Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society, Univ. of Tokyo, Mar. 6-7, 2008, pp. 21-26.

# Frame-free dynamic digital vision

Tobi Delbruck

Institute of Neuroinformatics, University and ETH Zurich, Winterthurerstr. 190, CH-8057 Zurich, Switzerland
tobi@ini.phys.ethz.ch

***ABSTRACT* — Conventional image sensors produce massive amounts of redundant data and are limited in temporal resolution by the frame rate. This paper reviews our recent breakthrough in the development of a high-performance spike-event based dynamic vision sensor (DVS) that discards the frame concept entirely, and then describes novel digital methods for efficient low-level filtering and feature extraction and high-level object tracking that are based on the DVS spike events. These methods filter events, label them, or use them for object tracking. Filtering reduces the number of events but improves the ratio of informative events. Labeling attaches additional interpretation to the events, e.g. orientation or local optical flow. Tracking uses the events to track moving objects. Processing occurs on an event-by-event basis and uses the event time and identity as the basis for computation. A common memory object for filtering and labeling is a spatial map of most recent past event times. Processing methods typically use these past event times together with the present event in integer branching logic to filter, label, or synthesize new events. These methods are straightforwardly computed on serial digital hardware, resulting in a new event- and timing-based approach for visual computation that efficiently integrates a neural style of computation with digital hardware. All code is open-sourced in the jAER project (jaer.wiki.sourceforge.net).**

***Keywords* — Neuromorphic, AER, address-event, vision sensor, spike, surveillance, tracking, feature extraction, low-latency vision**

## I. INTRODUCTION

Conventional image processing methods rely on operating on the entire image in each frame, touching each pixel many times and leading to a high cost of computation and memory communication bandwidth, especially for high frame-rate applications. For example, a brute force computation of a set of wavelet transforms can cost thousands of machine instructions in floating point precision for each pixel of the image. Methods such as image pyramids [1] or integral image transforms [2] can reduce this computational cost but still require at least one pass over all pixels in each frame. In addition, the limited frame rate limits response latency and temporal resolution and greatly complicates tracking of fast moving objects.

We recently achieved a breakthrough in developing a Dynamic Vision Sensor (DVS) [3, 4] with unprecedented raw performance characteristics and usability. The DVS output consists of asynchronous address-events that signal scene reflectance changes at the times they occur (Fig. 1). This sensor loosely models that transient pathway in biological retinas. The output of the sensor is in the form of asynchronous digital spike address-events of pixels encoded on a shared digital bus. [5-7].

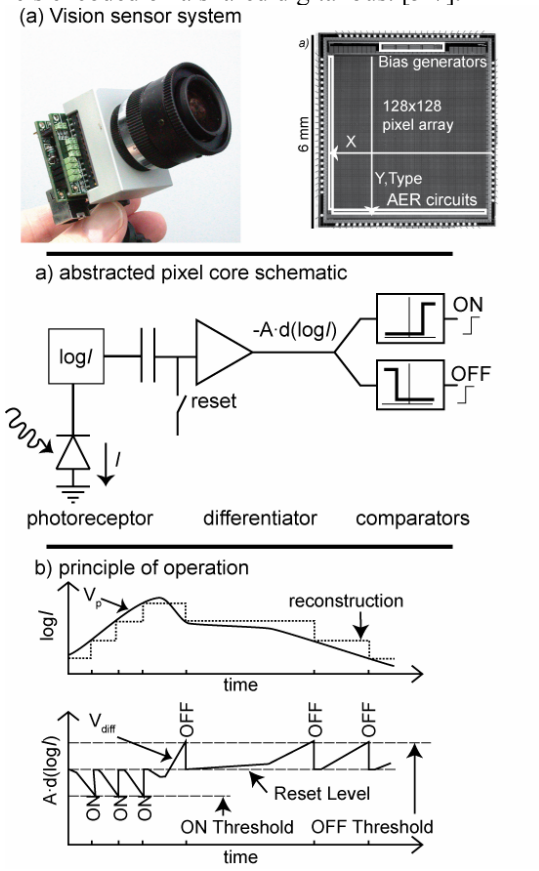


Fig. 1 DVS characteristics. a) the dynamic vision sensor with lens and USB2.0 interface; b) a die photograph labeled with components. Also shown is the row and column from a pixel that generates an event; c) abstracted schematic of the pixel which responds with events to fixed-size changes of log intensity; d) how the ON and OFF events are internally represented and output in response to an input signal.

