pyDVS: A Real-time Dynamic Vision Sensor Emulator using Off-the-Shelf Hardware

Garibaldi Pineda García*, Patrick Camilleri†, Qian Liu* and Steve Furber*

*School of Computer Science
University of Manchester
Manchester, United Kingdom
garibaldi.pinedagarcia@manchester.ac.uk

†School of Computer Science
University of Manchester
Manchester, United Kingdom
garibaldi.pinedagarcia@manchester.ac.uk

Abstract—**Vision is one of our most important senses, a vast amount of information is perceived through our eyes. Neuroscientists have performed many studies using vision as input to their experiments. Computational neuroscientists have typically used a brightness-to-rate encoding to use images as spike-based visual sources for it's natural mapping. Recently, neuromorphic Dynamic Vision Sensors (DVSs) were developed and, while they have excellent capabilities, they remain scarce and relatively expensive.**

We propose a visual input system inspired by the behaviour of a DVS but using a conventional digital camera as a sensor and a PC to encode the images. By using readily-available components, we believe most scientists would have access to a realistic spiking visual input source. While our primary goal is to provide systems with a real-time input, we have also been successful in transcoding well established image and video databases into spike train representations. Our main contribution is a DVS emulator extended by adding local inhibitory behaviour, adaptive thresholds and time-based encoding of the output spikes.

I. Introduction

In recent years the rate of increase of computer processors' performance has been slow; this is mainly because manufacturing technologies are reaching their physical limits. One way to improve performance is to use many processors in parallel, which has been successfully applied to parallel-friendly applications such as computer graphics. Meanwhile tasks such as pattern recognition remain difficult for computers, even with these technological advances.

Our brains are particularly good at learning and recognizing visual patterns (e.g. letters, dogs, houses, etc.). To achieve better performance for similar tasks on computers, scientists have looked to biology for inspiration. This has led to the rise of brain-like (neuromorphic) hardware, which mimics functional aspects of the nervous system. We can divide neuromorphic hardware into sensors (providing input), computing devices (which make use of information from sensors) and actuators (which control devices). Traditionally visual input has been obtained from images that are rate-encoded, that is every pixel represents a neuron emits a number of spikes proportional to its brightness, usually via a Poisson process [1]. While this might be a biologically-plausible encoding in the first phase of a "visual pipeline", it is unlikely that retinas transmit as many spikes into later stages. Furthermore, if we think in terms of digital networks, having each pixel represented by a Poisson process could incur high bandwidth requirements.

In 1989, Mead proposed a silicon retina consisting of individual photoreceptors and a resistor mesh which allowed nearby receptors to influence the output of a pixel [2]. Later, researchers developed frame-free Dynamic Vision Sensors (DVSs) [3, 4]. These feature independent pixels which emit a signal when their log-intensity values change above a certain threshold. Each of these signals could be interpreted as a neuron spike, thus These sensors have micro-second response time and excellent dynamic range properties, although they are still not as widely available as conventional cameras.

Katz et al. developed a DVS emulator in order to test behaviours for new sensor models [5]. In their work, they transform the image provided by a commercial camera into a spike stream [at 125 frames per second (FPS)]. In simple terms, the emulation is done by differencing video input with a reference frame; if this difference is larger than a threshold it produces an output and updates the reference. The number of spikes produced per pixel are proportional to the difference-to-threshold ratio. This emulator has been merged into their jAER project [6], a Java-based Address-Event Representation software framework that specializes in processing DVS output in real time.

In this work, we present a behavioural emulator of a DVS using a conventional digital camera as a sensor. Basing the emulator on widely available hardware allows computational neuroscientists to include video as a spike-encoded input. We present our basic emulator in Section II. In Section III we describe the spike encodings provided, Section IV discusses processing blocks added to the basic emulator. Results are given in Section V; conclusions and suggestions for future work are given in Sections VI and VII, respectively.

II. The emulator

Our emulator works by analysing the difference of the latest frame captured from the camera and a reference frame. If a pixel changes more than a certain threshold, then we generate an event which contains the pixel's coordinates and whether the change was positive or negative.

Pixels in real DVSs have a logarithmic response to light intensity, similarly most commercial cameras produce gamma-encoded images [7] for better bit utilization and, in the past, to be compliant with cathode ray tube (CRT) monitors. Since this encoding's response is similar to the logarithmic one used in a real DVS, we skip this step of the emulation.

