptop gamer GPU.

The above methods are serial algorithms that solve linear or nonlinear constraints; some of them use iteration or exclusion of events to make the solutions more robust. All methods so far have required at least several us/event on a fast PC that consumes many tens of watts.

Hardware implementations: [29] developed a microcontroller-based embedded implementation of a time of flight (TOF) OF method. It works well for isolated points but not for dense textured scenes; it also has aperture problems with edges. A simplified version of [23] using 3x3 windows was implemented on FPGA in [30]. The small windows restricts it to sharp edges.

This paper extends on our previous FPGA BMOF implementation [19]. We used a method called block-matching. Block-matching was developed for motion estimation for MPEG video encoders and there are many silicon implementations. We demonstrated only a basic implementation for a small block size of 9x9 pixels using a single pixel shift in the NSEW compass directions and a single fixed time slice duration. It does not work well on most real scenes. But the advantage of block matching is that in hardware, large blocks can be matched with few clock cycles and simple logic. For example, 21x21 blocks can easily be implemented

by parallel logic circuits, and these large block sizes are important for good flow accuracy with noisy sensor data.

Here, as described in this paper, we have extended from the original implementation. The paper is organized as follows: Sec. 2 explains the idea of block-matching and our improvements. Sec. 3 shows experimental results, and Sec. 4 concludes the paper.

2 ABMOF ALGORITHM

The pipeline of ABMOF is summarized in Fig 1 and the time slices and block matching are illustrated in Fig. 2. When a new event arrives, the event's timestamp is used by the rotation logic to determine whether the event slice is to be rotated. If yes, the slices are rotated and the slice duration d or event count parameter K or k is adapted based on the current slices's OF distribution. The adapted slice duration is sent as an input to the rotation logic. The adaptation takes the OF distribution of the previous slice as the input. We use a dashed connection in the figure to represent their relationship. Details of the rotation logic are introduced in Sec. 2.2.

All the new events will be accumulated to multi-scale slices. If the system is busy, the OF calculation for the event is skipped. Otherwise it triggers the OF calculation. The event skipping mechanism is introduced in Sec. 2.5. After removing outliers, the OF histogram is updated.

All the parameters that are used in this paper are summarized in Table 1.

TABLE 1: Symbols, description, and typical values/units.

symbol	description	typical values (default)
$w \times h$	width × height of pixel array	346x260
d	slice duration	1–100 ms (50)
K	global event number	1k–50k events (10k)
k	area event number	100–1k events (1k)
a	area dimension subsampling	5 bits
b	block dimension	11–21 pixels (21)
r	search radius	4–12 pixels (4)
s	# scales	1–3 (2)
p	skip count on PC for real-time	30–1000
g	# bits for slice counts	1–7 (3)
g	# bits for slice counts	1–7 (3)
D	average match distance	ideally $r/2$
(v_x, v_y)	OF result	pixels/sec (pps)

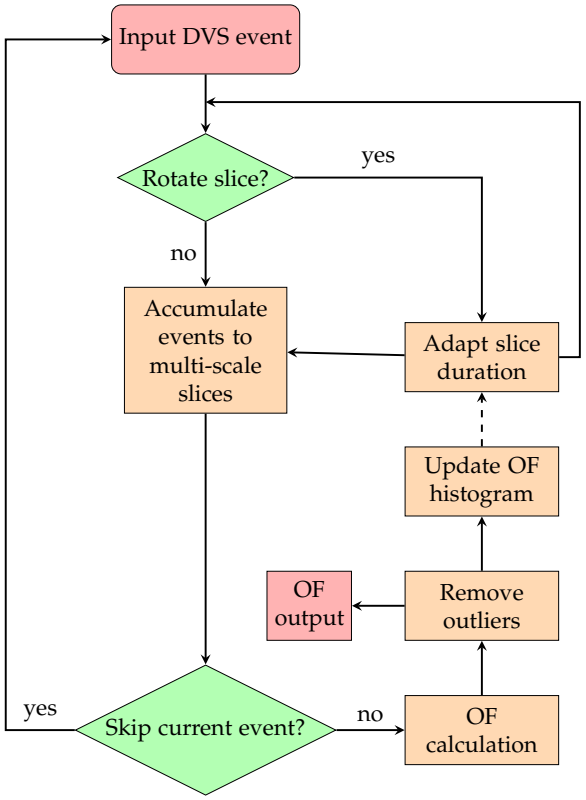


Fig. 1: The pipeline of our algorithm

2.1 Block-Matching DVS Time Slices

Fig. 2 shows the main principle of BMOF: Three time-slice memories store the events as 2D event histograms: Slice *t* accumulates the current events. Slices *t-d* and *t-2d* hold the previous two slices. *d* is the slice duration. When a new event arrives, it is accumulated to slice *t* by either incrementing the pixel value, or adding the polarity of the event to it. Which of these is done depends on if we ignore the event polarity. For the experiments in this paper, we usually ignored event polarity because the accuracy did not change significantly when we included it. Including polarity may enable better block matching, but it carries the price that one bit of the pixel memory is used for the sign bit. After accumulating the event, then the other two slices are then used to compute the OF based on the current event's location. When multi-scale slices are used (Sec. 2.4), then each slice is a pyramid of s slices.

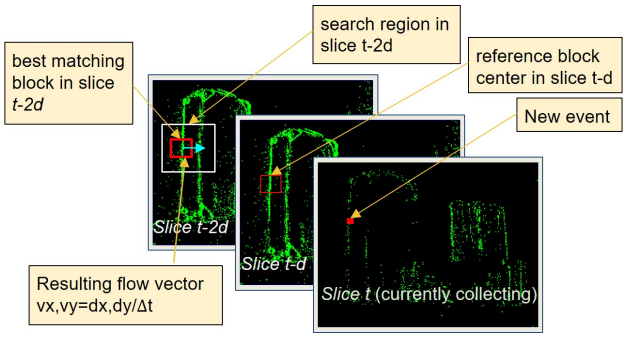


Fig. 2: BMOF block matching, on `boxes` from [24]

A reference block ($b*b$ pixels) is centered on the incoming event's location on the *t-d* slice map (red box in slice *t-d*). The best matching block on the *t-2d* slice is found based on Sum of Absolute Difference (**SAD**) inside a $(2r+1)^2$ search region, shown as a white rectangle in the *t-2d* slice. Thus the optical flow result is obtained by using these two blocks' offset (d_x, d_y), divided by the time interval Δt between these two slices. The time of each slice is taken as the average of the first and last timestamp of events accumulated to each slice.