The DVS was conceived in the CAVIAR project [8], where it provides the input to a chain of hybrid analog-digital address-event chips. The main achievement of this project was the realization of a real time spike-based system for visual processing consisting of series of feed forward processing components that model early visual processing, object classification and tracking. In the desire to build a system entirely based on neural-like architectures, the flexibility of procedural computation was lost and it became very difficult to configure the system to do anything other that what it was originally conceived to do.

This concern has led to a series of ongoing investigations of how the retina events can be digitally processed by algorithms running on standard hardware

and these algorithms are the main topic of this review. The main characteristics of these methods are 1) they are event-driven, which means they operate just on the pixels or areas of the image that need processing, 2) they are digital and are efficiently processed on synchronous digital hardware, 3) they extensively use the precise timing of the events. This combination of characteristics leads to a new approach for visual processing that integrates a biological style of processing with digital hardware. To encourage community development, all code is open-sourced in the jAER project [9].

## II. Dynamic Vision Sensor

The DVS improves on prior frame-based temporal difference detection imagers (e.g. [10]) by asynchronously responding to temporal contrast rather than absolute illumination, and on prior event-based imagers because they either do not reduce redundancy at all [11], reduce only spatial redundancy [12], have large fixed-pattern-noise (FPN), slow response, and limited dynamic range [13], or have low contrast sensitivity . The DVS is particularly suitable for tracking moving objects and has been used for various applications: high speed robotic target tracking [14], traffic data acquisition [15, 16], and in internal work for tracking particle motion in fluid-dynamics, tracking the wings of fruit-flies, eye-tracking, and rat paw tracking for spinal cord rehabilitation research.

The main properties of the DVS are summarized in Fig. 1 and Table I. Each address-event signifies a change in log intensity

$$\left|\Delta \log I\right| > T \quad (1)$$

where $I$ is the pixel illumination and $T$ is a global threshold. Each event thus means that log$I$ changed by $T$ since the last event and specifies in addition the sign of the change. For example, if T=0.1 then each event signifies approximately 10% change in intensity. This "relative" property encodes scene reflectance change. Because this computation is based on a very compressive logarithmic transformation in each pixel, it also allows for wide dynamic range operation (120 dB or 6 decades, compared with e.g. 60 dB for a high quality traditional image sensor). This wide dynamic range means that the sensor can be used with uncontrolled natural lighting that is typified by wide variations in scene illumination. The asynchronous response property also means that the events have a very short latency and the timing precision of the pixel response rather than being quantized to the traditional frame rate. Thus the "effective frame rate" is typically several kHz. If the scene is not very busy, then the data rate can easily be a factor of 100 lower than from a frame-based image sensor of equivalent time resolution. The design of the pixel also allows for unprecedented uniformity of response. The mismatch between pixel contrast thresholds is 2.1% contrast, so that the pixel event threshold can be set to a few percent contrast, allowing the device to sense real-world contrast signals rather than only artificial high contrast stimuli. The vision sensor also has integrated digitally-controlled biases that greatly reduce chip-to-chip variation in parameters and temperature sensitivity. And finally, the system we built has a standard USB2.0 interface that delivers time-stamped address-events to a host PC. This combination of features has meant that we have had the possibility of developing algorithms for using the sensor output and testing them easily in a wide range of real-world scenarios.

