HashHeat: An O(C) Complexity Hashing-based Filter for Dynamic Vision Sensor

Shasha Guo[1] Ziyang Kang[1] Lei Wang Shiming Li Weixia Xu

College of Computer Science and Technology
National University of Defense Technology
Changsha, Hunan, China. 410073
e-mail: guoshasha13, kangziyang14@nudt.edu.cn

Abstract— Neuromorphic event-based dynamic vision sensors (DVS) have much faster sampling rates and a higher dynamic range than frame-based imagers. However, they are sensitive to background activity (BA) events which are unwanted. We propose *HashHeat*, a hashing-based BA filter with O(C) complexity. It is the first spatiotemporal filter that doesn't scale with the DVS output size N and doesn't store the 32-bits timestamps. *HashHeat* consumes 100x less memory and increases the signal to noise ratio by 15x compared to previous designs.

I. Introduction

Images are widely utilized in deep learning. They are generally collected by frame-based image sensors, which have several common drawbacks like constant high power dissipation, sensitivity to bright ambient lighting, and low capability to capture fast objects. Neuromorphic event-based sensors are promising to address these problems.

Research on neuromorphic event-based sensors ("silicon retinae") started a few decades back [1]. Recently, the technology has matured to a point where there appears some commercially available sensors. Some of the popular sensors are Dynamic Vision Sensor (DVS) [2], Asynchronous Time-based Image Sensor (ATIS) [3], the sensitive DVS [4], and the Dynamic and Active pixel Vision Sensor (DAVIS) [5]. These sensors have several possible applications such as pose estimation [8], gesture-based remote control [9] and high speed corner detection [10].

Different from conventional frame-based imagers that work by sampling the scene at a fixed temporal rate (typically 30 frames per second), these sensors detect dynamic changes in illumination. This results in a higher dynamic range, higher sampling rate, and lower power consumption. However, they also produce background activity (BA) events even under constant illumination, caused by temporal noise and junction leakage currents [2], [6], [7].

There exist many noise filtering methods for event-based data, such as the Nearest Neighbor (NNb) filters [6], [11], [12] based on the spatiotemporal relation and some variations of NNb filters [13].

However, these spatiotemporal filters are memory intensive. For a DVS with N×N pixels, the minimum space complexity is O(N) [14], which puts a burden on the limited programmable logic (PL) embedded in the sensor head. There are two main reasons for the high space complexity of the current spatiotemporal filters. First, the memory occupation of these filters are dependent on the DVS output size N for the convenience of detecting spatiotemporal correlation. Second, memory cells of these filters are designed to store the timestamp and possible more information, i.e., at least 32-bits per cell. These two characteristics limit the possibility of further reducing memory consumption.

To tackle these challenges, we find that hashing functions, applied in Nearest Neighbor (NNb) search algorithms and collection element detection algorithms, can maintain the adjacency information and help to achieve high space efficiency. Inspired by these features, we design a spatiotemporal BA filter based on hash computing to obtain low memory cost. Our contributions are as follows. First, we propose *HashHeat*, an O(C) complexity spatiotemporal BA filter which aggressively saves memory cost. As we know, it is the first spatiotemporal filter with a constant space complexity and the first without storing the timestamp. C is to demonstrate that the space complexity is smaller than the O(N) proposed in [14]. Second, we propose a hardware design which can process an event at the latency of 10 ns. Third, *HashHeat* increases the signal to noise ratio by about 1.2x to nearly 15x compared with other baseline filters on a real dataset.

II. Background

A. DVS

The DVS128 [2] sensor is an event-based image sensor that generates asynchronous events when it detects the changes in log intensity. If the change of illumination exceeds an upper or lower threshold, the DVS128 will generate an "ON" event or "OFF" event respectively.

To encode all the event information for output, the DVS sensor uses Address Event Representation (AER) protocol [20] to create a quadruplets $e(p, x, y, ts)$ for each event. Specifically, p is polarity, i.e., ON or OFF, x and y are the two coordinates of the event respectively, and ts is a 32-bits timestamp, i.e., the timing information of an event.

B. BA Events

BA events are caused due to thermal noise and junction leakage currents. These events degrade the quality of the data and further incurs unnecessary communication bandwidth and computing resources. The BA and the real activity events differ in that the BA event lacks temporal correlation with events in its spatial neighborhood while the real activity events have. On the basis of this difference, the BA events can be filtered out

[1]Shasha Guo and Ziyang Kang contribute to this article equally.