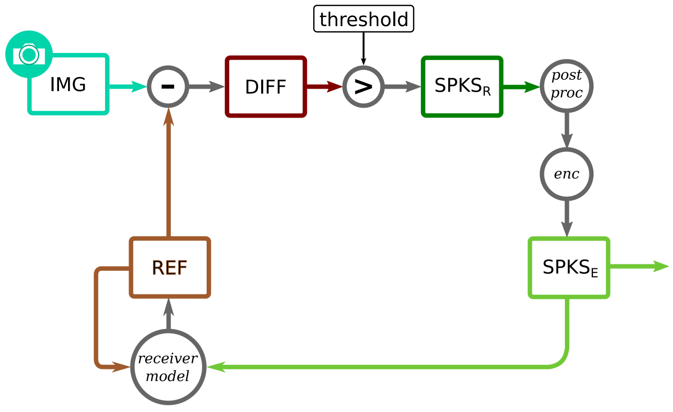


Fig. 1: DVS emulation diagram. Circles indicate operations and rectangles stages of visual information (from frames to spike trains)

Figure 1 shows the basic DVS emulation diagram, we obtain an image (IMG) from a video source and perform a difference with the reference frame (REF). We then apply a threshold filter to the difference frame (DIFF), the resulting pixels are considered *raw spikes* (SPKS_R). We can optionally post-process these pixels, as we'll demonstrate in Section IV, but we must encode them (Sec. III) so that they can be emitted as events (SPKS_E). Finally, depending on the selected type of output encoding, we simulate a receiver and update the reference frame accordingly.

III. Output Encoding

A. Rate-based

As in previous emulators [5], the standard output format is rate-based. To calculate the number of spikes that represents a change in brightness we use the following expression

$$N_H = \left\lfloor \min\left(T, \ \frac{\Delta B}{H}\right) \right\rfloor \tag{1}$$

where, in this encoding, N_H is the number of spikes needed to represent the change in brightness ΔB in terms of the threshold H. Notice that the maximum time (T) to transmit all the spikes for one frame is bound by the frame rate of the camera (fps). If the time resolution of spike events is one millisecond, then $T = 1000/fps$. At this stage we model a perfect receiver, so the update rule for the reference is

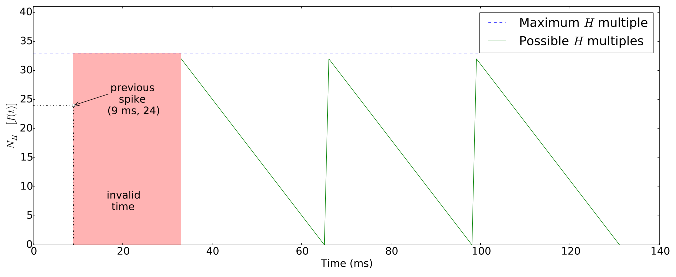
$$R_{now} = R_{last} + N_H \cdot H \tag{2}$$

B. Time-based

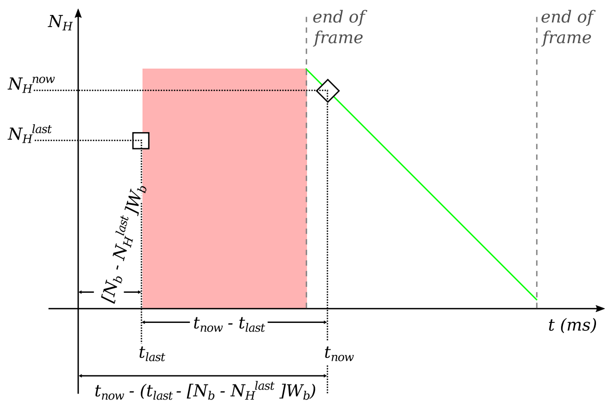
In its worst case rate-based encoding can send a spike per millisecond per pixel, which can potentially saturate communication channels. One way to prevent this is to encode the value that each spike represents in the time it is sent. Furthermore, Delorme and Thorpe used time-coded spikes to recognize faces with as little as one spike per pixel [8]; and some theories of neural computation require time encoding [9].

First we propose to linearly encode the number of thresholds exceeded. To do this the result of Equation 1 should be interpreted as the current bin number C_b (if a $1ms$ bin width was used). Similarly to the previous case, the maximum number of bins is $N_b = T/W_b$, which means the maximum brightness difference possible is $T \cdot H$.

Figure 2 shows the value-to-spike-time relation, in our proposal earlier spikes represent larger changes in intensity. The main advantage of this encoding is that a single spike could represent multiple rate-based spikes, though the encoded values are limited by time resolution and the frame rate of the camera.