TABLE I TMPDIFF128 DYNAMIC VISION SENSOR SPECIFICATIONS

| | |
|---|---|
| *Functionality* | Asynchronous temporal contrast |
| *Pixel size um (lambda)*<br>*Fill factor (%)* | 40x40 (200x200)<br>8.1%<br>(PD area 151μm$^2$) |
| *Fabrication process* | 4M 2P 0.35um |
| *Pixel complexity* | 26 transistors (14 analog), 3 capacitors |
| *Array size* | 128x128 |
| *Die size mm$^2$* | 6x6.3 |
| *Interface* | 15-bit word-parallel AER |
| *Power consumption* | 24mW @ 3.3V |
| *Dynamic range* | >120dB<br><0.1 lux to > 100 klux scene illumination with f/1.2 lens |
| *Photodiode dark current, 25 C* | 4fA (~10nA/cm$^2$)<br>Nwell photodiode |
| *Response latency*<br>*Events/sec* | 15μs @ 700mW/m$^2$<br>~1M events/sec |
| *Event threshold matching (1 sigma)* | 2.1% contrast |

## III. Event processing

Binning the DVS events into traditional frames immediately quantizes the time to the frame time and requires processing the entire frame.

Instead, in the event-driven style of computation, each event's location and timestamp are used in the order of arrival, inspired from the data-driven information processing occurring in brains. These algorithms also take advantage of the capabilities of synchronous digital processors for high speed iteration and branching logic operations.

The characteristics of these methods will be demonstrated by a number of examples. These methods have evolved naturally into the following classes:

- *filters* that clean up the input to reduce noise or redundancy,
- *labelers* that assign additional meaning besides ON or OFF—additional type information—to the events such as contour orientation or direction of motion. Based on these extended types, we can very cheaply compute global metrics such as image velocity.
- *trackers* that use events to track moving objects..

The filters and labelers also generally use one or several topographic memory maps of event times. These maps store the last event timestamps for each address.

The digital representation of these events allows attachment of arbitrary annotation. The events start with

precise timing and spatial location in the retina and with an ON or OFF type. As they are processed, extraneous events are discarded, and as they are labeled they can gain additional meaning. We attach this meaning to the event by means of an extended type that is analogous but not the same as cell type in cortex. Instead of expanding the representation by expanding the number of cells (as for the usual view of cortical processing), we instead assign increasing interpretation to the digital events. We can still carry along multiple interpretations, but these interpretations are carried by multiple events instead of activity on multiple hardware units. For instance, a representation of orientation that is halfway between two principle directions can still be represented as near-simultaneous events, each one signifying a different and nearby orientation. In addition, this extended event type information is not limited to binary existence. A motion event can carry along information about the speed and vector direction of the motion.

The organization of these events in memory is also important for efficiency of processing and flexibility of software development. The architecture we evolved over three generations of software refactoring is illustrated in Fig. 2. Events are bundled in packets. A packet is a reused memory object that contains a list of event objects. These event objects are references (in the Java sense) to structures that contain the extended type information. A particular filter or processor maintains its own reused output packet that holds the results. These packets are reused because the cost of object creation is much higher (typically a factor of 100) than the cost of object access. The packets are dynamically grown as necessary, although this expensive process only occurs a few times during program initialization. Dynamic memory (stack) usage is not very high because the reused packets are rarely allocated and need not be garbage-collected.

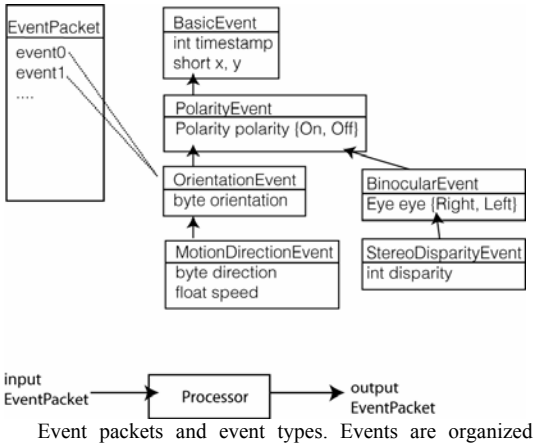


Fig. 2 Event packets and event types. Events are organized in packets that contain references (pointers) to event objects. These event objects are subclasses of a basic type. Each subclass elaborates the event type of the superclass that elaborate the event. These event packets are processed by event processors, outputting packets of the same type (filter) or new types (labeler). Some event processors do nothing to transform the input packet but compute metrics or object properties from the packet, e.g. global motion, tracked object lists.