The slices are rotated according to slice rotation logic (Sec. 2.2). The rotation discards the *t-2d* slice and uses its memory for the new slice *t*; similarly slice *t* becomes slice *t-d* and slice *t-d* becomes slice *t-2d*. In [19], the slice duration d was set by user manually. It is not convenient for general application since limits the speed range. In this paper, we propose several methods to adjust it adaptively.

2.2 Slice Rotation Methods

Slice rotation is the core part of our algorithm. It calculates when to rotate the slices to ensure good slice quality. Good slices should have sharp features, not too much displacement, and not be too sparse. This goal is achieved by feedforward and feedback control. We show the details of these two algorithms in the following subsections.

2.2.1 Feedforward Slice Rotation

The new events are accumulated into the latest slice, slice *t*. Slice *t* is only used for accumulation. After that, it will be rotated to be as a past slice and used for OF calculation.

In our original BMOF work [19], we implemented the method *ConstantDuration*, where each slice has the same duration d. Another obvious method is to rotate slices after a constant number K of events have been accumulated, called *ConstantEventNumber*.

- *ConstantDuration*: Here the slices are accumulated to time slices uniformly with duration d. This method is what we reported before and corresponds most closely to conventional frame based methods. It has the disadvantage that if the scene motion is too fast, then the movement between slices may be too large to be matched using a specified search distance. If the movement is too slow, then the features may not move enough between slices, resulting in reduced flow speed and angle resolution.
- *ConstantEventNumber*: Here the slices are accumulated until they contain a fixed total count of DVS events K. If K is large then the slices will tend to have larger d. But if the scene moves faster, then the rate of DVS events also increases, which for fixed K will decrease d. Therefore the *ConstantEventNumber* method automatically adapts d to the average overall scene dynamics.

A drawback of the *ConstantEventNumber* method is its global nature. For scenes which have lots of or very few textures, it is impossible to set a suitable global K. In order to address this problem, we propose a new rotation method called *AreaEventNumber*.

- *AreaEventNumber*: Instead of rotating the slices based on the sum of the whole slice event number, *AreaEventNumber* will trigger the slice rotation once any one of the area's event number (Area Event Counters) exceeds the threshold value k. Each area is $(w \times h)/2^a$ pixels, i.e. 10×8 pixels.

By using the *AreaEventNumber* method, slice rotation is data-driven by the accumulation of DVS events, but adapts the slice durations to match the area of the scene which has the most DVS activity. This adaptation prevents under-sampling that causes displacement that is too large to match between slices. Compared with *ConstantEventNumber* method, it preserves the advantage that the generated slices adapt to the scene dynamics. The local adaptability makes the slices more robust to variation and distribution of scene texture.

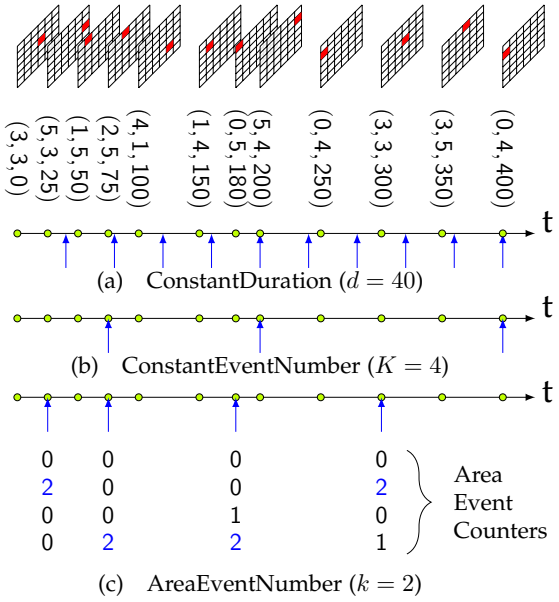


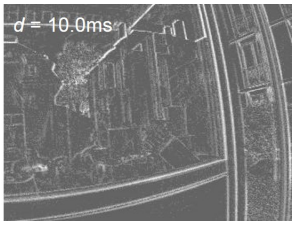
Fig. 3: Three feedforward slice rotation methods. The event stream is at the top of the figure. The information including event address (x,y) and timestamp is shown under the event stream. An example of these three slice methods is demonstrated here, with (a) slice duration $d = 40$, (b) global event number $K = 4$ and (c) area event number $k = 2$.

To make it even more robust and adaptive, the slice event number k is also adaptive to the scene. When the scene moves fast, the parameter k will be increased. Otherwise, it is decreased. Adaptation of k is further described in Sec. 2.2.2.

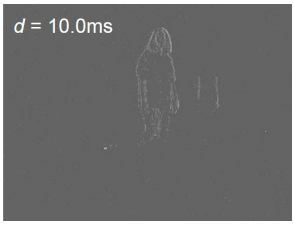
An example to demonstrate these three methods is shown in Fig 3. The blue arrows pointing to the three time axes represent these three rotation method results. It is obvious that both the time interval and the event number interval are fixed for *ConstantDuration* and *ConstantEventNumber*. However, both of them vary in the *AreaEventNumber* method which makes it more adaptive to the dynamic scene.

In Fig 4, we compare these three methods on two different scenes, one have sparse textures and the other have dense textures. Among them, Figs. 4a, 4c, and 4e are obtained from the same dense scene by the three different methods. Figs. 4b, 4d, and 4f are obtained from a sparse scene. Both of the dense and sparse scenes use the same parameter for every method which is $d = 10ms$ for *ConstantDuration*, $K = 10000$ for *ConstantEventNumber* and $k = 700$ for *AreaEventNumber*. The resulting slice durations are shown overlaid on each scene.