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

by detecting events which do not have spatial correlation with events generated by the neighborhood pixels. Such a filter is a spatiotemporal correlation filter. The filter decides whether an event is a real event or noise by checking the condition for all neighborhood pixels, $T_{NNb} - T_e < dT$. T_{NNb} is the timestamp of the latest event occurred in one of the neighborhood pixels. T_e is the timestamp of the event. And dT is the limitation for timestamp difference. If the condition is meet, the event is regarded as a real activity event. As for neighborhood events, they meet this condition: $|x_p - x| \leq 1$ and $|y_p - y| \leq 1$, where x_p and y_p are x-position and y-position of the pixel p, and x and y are the positions of the pixel generating the current event.

C. E2LSH

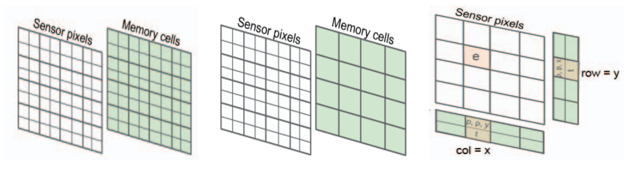
The main idea of Locality Sensitive Hashing (LSH) algorithm is to hash similar input items into the same "bucket" with high probability. This technique can be used for data clustering and NNb search. LSH was first applied in Hamming space, and then extended to Euclidean space. Exact Euclidean Locality Sensitive Hashing (E2LSH) is a scheme of LSH realized in Euclidean space [17]. Its basic principle is to reduce the dimension of high-dimension data via position sensitive functions based on p-stable distribution, so that two points close to each other in the original space are still very close after mapping. Hashing functions in E2LSH are formulated as $h(x) = \frac{ax+b}{w}$. a is a d-dimensions vector (the same dimension as data vector v), b is a random number between 0 and w, and w is used for segmenting lines. To reduce clustering errors, a set of k hashing functions are adopted. These k hash values are then processed by another two hashing functions to get two values for storage (building the hashtables) and search (indexing the hashtables).

D. BF

Bloom filter (BF) is a binary vector data structure proposed by Howard Bloom in 1970 [18]. It has good space and time efficiency and is used to detect whether an element is a member of a set. The typical method to find whether an element is in a set is saving all the elements, and comparing them with the element. Data structures such as link lists and trees are based on this idea. When the number of elements in the set increases, more space and time is needed, and the query speed becomes slower.

For checking an element, BF uses k hashing functions to encode the element and get k real values. Then, it uses the k values as indices to index an array with m cells. Each cell is 1 bit. Only all the k cells indexed by the indices are ones, the element is regarded as in the set.

There are still some problems with the above two methods when applying them to the filtering problem straightly. For E2LSH, it is computing intensive which has one mapping function of k hash operations and two more hashing functions. Furthermore, hashtables in E2LSH are memory intensive as the indices of each event are still stored explicitly. As for BF, the hashing functions is designed for set searching and doesn't maintain the spatiotemporal correlation well.



(a) Bs1 (b) Bs2 (c) Bs3

Fig. 1. Three event-based filters [14].

III. Related work

A. Event-based filters

Here we introduce three event-based filters with O(N^2), O(N/s) and O(N) space complexity respectively, and they will be denoted as Bs1, Bs2, and Bs3 in the rest content.

In Bs1 filter, each pixel has a memory cell for storing the last event's timestamp. The stored timestamps are used for computing the spatiotemporal correlation (Fig.1(a)).

Bs2 filter uses sub-sampling groups to reduce the memory occupation [6]. Each group with factor s contains s^2 pixels and uses one memory cell for storing the timestamp of the most recent event of the group (Fig.1(b)).

Bs3 filter assigns two memory cells to each row and each column to store the most recent event in that row or column (Fig.1(c)) [14]. This filter is designed to store all the information of an event, with two 32-bits memory cells. One of them is for storing the timestamp. The other is for polarity and the x-axis or y-axis position.