Generally, the number of events is reduced by each stage of processing, so later stages need do less work and can also do more expensive computations.

In the jAER implementation, and memory buffer is used between the vision sensor and the processing and the processing occurs in buffer-sized packets. The latency can be as long as the time between the last events in successive packets plus the processing time. These packets are analogous to frames, but are not the same thing. A packet can represent a variable amount of real time depending on the events in the packet. Packets will tend to carry more identical amounts of useful information than frames. Our hardware interface (USB) between the vision sensor and a host PC is built to ensure that these packets get delivered to the host with a minimum frequency, typically 100 Hz. Then the maximum packet latency is 10 ms. But the latency can be much smaller if the event rate is higher. For example, the USB chip that we use has hardware buffers of 128 events. If the event rate is 1 MHz, then 128 events fill the FIFO in 128 us and thus the latency due to the device interface is about 200 times shorter than the 30 ms per frame from a 30 Hz camera.

*Software infrastructure*

The jAER project is implemented in Java and presently consists of about 300 classes. jAER allows for flexibly capturing events from multiple hardware sources, rendering events to the screen (as viewable frames or other representation, e.g. space-time), and recording and playing them back. The event-processing algorithms described here can be enabled as desired by an automatically-generated software GUI interface that also allows control of method parameters and handles persistence. All methods can run in real time at <30% load on live retina events on a standard 2005 laptop computer (Pentium M, 2 GHz). Quantitative performance metrics are shown later.

## IV. EVENT FILTERING

Filtering of the event stream transforms events or discards events that can arise from background activity or redundant sources. We will describe 3 examples of these filters.

*Background activity filter*

This filter removes uncorrelated background activity (caused on the device by transistor switch leakage or noise). It only passes activity that is supported by recent nearby past activity Background activity is uncorrelated and is largely filtered away, while events that are generated by moving objects, even if they are only single pixel in size, mostly pass through. This filter uses a single map of event timestamps to store its state, i.e., an array of 128x128x2 32 bit integer timestamp values. (131kB). This filter has a single parameter $T$ which specifies the support time for which an event will be passed. The steps

for each event are as follows:

1. Store the event's timestamp in all 8 neighboring pixel's timestamp memory, overwriting the previous values.
2. Check if the present timestamp is within $T$ of the previous value written to the timestamp map at this event's location. If a previous event has occurred recently, pass the event to the output, otherwise discard it.

(This implementation avoids iteration and branching over all neighboring pixels by simply storing an event's timestamp in all neighbors. Then only a single conditional branch is necessary.)

Typical snapshot results of the background activity filter are shown in Fig. 3. This filter is very effective at removing background activity; using typical DVS biasing the background rate is reduced from 3 kHz to about 50 Hz, a factor of 60, while the rate of activity caused by a moving stimulus is unnoticeably affected.