(a) Possible values after a spike has been received.



(b) Time differences between current and previous spikes.

Fig. 2: Linear time-based encoding of thresholds (H) exceeded. Frame rate was 30 FPS and we used 33 time bins.

To decode the spikes on the receiver end, we must keep track of the time and value from the last collected spiked. Let Δt be the difference in arrival time between the previous and current spike (Eq. 3)

$$\Delta t = t_{now} - \left(t_{last} + \left[N_b - N_H^{last}\right] W_b\right) \tag{3}$$

where the subtraction of $N_b - N_H^{last} W_b$ shifts t_{last} to the beginning of its origin frame. We can now obtain the current spike time bin by computing the remainder of the division with time period T and dividing by the bin width.

$$C_b = \frac{\mathrm{mod}\,(\Delta t,\ T)}{W_b} \tag{4}$$

here, 'mod' calculates its arguments division remainder. Now that the time bin C_b is known, all it takes to compute the number of thresholds exceeded is

$$N_H^{now} = \lfloor N_b - C_b \rfloor \tag{5}$$

To mitigate the limitation on maximum encoded values we propose *binary encoding* of the number of thresholds exceeded. The main advantage of this technique would be to achieve large values with fewer time bins as the growth is exponential (Fig. 3). We propose to use this encoding in two ways: *shoot-and-refine* and *full value*.

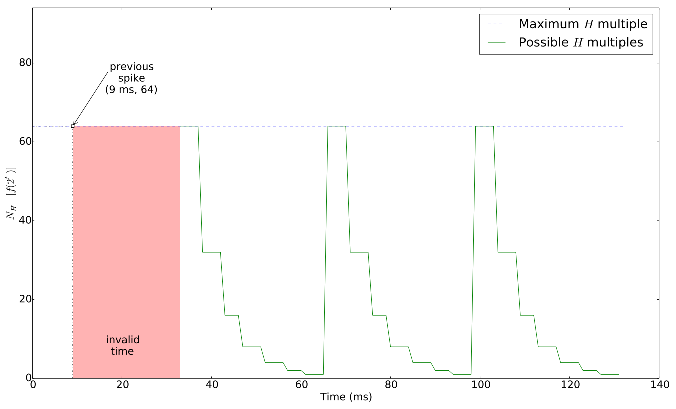


Fig. 3: Exponential time-based encoding of thresholds (H) exceeded. Frame rate was 30 FPS and we used 7 time bins.

In the *shoot-and-refine* mode, a single spike is sent with an over- or underestimate of the desired value (e.g. the next power of two), and in the following frames we send at most one spike to refine the received value towards the desired one. Decoding can be done in a similar way as the linear case, but we divide the available time in N_b bins of width is $W_b = T/N_b$. Now to compute the time difference

$$\Delta t = t_{now} - \left(t_{last} - \left[N_b - log_2(N_H^{last}) + 1\right] W_b\right) \tag{6}$$

Since we are using time bins that are larger than the system's time resolution, the term $N_b - N_H^{last} W_b$ of Eq. 3 is transformed into

$$\left[N_b - log_2(N_H^{last}) + 1\right] W_b \tag{7}$$

The correct bin is calculated using Eq. 4 as in the linear case; finally for the decoded value

$$N_H^{now} = 2^{\lfloor N_b - B \rfloor} \tag{8}$$

For the *full value* mode many spikes would be sent per pixel each frame, thus providing better resolution to the sent value, the downside to this is that an accumulation buffer is needed in addition to the previous spike time and value buffers. Decoding multiple spikes per frame has to be split in two cases: first if the new spike arrives before the current period is finished, then we accumulate its value to the current decoded value; otherwise we replace the contents of the accumulation buffer with the newly decoded value. Figure 4 shows the spike representation of an MNIST digit using the proposed encoding mechanisms. The image was scaled-down to 8×8 for clarity and the reference frame's values were initialized at half the scale range.

IV. Additional Behaviours

In this segment we describe modifications done to the basic DVS emulator, these changes are rendered in Figure 5. A history decay mechanism was added to the receiver model, constant threshold has been exchanged by an adaptive version, and an inhibitory block (*max*) completes the list of changes.

