pyDVS: An Extensible, 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, qian.liu2, steve.furber
@manchester.ac.uk

† Entrepreneur First
100 Clements Road
London, United Kingdom
patrick.camilleri@gmail.com

***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 its 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 live 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 framework which can be extended, as we demonstrate by adding local inhibitory behaviour, adaptive thresholds and spike-timing encoding.

I. Introduction

In recent years the rate of increase of individual 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 is interpreted as a neuron that will fire at a rate 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 sensors feature independent pixels, which emit a signal when their log-intensity values change by a certain threshold. These sensors have microsecond response time, excellent dynamic range properties and frame-free output, although they are (still) not as widely available as conventional cameras and are relatively expensive.

An alternative that could reduce the cost and scarcity of DVSs while keeping spike rates low is to emulate the behaviour of a DVS. Katz et al. developed a DVS emulator to test behaviours for new sensor models [5]. In their work, they transform video [at 125 frames per second (FPS)] provided by a commercial camera into a spike stream. 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 is 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 an extensible 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 without the cost of a DVS. We present our basic emulator in Section II. In Section III we present our extensions to the emulator. Results are given in Section IV; conclusions and suggestions for future work are given in Sections V and VI, respectively.

II. The basic emulator

A Dynamic Vision Sensor is a device which transforms intensity changes (in a logarithmic scale) into events. In conventional video cameras pixels capture light to form an image and repeat this process in a regular time period (T), these images are usually called frames (Figure 1).

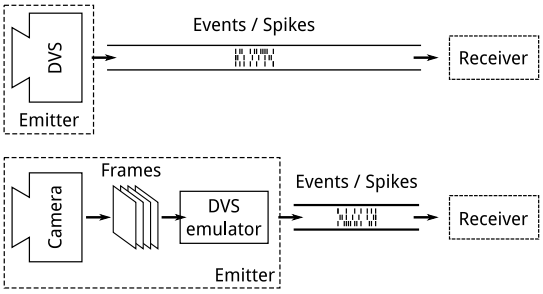


Fig. 1: DVS and emulator diagrams.

To emulate a frame-free system, we convert pixels' brightness differences into events. Our emulator works by analysing the difference between the latest frame captured from the camera and a reference frame. If a pixel changes by more than a certain threshold (H), then we generate an event which contains the pixel's coordinates and whether the change was positive or negative.

Figure 2 shows the DVS emulation framework diagram, we obtain an image (IMG) from a video source and subtract a reference frame from it (REF). We then apply a threshold filter to the difference frame (DIFF), the remaining pixels are considered *raw spikes* (SPKS_R). We can optionally post-process these pixels, as we'll demonstrate in Section III-D, but we must encode them 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.

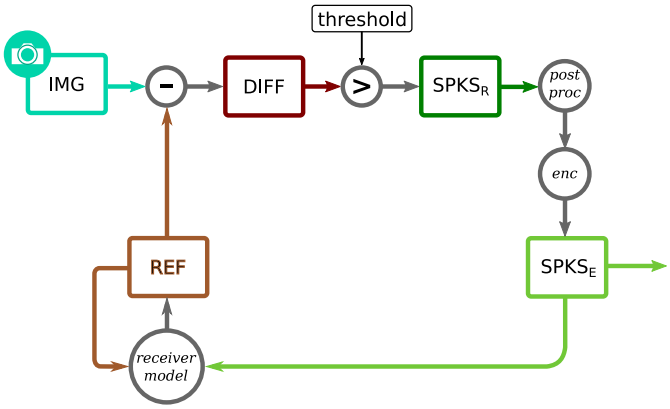


Fig. 2: DVS emulation diagram. Circles indicate operations and rectangles stages of visual information (from frames to spike trains).

Pixels in 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 (Fig. 3), we use frame's brightness levels without modifications.