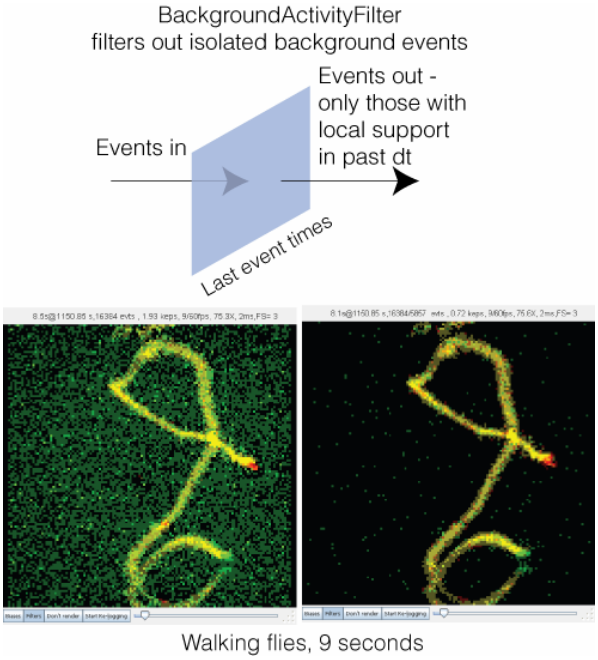


Fig. 3 Example of event-filtering. BackgroundActivityFilter filters out about 2/3 of the events that lack spatio temporal support, leaving only the walking fruit fly.

## V. LOW LEVEL VISUAL FEATURE EXTRACTION

Low level feature extraction labelers take the event stream and assign additional interpretation to the events, e.g., the edge orientation or the direction and speed of motion of an edge.

*Orientation labeler*

A moving edge will tend to produce events that are correlated more closely in time with nearby events from the same edge. The orientation labeler (Fig. 4) takes ON and OFF events from the vision sensor and labels them with an additional 'orientation type' that signals their angle of maximum correlation with past events in the nearby vicinity. The orientation type can take 4 values corresponding to 4 orientations separated by 45 degrees. This labeler uses a topographic memory of past event times like the background activity filter. There is a separate map for each retina polarity so that ON events can be correlated with ON and OFF with OFF. The orientation labeler parameters are the length of the receptive field in pixels and the minimum allowed correlation time. For each each orientation, past event times are compared with the present event time along the direction of orientation to compute the degree of correlation of the present event with past events. Events that pass the correlation test are output. The correlation measure can be chosen to be either the maximum time difference or the average time difference. Smaller time differences indicate better correlation. An option allows either outputting all orientations that pass the test or only the one that is best. The lookups (array offsets) into the memory of past event times are pre-computed when the labeler parameters are modified. The steps are as follows:

1. Store the event time in the map of times, pre-applying a subsampling bit shift if desired.
2. For each orientation, measure the correlation time in the area of the receptive field
3. Output an event for the best correlation if it passes the criterion test.

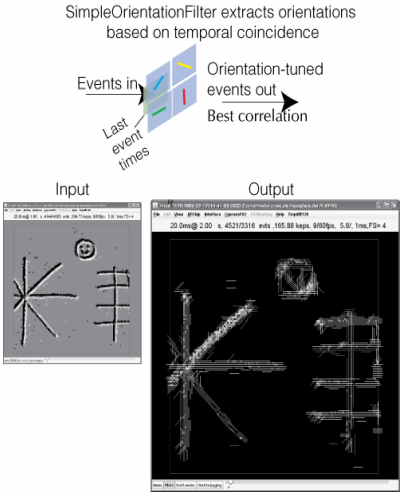


Fig. 4 Example of event labeler: SimpleOrientationFilter annotates events with the edge orientation. Each panel shows a different orientation type output.

## VI. TRACKING

The basic cluster tracker tracks multiple moving objects [14, 17]. It does this by using a model of an object as a spatially-connected rectangular source of events. As the objects move they generate events. These events are used to move the clusters. The key advantages of the cluster tracker are

1. There is no correspondence problem because there are no frames, so the events between rendered views still push along the clusters.

2. Only pixels that generate events need to be processed and the cost of this processing is dominated by the search for the nearest existing cluster, which is typically a cheap operation because there are few clusters.

The cluster has a size that is fixed but can be a function of location in the image. In some scenarios such as looking down from a highway overpass, the class of objects is rather small, consisting of cars, trucks and motorcycles, and these can all be clumped into a single size. This size in the image plane is a function of height in the image because the vehicles near the horizon are small and the ones passing under the bridge are maximum size. Additionally, the vehicles near the horizon are all about the same size because they are viewed head-on. In other scenarios, all the objects are nearly the same size. Such is the case of looking at particles in a hydrodynamic tank experiment or falling raindrops. In other scenarios, objects fall into a distinct and small set of classes, e.g. cars and pedestrians, but we have not developed a cluster tracker that can distinguish these classes.