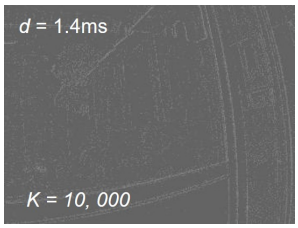
Neither *ConstantDuration* nor *ConstantEventNumber*



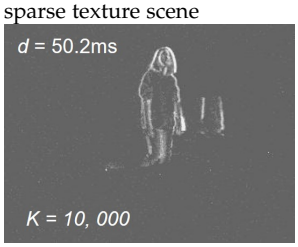
(a) *ConstantDuration* on dense texture scene



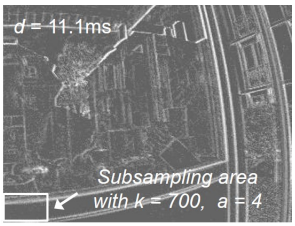
(b) *ConstantDuration* on sparse texture scene



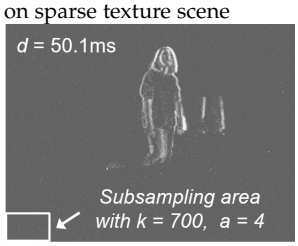
(c) *ConstanEventNumber* on dense texture scene



(d) *ConstanEventNumber* on sparse texture scene



(e) *AreaEventNumber* on dense texture scene



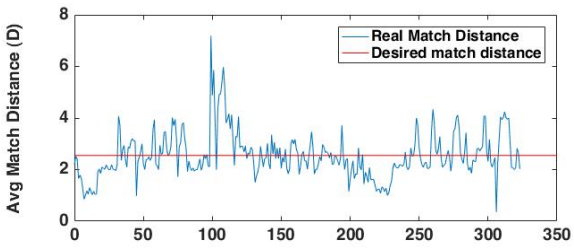
(f) *AreaEventNumber* on sparse texture scene

Fig. 4: Comparison between event slices generated by three methods.

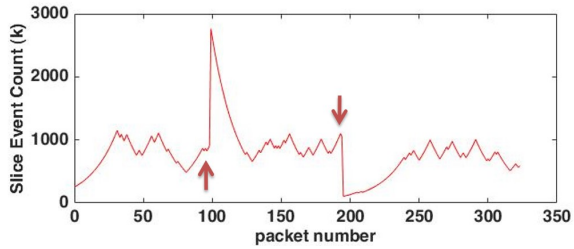
work well on both of these two scenes with fixed values of d or K. For example, *ConstantDuration* fails in the sparse scene because d was set for the faster motion in Fig. 4a and the duration was too short for the slower motion in Fig. 4b. *ConstantEventNumber* makes the slice too short in duration in the dense scene in Fig. 4c, because K was set to make a good slice for Fig. 4d. However, *AreaEventNumber* with fixed parameter k functions well on both of scenes, because it correctly creates the Fig. 4f slice after being set for the dense scene in Fig. 4e. It shows that *AreaEventNumber* is more robust to dynamic scene content.

2.2.2 Feedback Control of Slice Duration

Another method to automatically adjust the slice duration is possible via feedback control. An optical flow distribution histogram is reset after each slice rotation and then collects the distribution of OF results. The histograms average match distance D is calculated. If $D > r/2$ where r is the search radius, it means that the slice duration is too long, and so the slice duration or event number is decreased. Otherwise, if $D < r/2$, then it indicates the slices are too brief in duration, and the slice duration or event number is increased. It is possible that slice durations that are too brief or lengthy result in OF results of very small matching distance that are the result of a bias in the search algorithm towards zero motion (small match distance). Stability is improved by limiting the slice duration range within application-specific limits. For the control policy, we so far used bang-bang control. A fixed factor of $\pm 5\%$ adjusts the slice duration, where the sign of the relative change of duration is the sign of $r/2 - D$. More sophisticated control policies are clearly possible, since the value of the error is directly predictive of the necessary change in the duration.



(a) OF result's match distance



(b) Feedback on slice event count

Fig. 5: Feedback on slice event number. (a) shows the OF result's real match distance and its desired match distance. (b) represents the slice event count number.

Since the principle of the feedback control on event number and slice duration is similar, we show only an example of feedback control on event number k here. The data in Fig. 5 shows an example of event number control using the *AreaEventNumber* rotation policy with feedback control of k. Fig. 5a shows the average OF match distance D. The feedback control of event number holds D at its $r/2$ value of about 2.5 pixels. Fig. 5b shows the event number k. It has a steady state value of about 1000. Around packet 100 (1st arrow), k was manually perturbed to a large value, resulting in a increase in D, but it rapidly returns to the steady state value. At around packet 200 (2nd arrow), k was manually reduced to a small value, resulting in a small D of about 1 pixel. Again D returns to the steady state value. This data shows the stability of the event number control with this feedback mechanism.

2.3 Search Method

The implementation of [19] searched only the target block and its 8 nearest neighbors. An improvement is offered by extending the search range to a larger distance range r. The block matching search method can be done by exhaustive full search. It has the best search accuracy but is expensive since the cost grows quadratically with r. A more efficient

method is diamond search [31], which we implemented. It makes a trade off between computation and accuracy. Our results shows that it has about 90% chance to hit on the best matching block with a cost 14X less than the full search, for $r = 12$. Using the diamond search improves the algorithm's real-time performance significantly.

2.4 Multi Scale and Multi-Bit Time Slices

A limitation of the approach described so far is the limited dynamic speed range of the method, since matching can only cover a spatial range of square radius r around the reference location. One way to increase the search range by a factor of 2^s with only s linear increase in search time is to use a multi-scale pyramid [32]. In this method, events are accumulated into a stack of time slices. Each slice in the stack subsamples the original event addresses in x and y directions by a factor of 2 more than the previous scale. I.e. if $s = 0$ means the original full resolution scale, then events are accumulated into scale s by first right shifting the event x and y addresses by s bits, and then accumulating the resulting event into the scale s slice, which has only $1/2^s$ as many pixels for each dimension. For example, in the $s = 1$ scale slice, each pixel accumulates events from a 2x2 pixel region in the full resolution original pixel address space. To prevent saturation, we use multiple bits g for each value; for example $g = 3$ allows up to 7 unsigned events for each pixel when we ignore the event polarity, or up to ± 2 events when we use polarity. Thus the total slice memory required for an N pixel sensor is $3Ng\sum_{m=0}^{s-1} 2^{-m}$ bits.