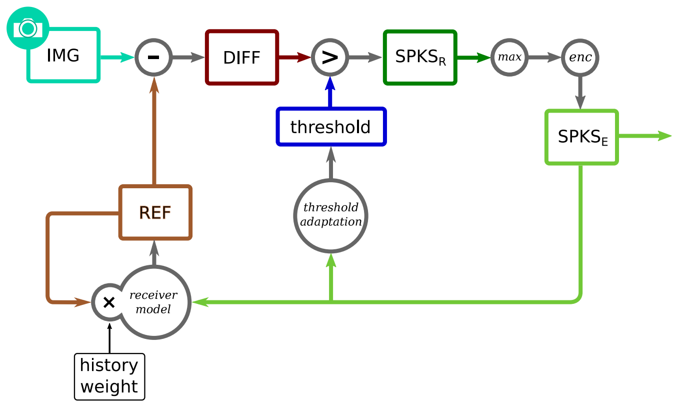


Fig. 5: DVS emulation with adaptive thresholds, local inhibition and history decay.

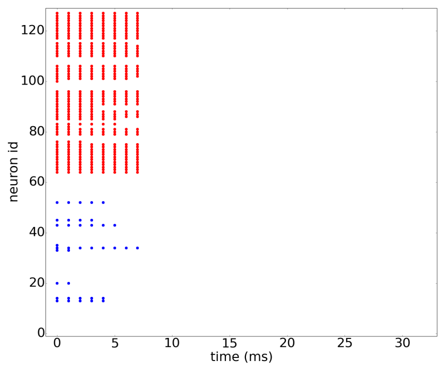
A. History decay

Initially we described a system where every spike that is sent will be captured on the receiver end, but this is not always the case. To cope with the latter cases, we now introduce a history decay mechanism which will allow the receiver to, in the long term, recover from missing spikes. Let $D \in \mathbb{R} = (0, 1]$ be the weight history has to calculate the new reference value, then the reference's update rule becomes

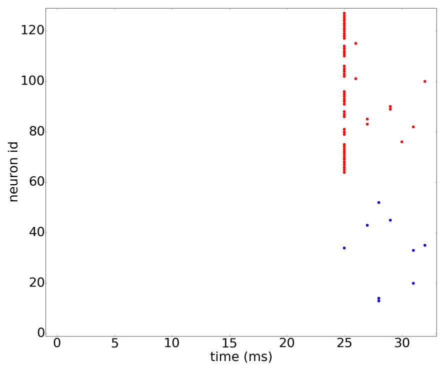
$$R_{now} = D \cdot R_{last} + N_H \cdot H \tag{9}$$

B. Adaptive threshold

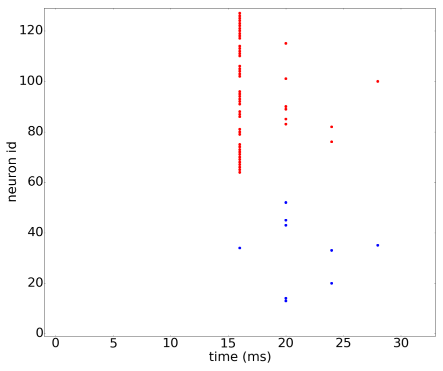
A subtle detail that other emulators have not captured is the slow-changing pixels, cameras would capture these as similar values in each frame thus the difference would never be enough to generate a spike. Meanwhile in a real DVS pixels receiving insufficient light to immediately trigger a spike, still gain some charge and, after some time, will generate a spike event. We propose to mimic this behaviour by adapting the threshold in a per-pixel basis, that is reduce it if a pixel did not create a spike. Since thresholds value may lowered, they also



(a) Rate coded spikes.



(b) Time coded spikes (linear).



(c) Time coded spikes (exponential).

Fig. 4: Difference between spike encodings for the first wave of spikes after the presentation of a MNIST digit.

have to be increased if a spike was generated. These changes in the threshold effectively add a low-pass filter to the system.

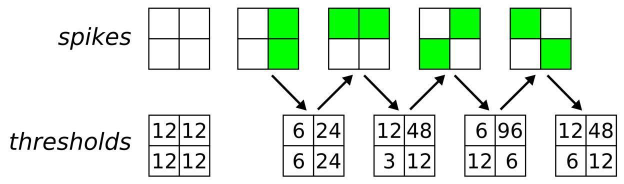


Fig. 6: Adaptive threshold behaviour.

C. Local inhibition

In mammalian retinas inhibitory circuits play an important role. Some researchers have suggested that it reduces the number of spikes needed to represent what the eye is sensing [10]. Our emulator's inhibition mechanism follows a similar idea, since neighbouring pixels have similar values, we suppose that they are transmitting redundant information. The inhibitory behaviour is simply a local MAX operation (similar to complex cells in the HMAX model [11]) of pixel areas. An example is shown in Figure 7, the maximum value (in green; 77) will generate a spike, while other values (in red; 0, 31, and 15) are blocked.