The steps for the cluster tracker are outlined as follows. For each packet of events:

1. For each event, find the nearest existing cluster.
    1. If the event is within the cluster radius of the center of the cluster, add the event to the cluster by pushing the cluster a bit towards the even and updating the last event time of the cluster.
    2. If the event is not close to any cluster, seed a new cluster if there are spare unused clusters to allocate. A cluster is not marked as "visible" until it receives a certain number of events.
2. Iterate over all clusters, pruning out those clusters that have not received sufficient support. A cluster is pruned if it has not received an event for a "support" time.
3. Iterate over all clusters to merge clusters that belong to the same object. This merging operation is necessary because new clusters can be formed when an object increases in size or changes aspect ratio. This iteration continues until there are no more clusters to merge and proceeds as follows:

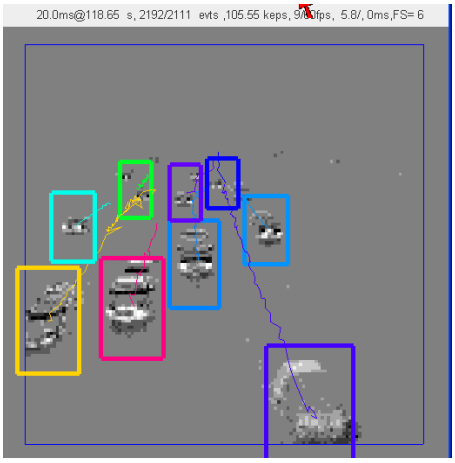


Fig. 5 Object tracking: RectangularClusterTracker tracks multiple cars from highway overpass.

The tracker has been used as part of a robotic goalie that achieves an effective frame rate of 550 FPS and a reaction latency of 3ms with a 4% processor load, using standard USB interfaces [14]. This combination of metrics would be impossible to achieve using conventional frame based vision.

## VII. Performance

The costs of digital event processing on a host PC platform are shown in Table II. The measurements were taken on a single core Pentium M laptop with 2.13GHz processor, 2GB RAM, Windows XP SP2, running at "Maximum Performance" settings (800 MHz clock), running the Java 1.6 virtual machine.

These measurements show that these algorithms running on a 2005 laptop processor consume from 100-1000 ns per event, so each event requires from a few hundred to a few thousand machine instructions. These timings constrain the real time capability. For example, if the event processing requires 1 us/event, then the hardware can process 1 million events per second. Since the maximum event output rate of the present sensor is about 1 Meps, a 2005 platform can process any input condition in real time. In fact, at rendering frame rates of 50 Hz, load on a contemporary laptop computer rarely exceeds 30% even when the most expensive processing is enabled.

TABLE II Performance.

| Algorithm | us/event (1024 event packets) |
|---|---|
| *BackgroundActivityFilter* | 0.1 |
| *SimpleOrientationLabler* | 0.7, RF is 5x1 pixels |
| *RectangularClusterTracker* | 0.5, 14 objects |

## VIII. Summary and Conclusion

The main achievement of this work is the development of novel event-based digital visual processing methods for low and high level vision. To our knowledge a general set of methods of utilizing event timing has not been previously described. These methods can be

efficiently realized on fixed-point embedded platforms. They capture the flavor of biological spike-based processing in synchronous digital hardware.

None of these methods were conceived before the vision sensor was built in a form that readily allowed its everyday use away from the lab bench. It was only after the device was realized with a convenient (USB) interface and a large software infrastructure was built to visualize the data from the sensor that we began to develop the methods described here for processing and using the events. Thus this development was stemmed directly from the availability of a highly usable form of a new class of vision sensor.