To compute the OF, each event is processed independently for each scale. The match that has the minimum SAD is selected as the OF. Using multiple scales is beneficial particularly in noisy situations, where the event flow is sparse. The binning of events helps to find good matches.

2.5 Adaptive Event skipping

For high speed or densely textured scenes, the event rate becomes high. If we still compute OF for each event the real-time performance will be influenced dramatically and the algorithm quickly falls behind the actual incoming event rate. To address this problem, we propose an event skipping method. If the processing time is higher than a threshold we set, the following events do not have their OF calculated. However, they are still accumulated to the current slice. By doing this, we can get a trade off between the OF density and the real-time performance. The adaptive event skipping algorithm uses a skip parameter p, which is increased or decreased depending on the frame rate. If the actual frame rate is slower than the desired frame rate set by the user, it means it takes too much time to process the event, then p increases. Otherwise, it decreases. On a Corei7-975 PC in Java 1.8, the ABMOF implementation requires about 15 us per event with the default parameters in Table 1 and $p = 1$. With $p = 1000$, the time drops to an average of about 260 ns/event; there is some overhead for each packet and for slice rotation, which is why it only drops by only a factor of 600X. It is also easy to implement in hardware. A FIFO forms a buffer for the incoming events. The event skipping will be designed as a switch and will be connected to the FIFO half-full flag. If the FIFO is half-full, it means the processing time is falling behind, and event skipping will be enabled.

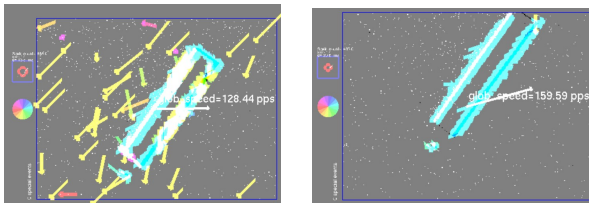
2.6 Outlier Rejection

To improve the accuracy, we developed outlier rejection to filter out OF results with poor matching quality. We use two parameters to reject the outliers. One parameter is `validPixOccupancy`; it determines the percentage of valid pixels in two blocks that will be compared. Valid pixels are the pixels where events were accumulated. The reason for setting this parameter is sometimes the blocks are too sparse, which makes the distance metric get a meaningless result. By only calculating matching for blocks that are filled with sufficient valid pixels, we can reject misleading results.

A second outlier rejection parameter is called `maxAllowedSadDistance`. The minimum distance between the reference block and the candidate block must be smaller than `maxAllowedSadDistance`, otherwise the OF event will be rejected. Thus, the best matching search block may actually be a poor match, and `maxAllowedSadDistance` allows rejecting best matches if the match distance is too large.

The effect of these parameters is shown in example data in Fig 6 from a simple case of a black bar moving up and to the the right. The flow results are visibly cleaner using these outlier rejection criteria.

Both outlier rejection mechanisms are easily implemented in hardware. For example, the valid pixel occupancy can be realized by pixel subtraction units that output a large value if both operands are zero. The confidence threshold can be realized by a comparator on the final best match output result that flags a distance that is too large.



(a) Without outliers rejection (b) With outliers rejection

Fig. 6: Example of outlier rejection using `maxAllowedSadDistance` and `validPixOccupancy`. (a): without outlier rejection. (b): using outlier rejection with `maxAllowedSadDistance=0.5` and `validPixOccupancy`=0.01.

3 Results

The algorithms were implemented in jAER [33]. In this section, we show several experiments to validate the ABMOF algorithm. They can be classified into two types. The first type are the experiments with ground truth OF. We tested it on two datasets: `slider_hdr_far` is from [34]. It shows a flat poster scene with fixed depth of 58 cm where the camera is moved laterally by a motorized cart, resulting in uniform flow of about 90 pixels per second (pps). It was recorded with high lighting contrast. `pavement_fast` is a scene with extremely fast flow of 34k pps recording from a car, with a down-looking camera recording an asphalt pavement. The files converted to jAER aedat format are

included in our dataset. We show both the qualitative and quantitative results in these experiments; see Sec. 3.1

In the second type of experiment, we test our algorithm on several complex scenes. They consist of camera rotation over gravel (`gravel`), flow through an indoor office environment (`office`), and uniform flow created by a variety of shapes (`shapes`, from [34]). The dataset is provided[1] to support the tests for other future algorithms. Due to unknown ground truth in these files, we show only a comparison between the ABMOF and Lukas Kanade results using the generated slices for these data; see Sec. 3.2.

For the new data we gathered for this work, we used an unpublished advanced DAVIS with 346x260 pixel resolution and integrated on-chip APS readout circuits, allowing a maximum APS frame rate of about 50 Hz. This DAVIS346 has pixels with integrated microlenses, optimized photodiodes, and antireflection coating, which together increase the effective quantum efficiency to about 24% compared with the previous DAVIS240C QE of 7%.

For the `slider_hdr_far` data, the groundtruth camera position is provided for each time point and the scene has a provided uniform depth. By using the camera calibration data and pinhole camera model, we converted the pose groundtruth data to a global optical flow groundtruth.

For `pavement_fast`, we manually measured the flow using a jAER [33] software filter called *Speedometer*, which allows using the mouse to mark a moving feature point at different time points and measures the distance and time between these marks.

3.1 Type I experiment result

We show the results of type I experiments in this subsection. We measured four metrics to evaluate the algorithms. They are event density (**ED**), translational global flow (**GF**), Average Endpoint Error (**AEE**) and Average Angular Error (**AAE**). ED is the fraction of DVS events that result in OF results. DVS events are skipped because block matching fails to pass outlier rejection tests; we set $p = 1$ for these tests. ED relates to the density of the flow computation. LK has very low density because it relies on features. An ED of 100% means that all pixel brightness changes result in OF events. AEE and AAR are defined for DVS OF in [24].