B. Frame-based heatmap filter

This frame-based filter processes a picture of fixed number of events at a time and output a denoised frame rather than checking each event. The heatmap is a matrix, of which the size equals to the size of the pixels of the DVS output. When an event occurs at a pixel, the corresponding element in the heatmap will add 1 to itself. When all events in a picture frame are processed, a threshold is calculated based on the heatmap by equation 1 [15].

$$Thr = \frac{\sum_{i \in S} h_i}{Card\,|S|} \tag{1}$$

Where S is the set of all the non-zero elements in the heatmap and $Card$ means the number of elements in a set. If one element in the heatmap exceeds the threshold, it means this pixels have generated enough events. And these events are regarded as real activity events.

There are also some other filtering methods. Vandana et.al. [21] proposed a filter with neuromorphic integrate-and-fire neurons which integrate spikes not only from the corresponding pixel but also its neighborhood pixels for firing. And [19] assigns a lifetime to each event and the lifetime of a noise event will be assigned 0.

IV. Proposed Filter

We propose a hashing-based spatiotemporal filter, *HashHeat*. As shown in Fig.2(a), *HashHeat* comprises four main steps. **Hash** gets the AER representation of an event as input, and uses k hashing functions to process the event information, and outputs k hash values $(h_1, h_2, ..., h_k)$ to the next stage. The k hashing functions are locality sensitive hashing functions, so as to maintain the spatiotemporal correlation information. **Index** receives the k hash values, and uses them to index the m-array, and outputs k heat values, $(v_1, v_2, ..., v_k)$. The m-array records the information of previous events, and is the key for checking the correlation between the current event and previous events. The k heat values are then forwarded into the **Check** stage. In this stage, they are compared with the threshold to generate an indicating signal, *flag*. If there is one element smaller than the threshold, *flag* will be set to 0 showing that the event is a noise event. And last but not least, **Update** receives the *flag* and the k hash values as input, and adds a certain increment to the k positions based on the value of *flag*. This enables the m-array to incorporate the current event information into itself.

The m-array needs reset at a certain frequency, since the cells of the m-array can only count finite numbers and will overflow without reset. Therefore, we define a *processing window* with fixed number of events segmented from the event stream. At the begining of each *processing window*, the m-array is reset. The begining and the length of processing window can be adjusted manually.

Cells in the m-array can be set to be 1 bit or multiple bits. The 1-bit m-array is the most space-efficient configuration for *HashHeat* (Fig.2(b)), and is capable of filtering BA events in a relative simple scene like moving a laser pointer in front of the DVS. However, checking based on the 1-bit m-array is prone to be interfered in a noisy environment, as a large number of noise events will update many locations in the m-array from 0 to 1. These locations should not be 1 until there are real activity events indexing them and updating them.

Inspired by the frame-based heatmap filter, we propose that heat value is a more promising metric for evaluating the correlation between events compared with the binary signal 1 and 0 in a noisy environment. Therefore, we propose the X-bit m-array (Fig.2(c)) so as to record the heat value of the cell. The heat value is the weighted sum of the indexed times of a cell. The X-bit m-array is also named heat-array. Compared with the baseline filters, it is still memory efficient since X doesn't need to be large considering the *processing window*.

The general form of *HashHeat* (Fig.2(c)) requires changeable thresholds. With the progress of processing, elements of the m-array are increasing. If the threshold keeps stable at a low level, the events in later processing progress will be tagged as real activity events with a high probability, as most of the cells will have large heat values at that time. Also, if we change the threshold at the coming of each event, it is inefficient and unnecessary because one event will not change many cells of the m-array. We consider the average heat value under the worst condition as the $threshold$, which can be formulated as equation 2.

$$threshold = \frac{EN \times k \times 2}{m}. \tag{2}$$

The worst condition refers to the theoretical maximum sum of the m-array at a certain timing point. EN is the number of input events from the begining of the *processing window*, k is the number of hashing functions and m is the length of the m-array. Due to the locality sensitive features of the hashing functions used in *HashHeat*, real activity events with spatiotemporal correlation will result in similar hash values. Thus the heat values in the m-array are more likely to be unevenly distributed. Therefore, this threshold is able to distinguish the cells with high heat values from those with low heat values.

Using hashing functions, *HashHeat* encodes the spatiotemporal information and thus is free from storing timestamps and position information. Using the m-array, *HashHeat* further compresses the memory occupation for checking the correlation between events.