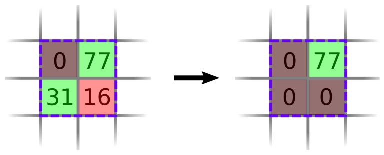
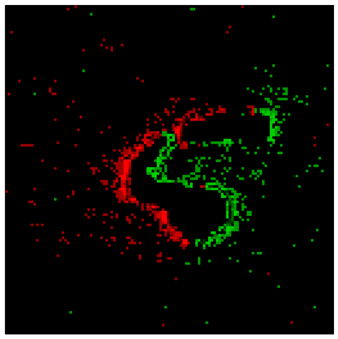


Fig. 7: Local inhibition mechanism, quantities represent the absolute value of the difference between an image and the reference. Green and red mean a spike event, brown pixels did not spike

V. Results

The emulator was developed and tested in a desktop computer (Intel i5, 8GB RAM) using the Python and Cython programming languages. We targeted a maximum 128 × 128-pixel resolution, which can perform at a 60 FPS (lower resolutions -64, 32 and 16 pixel- are also available, and can run at higher frame rates). This project is open source and it's available at https://github.com/chanokin/pyDVS.

The initial goal was to provide an alternative for computational neuroscientists that required visual input but could not afford a real DVS. To test the emulators compatibility with neuromorphic hardware, we created a PyNN-compatible [12] code template that communicates to the SpiNNaker platform [13] over Ethernet. Figure 8


(a) Recording from DVS emulator.

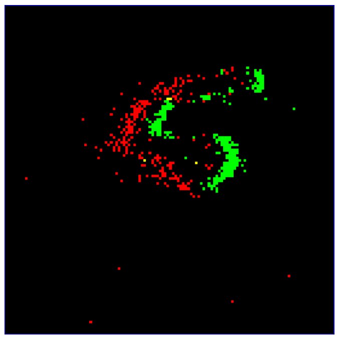
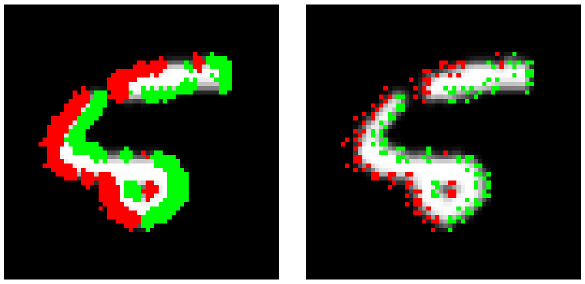

(b) Recording from silicon retina [4].

Fig. 8: DVS VS CAM!!!!!!!!!!!!!!!! 4evaaaaa!!!!!!!!!!!

The emulator can natively encode videos and we provide a "virtual camera" that simulates movement on images so they can also be perceived. Using the virtual camera and the MNIST hand-written digits [14] we demonstrate the emulator's behaviours.

The inhibitory behaviour will reduce the amount of spikes that the emulator produces per frame, while keeping some of the information needed to represent the visual input. Figure 9a shows the detected spikes as the digit traverses to the right, after the inhibition step fewer spikes remain while keeping the overall shape (Fig. 9b). Since not all the information is sent on the first frame and objects in video generally do not move fast enough to disappear, there is a *persistence of vision*-like effect of spikes for succeeding frames.



(a) Raw spikes. (b) Inhibited spikes.

Fig. 9: Difference between raw and local inhibited spikes from a traversing image.

Although the majority of neuromorphic communications are fire-and-forget [15], we were concerned that some information may be lost during transmission and so a history decay mechanism is included to cope with this problem. An example of how both sides of the transmission line can recover from a loss of spikes is shown in Figure 10; of particular interest are the *sender* and *receiver* rows, which show what both ends "see". The leftmost column illustrates how the system starts and what spikes are going to be sent. If some of the spikes are lost (second column) we cannot reconstruct the picture correctly on the receiver end. After 40 waves of spikes (rightmost column), the absolute difference ($|send - recv|$ row) between the images from both ends has pixels whose average value is 8; this means that the missing information has been retransmitted due to history decay. Another effect of this mechanism is that the need to constantly move the image is no longer needed since the reference image's value tend to zero.