Besides the ABMOF, we also implemented the Lucas Kanade (**LK**) OF calculation based on our generated adaptive event slices using OpenCV and we call it ABMOF_LK. ABMOF_LK uses our algorithm to set the time slice duration, and these generated slices are treated as conventional gray scale image frames. In ABMOF_LK, corners are first extracted by Shi-Tomasi corner detector [35] and then they are are passed to the LK tracking algorithm implemented in OpenCV [2]. LK estimates the OF result based on these features.

We also compared the AMBOF OF methods with previously published implementations from [24]: *DirectionSelectiveFlow* (DS) [20], the event-based *LucasKanadeKFlow* (EBLK) [21], and *LocalPlanesFlow* (LP) [23].

1. ABMOF dataset README link for review

3.1.1 `slider` scene

Fig. 7 shows the qualitative results of ABMOF and ABMOF_LK on the `slider_hdr_far` data. This is a high dynamic range scene of a flat poster with uniform flow about about 90 pps. Because of the lighting contrast, the APS images are sometime extremely over- or underexposed, but the DVS events respond to the local brightness changes. Table 2 reports the quantitative comparison. By the VO groundtruth to OF groundtruth conversion, we can compare them over time, as shown in Fig. 8. Table 2 shows that ABMOF_LK's GF error on the `slider_hdr_far` data is less than 1pps and ABMOF's GF error is less than 4pps. ABMOF_LK is more accurate than ABMOF, but has much lower ED. Fig. 8a shows a very clear periodic oscillation in v_x for both *ABMOF* methods, which is caused by the simple bang-bang control of the slice duration coupled with match distance quantization. This oscillation is confirmed by the trace *ABMOF fixed with 45ms*, where we fixed $d = 45$ms; its v_x flow is a bit too small because of the quantization of match distance. The conventional LK method on the frames also obtains the average correct flow (and does not have the controller oscillation), but as seen in Fig. 7c, this estimate is sometimes based on a single keypoint. That is the cause the outliers for *Frame_based LK* in Fig. 8b around 5s and 6s where the frame LK method suffered large aperture error.

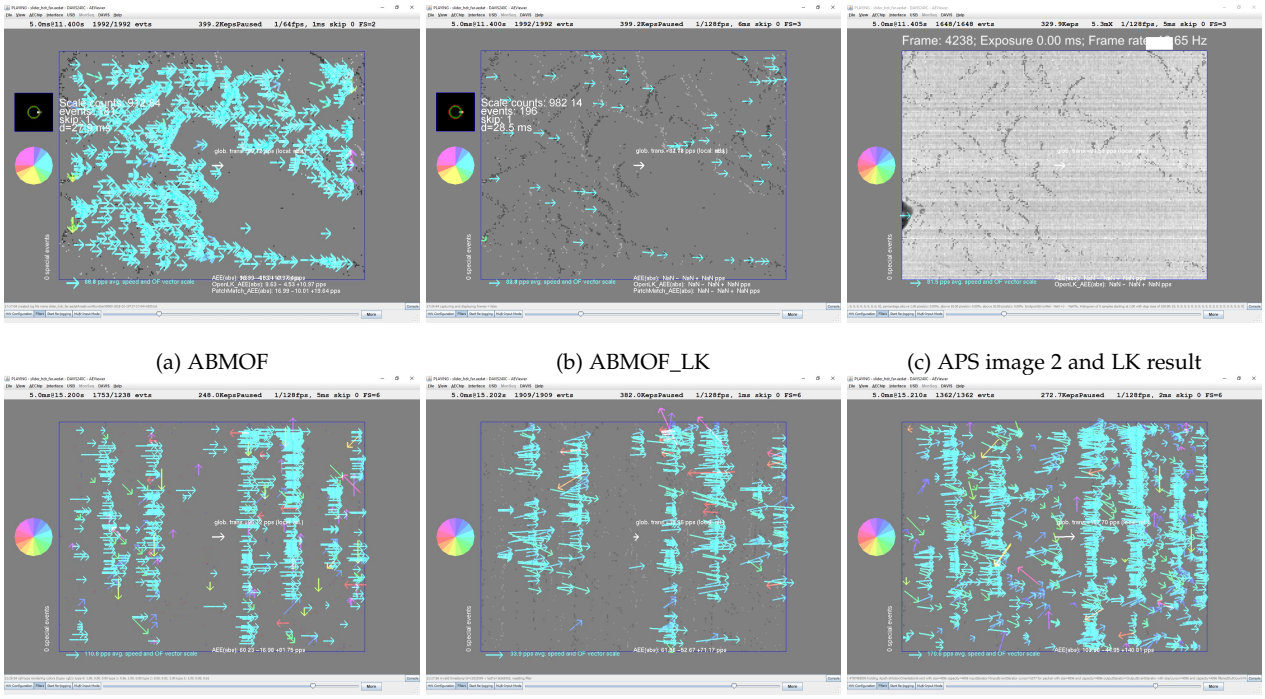
This experiment validates that the slice rotation methods result in quantitative flow magnitude that is the same as from Frame based LK . The ABMOF methods are oscillatory using the current k controller, but have much higher density than the frame-based LK method. All ABMOF methods are all much more accurate and less noisy than the prior DS, EBLK, and LP methods.

3.1.2 `pavement_fast` scene

Fig. 9 shows the results of a very high speed experiment on `pavement_fast`, which was recorded from a car with a down-looking camera aimed at the asphalt pavement road surface. The global flow is an extremely fast 32k pps, which means that a pixel crosses the 346-pixel array in about 10 ms. Figs. 9a - 9b compares ABMOF and ABMOF_LK on DVS time slices, and Fig. 9c shows conventional LK on successive DAVIS APS images (the 2nd image is shown under the flow result). Both ABMOF and ABMOF_LK correctly measure the true flow using a slice duration of only 450 us, equivalent to a frame rate of 22 kHz and a 14 pixel displacement between slices. The consecutive APS image frames were collected at the maximum frame rate of 50 Hz, but because the motion is so fast, even the short DAVIS global shutter exposure of 0.7 ms resulted in visible image blur of several pixels. And since the consecutive frames are separated by 20ms, the images are completely uncorrelated and the resulting flow is meaningless as seen in Fig. 9c.