V. Hardware Design

Based on the working scheme (Fig.2(a)), we propose the hardware design of *HashHeat* filter (Fig.3). The input to *HashHeat* include the x and y positions, and timestamp of an event. The output of *HashHeat* is a flag indicating whether this event is a BA noise event or not.

To speed up the processing of event streams, we design the pipelined architecture. In Stage 1, the timestamp T in each packet minus the baseline timestamp, which is the timestamp of the first event in the frame, generating a new timestamp $T2$. This offset operation is to prevent the timestamp from being too large to process. The x-position and y-position are not changed in this stage. In Stage 2, the pre-processed vector and the corresponding parameters of the hash function perform a dot operation. Next, the results will be modded in Stage 3 for limiting the hash values to a certain range. In this design, the length of the m-array is designed to be 128. In Stage 4, all the values obtained by indexing the m-array will be compared with the threshold. Multiple comparison operations are performed in parallel. Then we accumulate all the comparison results. If the sum is less than k, it indicates that an event is a noise event and the $Noiseflag$ will be 1. Finally, the heat-array will be updated differently according to the value of the flag.

The process can be formulated as equation 3. T_{base} is the timestamp of the first event in a *processing window*. Other parameters have been introduced before.

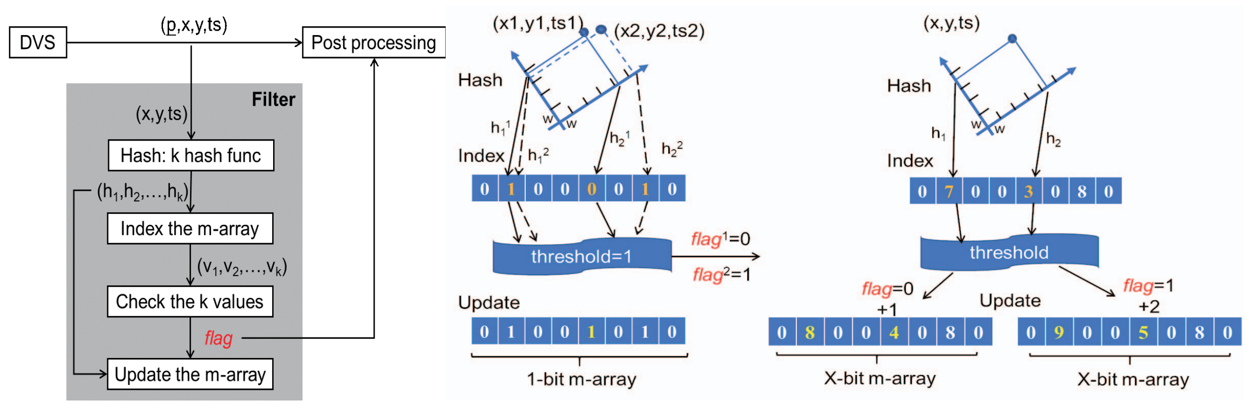
$$\begin{gathered} T2 = T - T_{base}, \\ h_j^{'} = a_1 x + a_2 y + a_3 T2, \\ h_j = \left(h_j^{'} \; mod \; w\right) \; mod \; m, \\ flag = \exists \left(marray\left(v_j\right) < Thr_i\right), \\ marray(h_j) = marray(h_j) + increment. \end{gathered} \tag{3}$$

VI. Experiment

A. Hardware

We use FPGA Artix-7 and Vivado for synthesizing and measuring performance such as power, hardware resources consumption, and timing through cycle-accurate simulations. We have implemented a simulator in MATLAB to verify our ideas and to use it as a reference model for our hardware implementation. The hardware behaves the same as the simulator.

HashHeat works at 100MHz. The computing latency is amortized by the pipeline which enables *HashHeat* to process an event at a latency of 10ns. Table I gives the utilization of resources of the hardware. DSP is used in Stage 2. As



(a) Working scheme of *HashHeat* filter (b) Special case of *HashHeat*. (c) General form of *HashHeat*.

Fig. 2. *HashHeat* filter. Fig.2(b) shows a special case of Fig.2(c). In Fig.2(b), it is only 1-bit per cell in the m-array, which saves the most space. The updating rule is simple. If a cell is 0 at the begining and it is indexed, it will turn into 1 and will not change to 0 again by later events within the processing window. And it doesn't depend on the flag to update. In Fig.2(c), the m-array is X-bits per cell (X is bigger than 1). And a cell will add a certain delta to itself when it is indexed. The delta is relative to the flag.