VI. Conclusion

An important contribution of the field of computer vision research has been the development of image and video databases. In order to utilize them in spiking neural networks without pointing a real DVS at a monitor, we developed the emulator presented here. Using well known datasets, allows an easier comparison between spiking and traditional neural networks.

Emulating a DVS provides the flexibility to modify the system's behaviour in software. One of such changes is to encode values represented by spikes using time instead of rate. By providing such output encoding, this work may incentivize scientist to depart from rate-coded SNNs and explore time-coded ones.

Our inhibitory component has the side-effect similar to persistence of stimuli, but in this case neighbouring pixels tend to keep the similar shapes in succeeding frames.

VII. Future work

While the image resolution of the current version of our emulator is low, we've developed an OpenCL version. By

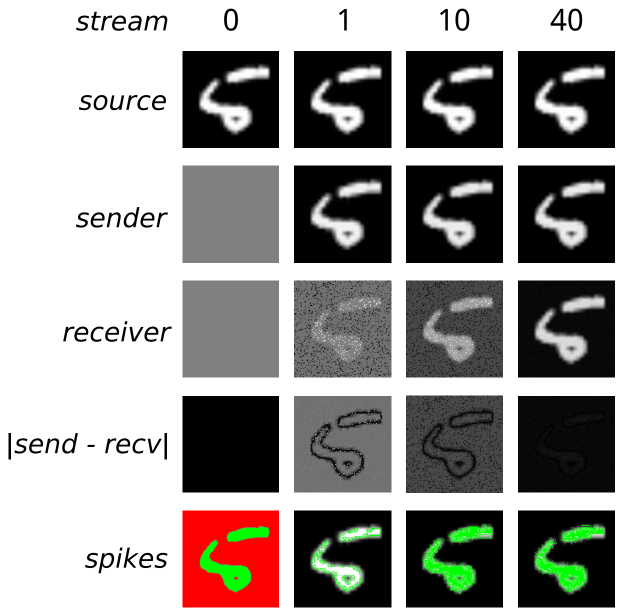


Fig. 10: History decay helps to remove transmission errors (1 spike per pixel with linear time encoding).

using the parallel processing nature of Graphics Processing Units it was possible to encode images at higher resolutions[1]; the main problem with this large imagery is that serializing and transmitting such quantities of spikes has proven a hard task.

Research on encoding spikes using convolution kernels is ongoing. We've explored using a kernel based on Carver Mead's original silicon retina [2] connectivity and biologically inspired difference of Gaussian kernels [10]. These types of encoding could prove to be more efficient as a single spike would represent a region of the image instead of a single pixel.

Acknowledgement

EPSRC???
HBP???
SEP
Talks at Capo Caccia

References

[1] D. L. Snyder and M. I. Miller, *Random point processes in time and space*. Springer Science & Business Media, 1991.

[2] C. Mead, *Analog VLSI Implementation of Neural Systems*, C. Mead and M. Ismail, Eds. Boston, MA: Springer US, 1989.

[3] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 db 15 us latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[4] J. A. Lenero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6 us latency asynchronous

[1]We tested up to 1080p video at 30 FPS

frame-free event-driven dynamic-vision-sensor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, June 2011.

[5] M. L. Katz, K. Nikolic, and T. Delbruck, "Live demonstration: Behavioural emulation of event-based vision sensors," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, May 2012, pp. 736–740.

[6] 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.

[7] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[8] A. Delorme and S. Thorpe, "Face identification using one spike per neuron: resistance to image degradations," *Neural Networks*, vol. 14, no. 6-7, pp. 795–803, 2001.

[9] E. M. Izhikevich, "Polychronization: computation with spikes," *Neural computation*, vol. 18, no. 2, pp. 245–282, 2006.

[10] B. S. Bhattacharya and S. B. Furber, "Biologically inspired means for rank-order encoding images: A quantitative analysis," *IEEE Transactions on Neural Networks*, vol. 21, no. 7, pp. 1087–1099, July 2010.

[11] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature neuroscience*, vol. 2, no. 11, pp. 1019–1025, 1999.

[12] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, no. 11, 2009.

[13] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, Dec 2013.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[15] A. D. Rast, A. B. Stokes, S. Davies, S. V. Adams, H. Akolkar, D. R. Lester, C. Bartolozzi, A. Cangelosi, and S. Furber, *Neural Information Processing: 22nd International Conference, ICONIP 2015, November 9-12, 2015, Proceedings, Part IV*. Springer International Publishing, 2015, ch. Transport-Independent Protocols for Universal AER Communications, pp. 675–684.