3.2 Type II experiment result

The final experiments are from natural scenes that contain a range of directions and speeds. Since we lack ground truth OF for these natural scenes, we only show the qualitative comparison of ABMOF and ABMOF_LK. These results are shown in Fig 10. We use vectors to represent the OF result; color also shows the direction. For clarity, we set $p = 1000$



(a) ABMOF (b) ABMOF_LK (c) APS image 2 and LK result

(d) DS (e) EBLK (f) LP

Fig. 7: Result of ABMOF, ABMOF_LK and standard LK on image frames on `slider_hdr_fast`. For 7a and 7b, we use *AreaEventNumber* with feedback enabled, $r = 4$, and $s = 2$, and used standard LK on successive APS images in 7c.

TABLE 2: Comparison of algorithm's overall accuracy on `slider_hdr_far`.

method	event density	global flow (pps)	AEE (pps)	AAE (°)
Groundtruth	-	$[90.50, 0] \pm [0.43, 0]$	-	-
ABMOF_LK	0.39%	$[89.75, 0.44] \pm [6.30, 3.56]$	8.75±27.51	2.95±3.41
ABMOF	37.96%	$[86.85, 0.17] \pm [8.46, 1.25]$	12.68±16.28	3.66±8.31
Frame_based_LK	-	$[89.51, 0.20] \pm [3.203.48]$	5.47±42.07	1.30±4.72
DS ([20], [24])	49.86%	$[74.97, 2.98] \pm [17.42, 4.79]$	57.71±53.31	21.46±39.13
EBLK ([21], [24])	17.53%	$[28.06, -0.11] \pm [4.09, 1.32]$	60.32±15.92	13.52±25.51
LP ([23], [24])	83.88%	$[161.14, 11.69] \pm [8.67, 12.13]$	99.00±75.86	16.99±24.41

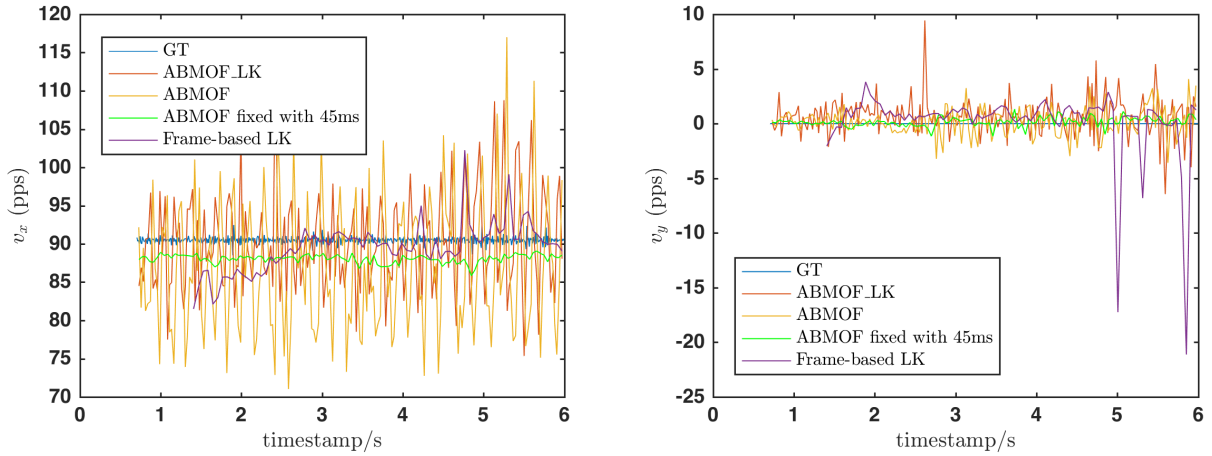
for ABMOF. The ABMOF and ABMOF_LK produce very similar OF for these natural scenes. For `shapes`, ABMOF flow is quite dense along object edges and the large block size of 21 pixels results in true OF rather than normal flow. For this same scene, ABMOF_LK attaches OF only to object corners; ABMOF_LK misses the OF on the upper right ellipse, but the overall flow is more uniform.

4 Conclusion and Discussion

We proposed improvements for the basic BMOF algorithm for DVS. A new pipeline adjusts the slice duration based on the local movement rather than the global motion. Using multi-scale bitmaps allows a larger range of movement speeds to be economically computed and makes the operation more robust to noisy sensor data. A feedback mechanism for slice duration makes the average displacement between two slices close to the half of the search distance. These improvements on the basic BMOF achieves a good trade off between good quality of optical flow estimation and a low computation cost. We also compared our generated adaptive slices with the constant time duration slices.

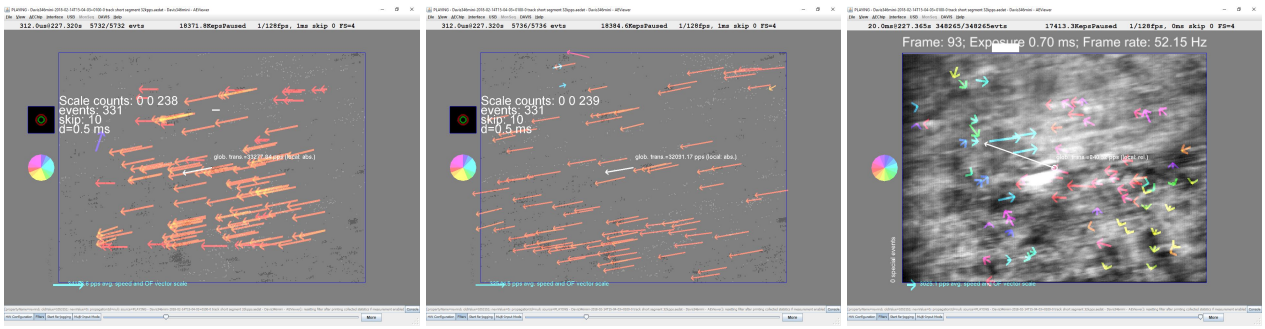
The reason we implemented the ABMOF_LK and show the results here is to show the event slices generated by our adaptive method are robust to different scenes and can provide better grayscale images for frame-based algorithms to process. Although ABMOF_LK clearly produces more accurate OF, it is a frame-based OF method that must process every pixel in each frame and produces very sparse output. The gradient-based LK algorithm is also much more difficult and expensive to implement in logic circuits compared with ABMOF.