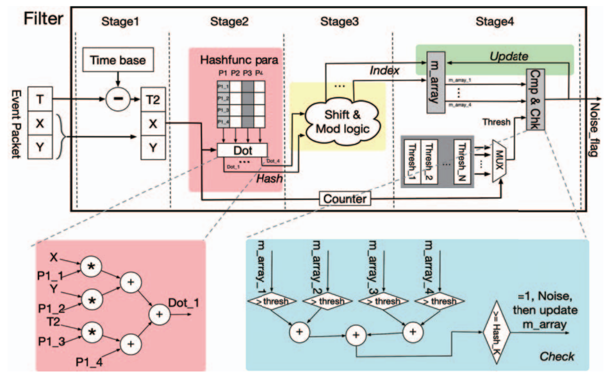


Fig. 3. Hardware design of *HashHeat*. $Thresh_i$ stands for threshold. The subscript i is larger than 1 when there are more than one threshold for large *processing window*. And a certain $Thresh_i$ will be chosen as threshold for comparison every time.

TABLE I
UTILIZATION OF HARDWARE RESOURCES.

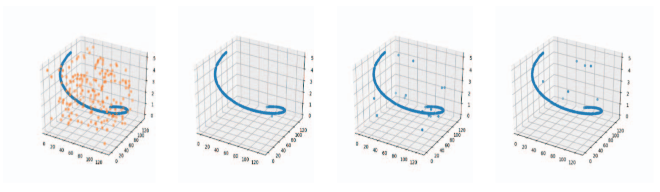
Resource	Utilization	Available	Utilization %
LUT	10400	20800	50.00
FF	2543	41600	6.11
DSP	20	90	22.22
IO	71	106	66.98
BUFG	1	32	3.13

HashHeat is computing-intensive, we use registers rather than BRAM to implement the heat-array, and thus there is no BRAM entry in the table. The power consumption is 0.471W.

The choice of the number of memory cells determines the space complexity C, and is affected by parameters such as k and X. We use 128 in this paper. The memory overhead is shown in Table II. The baseline filters are decribed in section III. It is clear that *HashHeat* has the least storage cost due to the elimination of storing timestamps and scaling with the DVS output size. We don't compare other logic overhead as the baseline filters also need computing logic.

B. Visual Effect

We use two datasets for validation, a synthetic and a real. The synthetic dataset is to mimic a laser pointing from bottom to top. The real dataset is a collected DVS dataset, DvsGesture [16], comprising 11 hand gesture categories from 29 subjects under 3 illumination conditions.



(a) Init (b) Bs1 (c) Bs3 (d) HH1bit

Fig. 4. Comparison of filters on synthetic dataset. HH1bit denotes *HashHeat* with the 1-bit m-array.

The performance of different filters on the synthetic dataset are shown in Fig.4. The x-axis and y-axis indicate the position of an event produced, and the verticle axis models the timestamp of an event. The blue points stand for real activity events and the orange points represent BA events. It is clear that Bs3 filter passed more errors than Bs1 and Bs2. In Bs3, pixels in one row share the same two memory cells and so are the elements in one col. Thus some noise events are likely to get support from other events occurred in the adjacent rows and cols, as these events will update the adjacent memory cells.

Fig.5 gives the performance of filters on the real dataset, DvsGesture. To make the event stream visible as a picture rather than the event flow, it is common to generate a frame from the events, either of fixed time length or of a constant number of events. We choose to use the fixed number of events (5,000 or 10,000 per frame). A pixel in the picture will be 255 if it has an event. Without loss of generality, we choose five gestures under three illumination conditions by three experimenters from DvsGesture for the convenience of illustration. It's worth noting that frames processed by *HashHeat* are cleaner than others.

