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

Pineda García, Garibaldi Camilleri, Patrick Liu, Qian Furber, Steve

May 31, 2016

Abstract

Vision is one of our most important senses, a vast amount of information is perceived through our eyes. Neuroscientists have performed several 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. Recently neuromorphic Dynamic Vision Sensors (DVSs) have surfaced and, while they have excellent capabilities, they remain scarce and, in some cases, difficult to use.

We propose a visual input system inspired by the behaviour of a DVS, but using a digital camera as a sensor. By using readily-available components, we believe most scientists would have access to a realistic spiking visual input source. While our primary goal was 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 contributions are adding locally inhibitory behaviour, adaptive thresholds and time-based encoding of the output spikes.

1 Introduction

In recent years the increments in computer processors' performance have been smaller; 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 like computer graphics. Meanwhile tasks like 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.). In order to achieve better performance for similar tasks on computers, scientists have looked into biology for inspiration. This has lead to the rise of brain-like (neuromorphic) hardware, which looks to mimic functional aspects of the nervous system. We can divide neuromorphic hardware into sensors (providing input) and computing devices (which make use of information from sensors). Visual input has been traditionally obtained from images that are rate-encoded using Poisson processes, while this might be a biologically-plausible encoding in the first phase of a "visual pipeline" it is unlikely that retinas transmit as much information into later stages. Furthermore, if we think of it in terms of digital networks, having each pixel represented by a Poisson process incurs in high bandwidth requirements.

In 1998, Mead proposed a silicon retina consisting of individual photoreceptors and a resistor mesh that allowed nearby receptors to influence on the output of a pixel [1]. Later, researchers developed frame-free Dynamic Vision Sensors (DVSs) [2], [3]. They feature independent pixels that emit a signal when its log-intensity value changes above a certain threshold. These sensors have μ-second response time and excellent dynamic range properties, although they are still not as widely available as regular cameras.

Katz, Nikolic, and Delbruck developed a DVS emulator in order to test behaviours for new sensor models [4]. 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 how many times the difference would pass the threshold. This emulator has been merged into their jAER project, a Java-based Address-Event Representation software framework that specializes on processing DVS output in real time.

In this work, we propose to emulate the behaviour of a DVS using a conventional digital camera as a sensor. Basing the emulator on widely available hardware, would allow most computational neuroscientists to include video as a spike-encoded input.

2 Our work

As previous emulators [4], our basic model works by analysing the difference of the latest frame captured from the camera and a reference frame. In our emulator we consider that most commercial cameras produce gamma-encoded images [5] to better utilize bits 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.

Figure 1 shows the basic DVS emulation diagram, first asynchronous pixel behaviour is approximated via differencing the current image obtained by the camera and a reference frame. To simulate the spike emission we set a rule, so that whenever a pixel's difference is

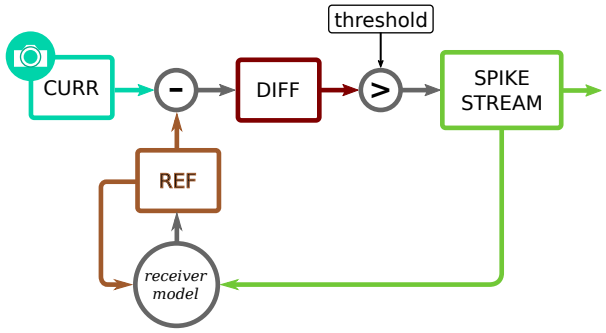


Figure 1: DVS emulation diagram. Circles indicate operations and rectangles stages of visual information (from frames to spike trains)

larger than a certain threshold, we mark that position as "spiked". Each spike carries a flag, *up* if there was a positive change in brightness or *down* if a decrease was registered, this can also be thought of as a spike's *sign*. Finally, depending on the selected type of output, we simulate a receiver and update the reference frame accordingly.

2.1 Output modes

2.1.1 Rate-based

As in previous emulators[4], 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 N_H is the number of spikes needed to represent the change in intensity 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 it's only possible to send one spike per millisecond, then $T = 1000/fps$. At this stage we model a perfect receiver, so the update rule for the reference is

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

2.1.2 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.

First we propose to linearly encode the number of thresholds exceeded. To do this the result of Equation 1 should be interpreted as a bin number B (if a $1ms$ bin width was used). As in the previous case the time to send all spikes for the current frame is at most $N_b = T/W_b$, which means the maximum 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.

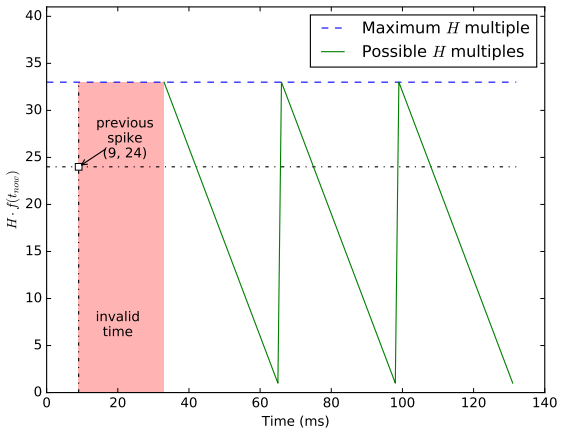


Figure 2: Linear time-based encoding of thresholds (H) exceeded

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)

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

where the subtraction of $N_b - N_H^{last} W_b$ sets the time reference to a multiple of time period T. We now divide the time difference by the time-bin width to obtain the current bin

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

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

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

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*.