The results shows that the accuracy of the algorithm



(a) v_x for `slider_hdr_far` (b) v_y for `slider_hdr_far`

Fig. 8: Comparison of measured and ground truth flow between OF methods on `slider_hdr_far`



(a) ABMOF (b) ABMOF_LK (c) APS image 2 and LK result

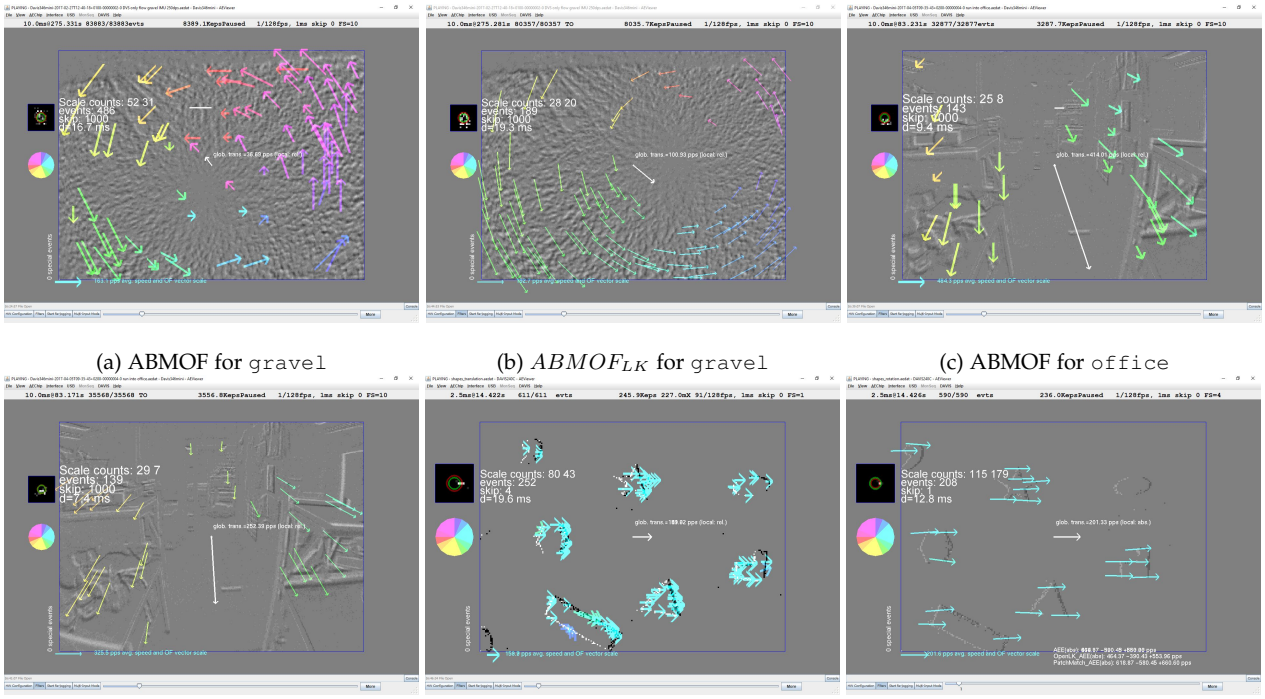
Fig. 9: Result of ABMOF, ABMOF_LK and standard LK on image frames on `pavement_fast`. For 9a and 9b, we fixed $d = 450$ us, $r = 12$, and $s = 3$, and used standard LK on successive APS images in 9c.

mainly depends on the quality of the generated slices. By using the rather large block dimension $r = 21$, ABMOF avoids most aperture problems except on extended edges longer than the block dimension.

The dynamic range of speeds allowed by ABMOF is determined by the search distance r and the number of scales s. In any one moment, the range of match distances spans from 0 to $r2^s$ pixels along each axis, e.g., with search distance $r = 4$ and $s = 3$ scales the OF can span 0 to ± 32 pixels, although the speed resolution decreases as the scale increases. In our experience this is sufficient range to cover real scenes where the camera is rotating, or translating through a cluttered environment with nearby and far objects. With the adaptive slice duration, fast motion can result in slice durations that are fractions of a millisecond, as in the `pavement_fast` example of Sec 3.1.2, allowing measurement of speeds > 10k pps. This result was previously only the domain of high-end gaming mouse sensors such as [36]; these are capable of up to several thousand FPS but require active illumination and have less than $50x50$ pixel arrays that are more than a 100 times fewer pixels than the DAVIS used here; also, they only measure global translational flow.

By extrapolating the FPGA hardware implementation costs from [19], we estimate that ABMOF can be implemented on a medium sized FPGA fabric, such as Xilinx Zynq-7000 family chip XC7Z100. The total/percent utilization will require about 35k/6% Flip-Flops, 100k/36% Look-Up Tables and 400 kb/1% block RAM. In addition, the design from [19] is not optimized. The resulting IP block could later be integrated together with the sensor in a custom digital core.

The most widely used applications of OF are in optical mice and video compression, where probably at least a billion ICs have been produced that estimate motion based on block matching. In robotics, most visual odometry (**VO**) pipelines do not currently use OF, but an economical implementation could enable direct VO based on DVS in hardware, rather than the impressive but expensive software solutions [15], [37], [38]. Recent success in combining DVS with convolutional neural networks (CNNs) by using constant event number frames [39], [40], [41], [42] also can benefit from the smarter ABMOF slice methods, and the OF



(a) ABMOF for `gravel` (b) $ABMOF_{LK}$ for `gravel` (c) ABMOF for `office`

(d) ABMOF_LK for `office` (e) ABMOF for `shapes` (f) ABMOF_LK for `shapes`

Fig. 10: Result of the algorithms in different scenes. All scenes were captured using identical $s = 2$ scales, block size $b = 21$ pixels, using diamond search, with search distance $r = 4$ pixels and using feedback control of *AreaEventNumber* k. OF color and angle represents direction according to the color wheel and vectors' length means OF speed relative to the scale shown at bottom left of each frame. The histogram above each color wheel shows the OF distribution and mean match distance (green circle). The white arrow from center of image shows global average flow. The white statistics text shows the number of OF events for each scale, the number of OF events, the current skip count p, and the last slice duration d.

could provide useful input channel information to better enable dynamic scene analysis.