C. Error Analysis

Here we introduce three metrics for evaluating error. Before introducing the metrics, it is necessary to introduce a concept in the field of object tracking, the bounding box. A bounding

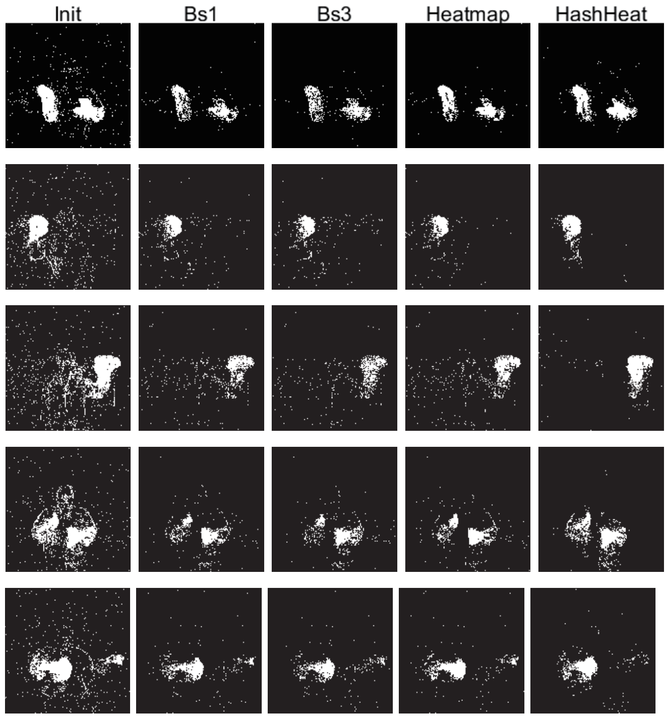


Fig. 5. Illustration of performance of filters on five gestures under three illumination condictions by three users (5,000 events per frame). The five gestures are hand clap, left/right hand wave, air drum and air guitar per row from the top down. The performance of Bs2 filter is similar to Bs1 and thus is not listed.

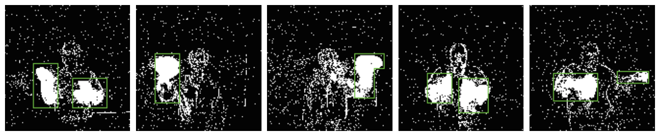


Fig. 6. Bounding boxes in five gestures (10,000 events per frame).

box is to mark the range of an object in the picture frame. Events in the bounding box are considered to be signal events, and events out of the bounding box are regarded to be noise events.

In this work, we choose bounding boxes manually as we focus on the noise filtering rather than object tracking where the choice of bounding boxes are the key element. Fig.6 shows the initial frames with bounding boxes.

We first introduce the notation E_p^q, which is the total number of p events (replacing p with 'signal' or 'noise') before filtering ($q = o$) or after filtering ($q = f$). We use three metrics, SR, NR and SNR inspired by [21], which can be formulated by equation 4. SR is the ratio of signal events remaining after filtering, NR is the ratio of noise events remaining after filtering, and SNR^q is the ratio of signal to noise events before filtering or after filtering.

$$SR = \frac{E^f_{signal}}{E^o_{signal}}, NR = \frac{E^f_{noise}}{E^o_{noise}}, SNR^q = \frac{E^q_{signal}}{E^q_{noise}} \quad (4)$$

For the three metrics introduced above, a good filter is expected to have a high SR and a low NR, which can lead to a high SNR^f.

For Heatmap filter, as we discussed in section III-B, it cannot give the check of event one by one, so it is a little tricky to compute the three metrics. The E^f_{signal}, E^o_{signal}, E^f_{noise} and E^o_{noise} are derived from the heatmap and weightmap.

TABLE II
STORAGE OVERHEAD.

Filter	Bs1	Bs2	Bs3	Heatmap	HashHeat
Bytes	65536	16384	2048	32768	256

Fig.7(a) and Fig.7(b) shows the SR and NR of the five filters for the five gesture frames. Filter Heatmap has the best SR while having the highest NR as well. This is because some pixels out of bounding boxes are also active and leads to frequent indexing of the corresponding cells. The heat values of these cells will then exceed the threshold and the events will be tagged as real events. In addition, Bs1 generally has the highest SR and SNR^f among the Bs-series filters which is reasonable as it requires the largest memory.

As shown in Fig.7(c) and Fig.7(d), *HashHeat* has the best SNR^f among all the filters for both the 5,000-events frames and 10,000-events frames. The maximum SNR^f improvement of *HashHeat* is up to 15x in Gesture2 for frames with 5000 events. And the minimum increment is nearly 1.2x in Gesture4 for frames with 5000 events.

In short, *HashHeat* has a better performance than the baseline filters w.r.t memory cost or signal to noise ratio.