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

$$\Delta t = t_{now} - \left(t_{last} - 2 * \left[N_b - log_2(N_H^{last}) W_b + 1\right]\right) \tag{6}$$

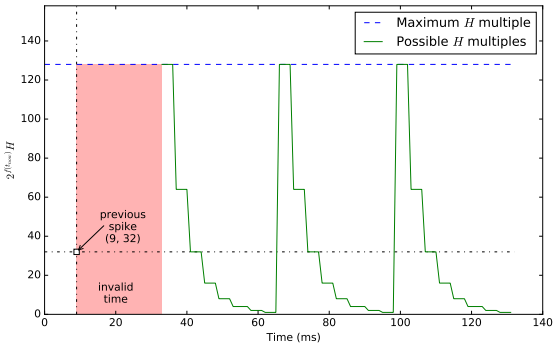


Figure 3: Exponential time-based encoding of thresholds (H) exceeded

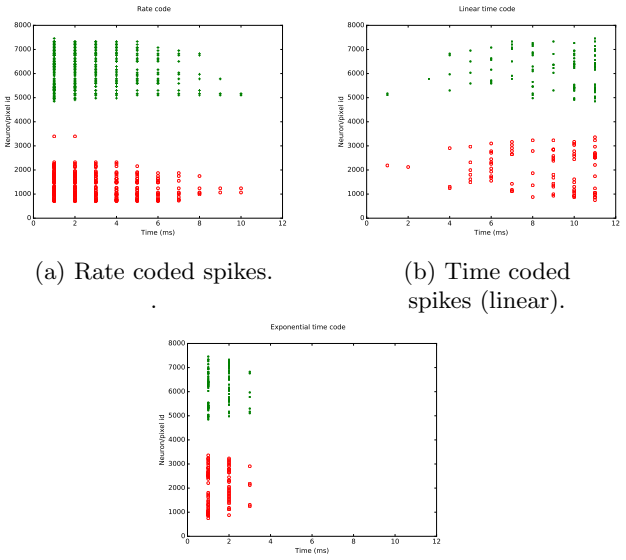
Since we are using time bins that are larger than the system's time resolution, the term $N_b - N_H^{last} W_b$ of Eq. 3 is transformed into

$$2 * \left[N_b - log_2(N_H^{last}) W_b + 1\right] \tag{7}$$

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.



(a) Rate coded spikes.

(b) Time coded spikes (linear).

(c) Time coded spikes (exponential).

Figure 4: Difference between spike encodings for the first presentation of an MNIST digit.

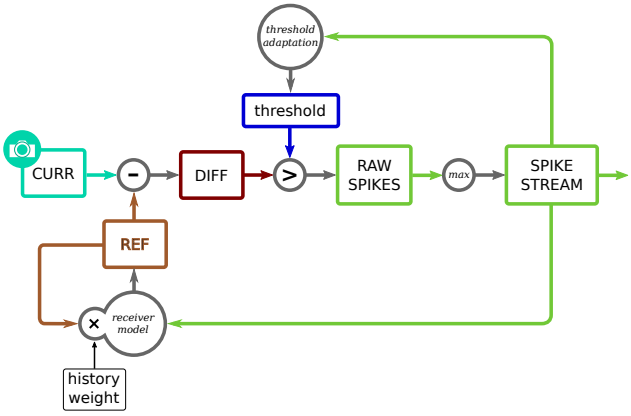


Figure 5: DVS emulation with adaptive thresholds, local inhibition and history decay.

2.2 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.

2.2.1 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

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

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

2.2.3 Local inhibition

In mammalian retinas inhibitory circuits play an important role. Some researchers have suggested that it

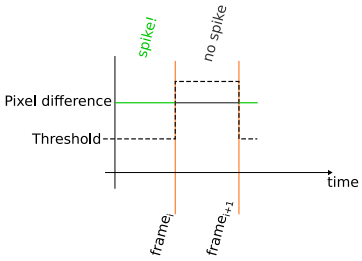


Figure 6: Adaptive threshold behaviour.

reduces the number of spikes needed to represent what the eye is sensing [6]. 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 [7]) 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.