Although these methods have been developed as software algorithms running on a standard PC platform, it is clear that many of these algorithms can be implemented in embedded hardware. One can consider a range of event-processing platforms (Fig. 6). Using host PCs for processing reduces development time and initial cost. The majority of work with AER systems has focused on the opposite extreme; using AER neuromorphic chips to process the output from other AER chips. Our industrial partners are using an embedded DSP platform, and our partners in the CAVIAR project are starting to use FPGAs for some simple event-based processing. This work is very recent and has substantial room for innovation at many levels. It has the potential of realization of small, fast, low power embedded sensory-motor processing systems that are beyond the reach of traditional approaches under the constraints of power, memory, and processor cost.

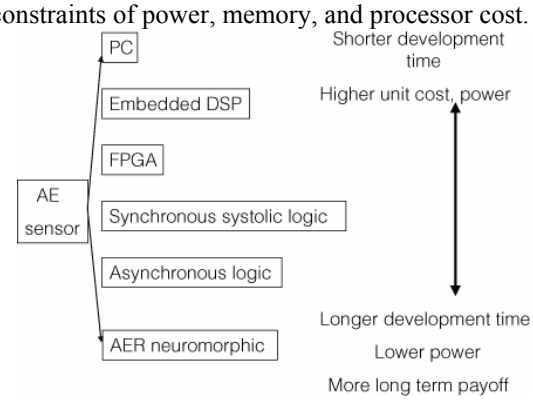


Fig. 6 Event processing hardware platforms. This paper described methods implemented mostly at the PC level, where development times are shortest.

## IX. Acknowledgements

The jAER project is on-going and has many contributors. Patrick Lichtsteiner has been a key contributor in building the DVS.

## X. References

[1] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 31(4), pp. 532-540, 1983.

[2] P. Viola and M. Jones, "Robust real time face detection," Eighth IEEE Conference on Computer Vision, 2001, pp. 747-747.

[3] P. Lichtsteiner, et al., "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," ISSCC Dig. of Tech. Papers, San Francisco, 2006, pp. 508-509 (27.9).

[4] P. Lichtsteiner, et al., "A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE J. Solid State Circuits*, vol. 43(2), pp. scheduled, 2008.

[5] J. Lazzaro, et al., "Silicon auditory processors as computer peripherals," *IEEE Trans.on Neural Networks*, vol. 4(pp. 523-528, 1993.

[6] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. Boston: Kluwer, 1994.

[7] K. A. Boahen, "A burst-mode word-serial address-event link-I transmitter design," *IEEE Transactions on Circuits and Systems I-Regular Papers*, vol. 51(7), pp. 1269-1280, 2004.

[8] R. Serrano-Gotarredona, et al., "AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems," Advances in Neural Information Processing Systems 18, Vancouver, 2005, pp. 1217-1224.

[9] T. Delbruck, "jAER open source project," 2007, Available:http://jaer.wiki.sourceforge.net.

[10] U. Mallik, et al., "Temporal change threshold detection imager," ISSCC Dig. of Tech. Papers, San Francisco, 2005, pp. 362-363.

[11] E. Culurciello and R. Etiene-Cummings, "Second generation of high dynamic range, arbitrated digital imager," 2004 International Symposium on Circuits and Systems (ISCAS 2004), Vancouver, Canada, 2004, pp. 828-831.

[12] P. F. Ruedi, et al., "A 128x128, pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction," *IEEE Journal of Solid-State Circuits*, vol. 38(12), pp. 2325-2333, 2003.

[13] K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip II: Testing and results," *IEEE Transactions on Biomedical Engineering*, vol. 51(4), pp. 667-675, 2004.

[14] T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," ISCAS 2007, New Orleans, 2007, pp. 845-848.

[15] A. Belbachir, et al., "Estimation of Vehicle Speed Based on Asynchronous Data from a Silicon Retina Optical Sensor," IEEE Intelligent Transportation Systems Conference ITSC 2006, Toronto, 2006, pp. 653-658.

[16] M. Litzenberger, et al., "Vehicle Counting with an Embedded Traffic Data System using an Optical Transient Sensor," Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE, 2007, pp. 36-40.

[17] M. Litzenberger, et al., "Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor," Digital Signal Processing Workshop, 12th-Signal Processing Education Workshop, 4th, 2006, pp. 173-178.