D. Discussion

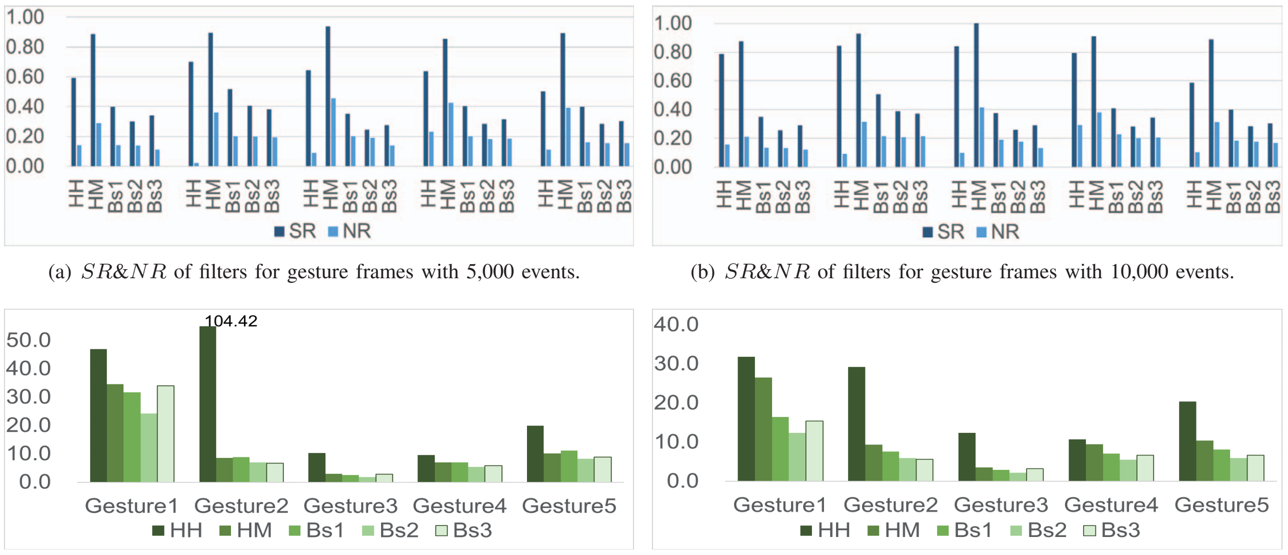
1) Advantage: Other than the heatmap filter processing frame by frame, *HashHeat* processes events streamly and in real-time. The heatmap filter can provide a visible frame with the effect of noise removal but not give the detection result of a single event. If a new frame with a different number of events is required, the filter must process the whole events again, which is inefficient and unreasonable. However, *HashHeat* detects every single event and generates a flag. By recording all flags which are only one-bit each, *HashHeat* is able to generate any frame at any required number of events and each event is processed only once.

2) The latency: DVS produces 1M events per second [2], and the time interval between two events is usually several μs. Thus *HashHeat* with a latency of 10 ns for an event can process the event stream in real time. Also, it can be integrated into other pre-processing methods for its fast speed.

For event-based sensors with events intervals less than 10ns, the current processing speed of *HashHeat* cannot meet the requirement. In this case, we need a FIFO for buffering events to prevent the loss of events. This work will be done in the future.

VII. Summary and Conclusions

Neuromorphic event-based sensors have witnessed rapid development in the past few decades, especially dynamic vision sensors. These sensors allow for much faster sampling rates and a higher dynamic range which outperform frame-based imagers. However, they are sensitive to background activity events which cost unnecessary communication and computing resources. Moreover, improved noise filtering will enhance performance in many applications. We propose *Hash-Heat*, an effective and memory efficient spatiotemporal BA



(a) $SR\&NR$ of filters for gesture frames with 5,000 events.

(b) $SR\&NR$ of filters for gesture frames with 10,000 events.

(c) SNR^f of filters for gesture frames with 5,000 events.

(d) SNR^f of filters for gesture frames with 10,000 events.

Fig. 7. Comparison of SR, NR and SNR^f. HH denotes HashHeat. HM denotes Heatmap.

filter with O(C) complexity. As far as we know, it is the first spatiotemporal filter that doesn't depend on the DVS output size N and doesn't require the storage of the 32-bits timestamp. The experimental results show that *HashHeat* increases the signal to noise ratio by about 1.2x to nearly 15x on DvsGesture compared with other baseline filters.