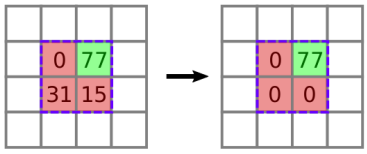
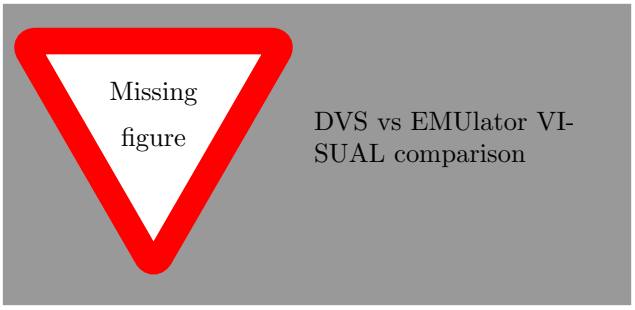


Figure 7: Local inhibition mechanism, quantities represent the absolute value of the difference between an image and reference.

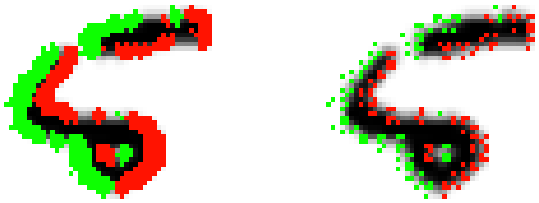
3 Results

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 [8] code template that communicates to the SpiNNaker platform [9] over Ethernet; it 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 [10] we demonstrate the emulator's behaviours.



The inhibitory behaviour will reduce the amount of spikes that the emulator produces, while keeping some of the information needed to represent the visual input. Figure 8a shows the detected spikes as the digit traverses to the right, after the inhibition step fewer spikes remain while keeping the overall shape (Fig. 8b).



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

Figure 8: Difference between raw and local inhibited spikes from a traversing image (inverted images[1]).

Although the majority of neuromorphic communications are fire-and-forget [11], 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 9; 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.

4 Conclusion

An important contribution of the field of computer vision research has been the development of image and

[1]Notice that most images in the tests are shown inverted. They originally where white over black and are now altered to black over white to better capture the behaviour

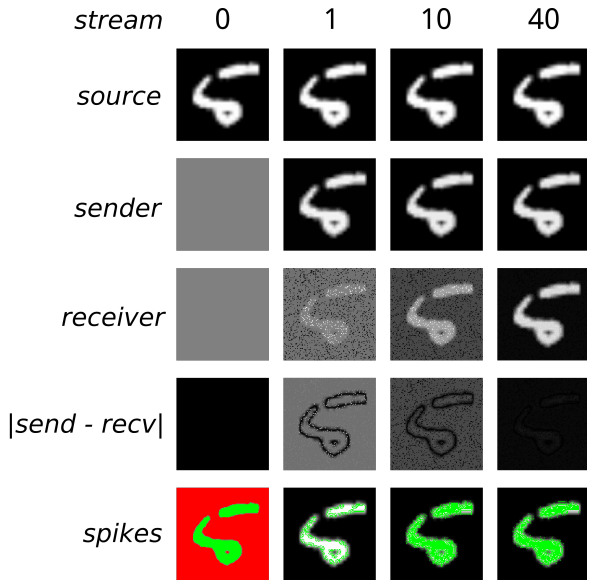


Figure 9: History decay helps to remove transmission errors (1 spike per pixel with linear time encoding).

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. Utilising 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 utilize time-encoded spikes, which can aid to break the first barrier to study networks that utilize this method of spike encoding.

5 Future work

While the image resolution of the current version of our emulator is low, we've developed an OpenCL version. By using the parallel processing nature of Graphics Processing Units it was possible to encode images at higher resolutions[2]; 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 [1] connectivity and biologically inspired difference of Gaussian kernels [6]. 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.

References

[1] C. Mead, *Analog VLSI Implementation of Neural Systems*, C. Mead and M. Ismail, Eds. Boston, MA: Springer US, 1998, ch. Adaptive Retina, pp. 239–246, ISBN: 978-1-4613-1639-8.

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

[2] 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, 2008, ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

[3] J. A. Lenero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6 us latency asynchronous frame-free event-driven dynamic-vision-sensor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, 2011, ISSN: 0018-9200. DOI: 10.1109/JSSC.2011.2118490.

[4] 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*, 2012, pp. 736–740. DOI: 10.1109/ISCAS.2012.6272143.

[5] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, ISBN: 1558607927.

[6] 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, 2010, ISSN: 1045-9227. DOI: 10.1109/TNN.2010.2048339.

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

[8] 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, ISSN: 1662-5196. DOI: 10.3389/neuro.11.011.2008. [Online]. Available: http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.011.2008/abstract.

[9] 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, 2013, ISSN: 0018-9340. DOI: 10.1109/TC.2012.142.

[10] 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, 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791.

[11] 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," in, S. Arik, T. Huang, K. W. Lai, and Q. Liu, Eds. Cham: Springer International Publishing, 2015,

ch. Transport-Independent Protocols for Universal AER Communications, pp. 675–684, ISBN: 978-3-319-26561-2. DOI: 10.1007/978-3-319-26561-2_79. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26561-2_79".