REFERENCES

[1] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[2] "OpenCV: Optical Flow." [Online]. Available: https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html

[3] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks," *arXiv:1612.01925 [cs]*, Dec. 2016, arXiv: 1612.01925. [Online]. Available: http://arxiv.org/abs/1612.01925

[4] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120 dB 15 s latency asynchronous temporal contrast vision sensor," vol. 43, no. 2, pp. 566–576. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4444573

[5] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-Based Neuromorphic Systems*. John Wiley and Sons Ltd., UK. [Online]. Available: 10.1002/9781118927601

[6] T. Delbruck, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2426–2429.

[7] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240x180 130 dB 3 us latency global shutter spatiotemporal vision sensor," vol. 49, no. 10, pp. 2333–2341.

[8] C. Li, C. Brandli, R. Berner, H. Liu, M. Yang, S.-C. Liu, and T. Delbruck, "An RGBW color VGA rolling and global shutter dynamic and active-pixel vision sensor," in *2015 International Image Sensor Workshop (IISW 2015)*. imagesensors.org, pp. 393–396.

[9] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," in *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, pp. 2130–2133.

[10] D. Weikersdorfer and J. Conradt, "Event-based particle filtering for robot self-localization," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 866–870.

[11] E. Mueggler, G. Gallego, and D. Scaramuzza, "Continuous-time trajectory estimation for event-based vision sensors," in *Robotics: Science and Systems XI*, no. EPFL-CONF-214686, 2015.

[12] G. Gallego, J. E. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza, "Event-based, 6-DOF camera tracking for high-speed applications," *arXiv preprint arXiv:1607.03468*, 2016.

[13] G. De Croon, H. Ho, C. De Wagter, E. Van Kampen, B. Remes, and Q. Chu, "Optic-flow based slope estimation for autonomous landing," *International Journal of Micro Air Vehicles*, vol. 5, no. 4, pp. 287–297, 2013.

[14] B. J. P. Hordijk, K. Y. Scheper, and G. G. de Croon, "Vertical landing for micro air vehicles using event-based optical flow," *arXiv preprint arXiv:1702.00061*, 2017.

[15] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based visual inertial odometry."

[16] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, "Low-latency visual odometry using event-based feature tracks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 16–23.

[17] A. Censi and D. Scaramuzza, "Low-latency event-based visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 703–710.

[18] D. Weikersdorfer, R. Hoffmann, and J. Conradt, "Simultaneous localization and mapping for event-based vision systems," in *International Conference on Computer Vision Systems*. Springer, 2013, pp. 133–142.

[19] M. Liu and T. Delbruck, "Block-matching optical flow for dynamic vision sensors: Algorithm and FPGA implementation," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[20] T. Delbruck, "Frame-free dynamic digital vision," in *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, 2008, pp. 21–26.

[21] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, no. 30, pp. 32–37, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608011002930?via%3Dihub

[22] F. Barranco, C. Fermüller, and Y. Aloimonos, "Contour motion estimation for asynchronous event-driven cameras," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1537–1556, 2014.

[23] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 2, pp. 407–417, 2014.

[24] B. Rueckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in Neuroscience*, vol. 10, p. 176, 2016. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2016.00176

[25] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras." [Online]. Available: http://arxiv.org/abs/1802.06898

[26] F. Barranco, C. Fermuller, and Y. Aloimonos, "Bio-inspired motion estimation with event-driven sensors," in *International Work-Conference on Artificial Neural Networks*. Springer, 2015, pp. 309–321.

[27] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous optical flow and intensity estimation from an event camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 884–892.

[28] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based feature tracking with probabilistic data association," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 4465–4470.

[29] J. Conradt, "On-board real-time optic-flow for miniature event-based vision sensors," in *Robotics and Biomimetics (ROBIO), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1858–1863.

[30] M. Tun Aung, R. Teo, and G. Orchard, "Event-based Plane-fitting Optical Flow for Dynamic Vision Sensors in FPGA," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, 2018.

[31] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," in *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on*, vol. 1. IEEE, 1997, pp. 292–296.

[32] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.

[33] T. Delbruck. (2007) Java AER framework. [Online]. Available: https://jaerproject.org

[34] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017.

[35] J. Shi *et al.*, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, 1994, pp. 593–600.

[36] PixArt Imaging Inc., "PixArt PMW3360 Optical Gaming Navigation Sensor," 2014. [Online]. Available: https://docs.google.com/viewer?url=http%3A%2F%2Fwww.pixart.com.tw%2Fupload%2FPMS0058-PMW3360DM-T2QU-NNDS-R1.30-06042016_20160902201411.pdf

[37] H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization," in *British Machine Vis. Conf.(BMVC)*, vol. 3, 2017.

[38] A. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? combining events, images, and IMU for robust visual SLAM in HDR and high speed scenarios." [Online]. Available: https://www.ifi.uzh.ch/dam/jcr:ce005b02-60da-43cd-870f-eb7d2fdd1619/RAL18_VidalRebecq.pdf

[39] D. P. Moeys *et al.*, "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, Jun. 2016, pp. 1–8. [Online]. Available: https://drive.google.com/open?id=0BzvXOhBHjRheVWVoRlBLUGViUlk

[40] I.-A. Lungu, F. Corradi, and T. Delbruck, "Live Demonstration: Convolutional Neural Network Driven by Dynamic Vision Sensor Playing RoShamBo," in *2017 IEEE Symposium on Circuits and Systems (ISCAS 2017)*, Baltimore, MD, USA, 2017. [Online]. Available: https://drive.google.com/file/d/0BzvXOhBHjRheYjNWZGYtNFpVRkU/view?usp=sharing

[41] A. Amir *et al.*, "A Low Power, Fully Event-Based Gesture Recognition System," 2017, pp. 7243–7252. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Amir_A_Low_Power_CVPR_2017_paper.html

[42] K. D. Fischl *et al.*, "Neuromorphic self-driving robot with retinomorphic vision and spike-based processing/closed-loop control," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2017, pp. 1–6.