Acknowledgments

We thank Professor Tobi Delbruck for the discussion. We thank reviewers and sheperds for their precious comments. This work is supported by National Key R&D Program of China (2018YFB2202603), HGJ (2018ZX01029-103) and the National Natural Science Foundation of China (61802427 and 61832018).

References

[1] M. Mahowald, and C. Mead. "The Silicon Retina." Scientific American 264.5(1991):76.

[2] P. Lichesteiner, C. Posch, and T. Delbruck. "A 128×128 120 dB 15μsec Latency Asynchronous Temporal Contrast Vision Sensro". *IEEE Jornal of Solid-State Circuits* 43.2 (2008), pp. 566–576.

[3] C. Posch, D. Matolin, and R. Wohlgenannt (2011). "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS". *IEEE J. Solid State Circ.* 46, 259–275.

[4] J. A. Lenero-Bardallo , T. Serrano-Gotarredona , and B. Linares-Barranco . "A 3.6 s Latency Asynchronous Frame-Free Event-Driven Dynamic-Vision-Sensor." *IEEE Journal of Solid-State Circuits* 46.6(2011):1443-1455.

[5] R. Berner, C. Brandli, M. Yang, S.-C. Liu, and T. Delbruck . "A 240×180 10mW 12μs latency sparse-output vision sensor for mobile applications," *VLSI Circuits (VLSIC)*, 2013 Symposium (Kyoto), 186–187.

[6] H. Liu, C. Brandli, C. Li, S. C. Liu, and T. Delbruck . "Design of a spatiotemporal correlation filter for event-based sensors," in *Proceedings of IEEE International Symposium on Circuits and Systems (Lisbon).* 2015.

[7] Hui Tian. "Noise Analysis in CMOS Image Sensors". PhD thesis. Stanford University, 2000.

[8] D. Reverter Valeiras, G. Orchard, S.-H. Ieng, R. B. Benosman, "Neuromorphic Event-Based 3D Pose Estimation." *Front. Neurosci.* 9 (2015) 522. doi:10.3389/fnins.2015.00522.

[9] J. Lee, T. Delbruck, P. K. J. Park, M. Pfeiffer, C. W. Shin, H. Ryu, et al. "Live demonstration: Gesture-Based remote control using stereo pair of dynamic vision sensors." *IEEE International Symposium on Circuits & Systems* 2012.

[10] M. Liu, W. Kao, T. Delbruck. "Live Demonstration: A Real-Time Event-Based Fast Corner Detection Demo Based on FPGA." *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops,* 2019.

[11] S. H. Ieng, C. Posch, and R. Benosman . "Asynchronous neuromorphic event-driven image filtering." *Proc. IEEE* 102, 1485–1499. 2014.

[12] A. Linares-Barranco, F. Gomez-Rodriguez, V. Villanueva, L. Longinotti, and T. Delbruck (2015). "A USB3.0 FPGA event-based filtering and tracking framework for dynamic vision sensors," *IEEE International Symposium on Circuits and Systems (Lisbon).* doi: 10.1109/ISCAS.2015.7169172

[13] D. Czech, and G. Orchard. "Evaluating noise filtering for event-based asynchronous change detection image sensors," *Proceedings of the IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics.* 2016.

[14] A. Khodamoradi, and R. Kastner . "O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors." *IEEE Transactions on Emerging Topics in Computing PP*, 1–1. 2018.

[15] https://github.com/yucicheung/CelexMatlabToolbox

[16] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017.

[17] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions." *Twentieth Symposium on Computational Geometry* (pp.253).

[18] B.H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM*, 13 (7): 422–426, 1970.

[19] E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza . "Lifetime estimation of events from Dynamic Vision Sensors." *IEEE International Conference on Robotics and Automation (ICRA)*, 4874–4881, 2015.

[20] A. Mortara and E. A. Vittoz, "A communication architecture tailored for analog VLSI artificial neural networks: intrinsic performance and limitations," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 459-466, 1994

[21] P. Vandana , B. Arindam , and O. Garrick . "A noise filtering algorithm for event-Based asynchronous change detection image sensors on TrueNorth and its implementation on TrueNorth." *Frontiers in Neuroscience* 12(2018):118-.