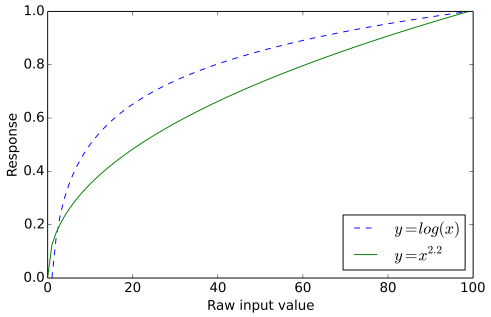


Fig. 3: Logarithmic and Gamma response functions (normalized, $\gamma = 2.2$)

We have constrained the output of our system to emit events in discrete time, so we divide time period T into N_b bins of width t_b (Fig. 4), and neurons (representing pixels) are only allowed to spike once per time bin. For example, if time interval $T = 10ms$ and $N_b = 10$ then $t_b = 1ms$, so neurons would fire at most once per millisecond. Since we are developing a real-time system, all events should be emitted between frames, so the maximum number of spikes per neuron per frame is N_b.

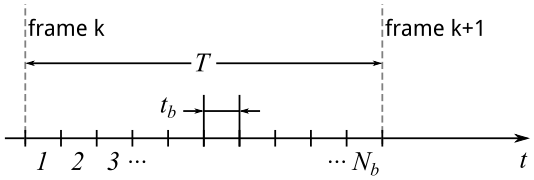


Fig. 4: Time discretization.

As in previous emulators [5], our basic emulator's output format is rate-based. Each emitted spike signifies the pixel changed by H brightness levels, where H is also the threshold. Let N_H be the integer division of a pixel's brightness change ΔB by the threshold H and limiting by N_b, the number of spikes (N_s) needed to represent ΔB will be

$$N_s = \min(N_b,\ \ N_H) = \min\left(N_b,\ \ \left\lfloor \frac{\Delta B}{H} \right\rfloor\right) \qquad (1)$$

At this stage we model a perfect receiver, so the update rule for the reference is

$$R_{now} = R_{last} + N_s \cdot H \qquad (2)$$

III. Emulator extensions

In its worst case rate-based encoding can send a spike per millisecond per pixel each frame, which is not biologically plausible and can potentially saturate communication channels.

Two of the extensions we have developed diminish the rate of spikes, either by changing output encoding (Sec. III-A) or post-processing generated spikes (Sec. III-D).

Another extension adds robustness to transmission by adding a history decay mechanism (Sec. III-B). The final extension replaces the constant threshold for an adaptive version (Sec. III-C). Figure 5 shows modifications on the basic DVS emulator.

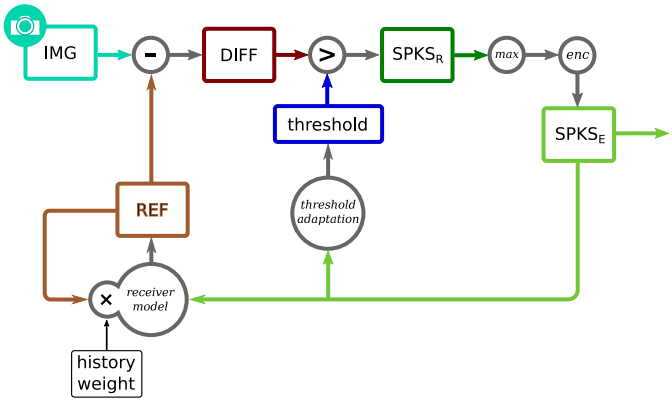
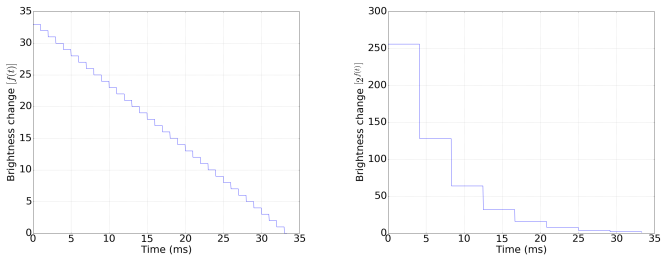


Fig. 5: DVS emulation with adaptive thresholds, local inhibition and history decay.

A. *Spike-time encoding*

One way to prevent high spike rates is to encode the brightness difference in the time a spike is emitted, similar to how pulse-position modulation works [8]. That is each time bin represents a brightness change, in either a linear or logarithmic scale. Furthermore, there's evidence that neurons use spike timing to encode values [9] and some theories of neural computation require spike-time coding [10].



(a) Linear response ($N_b = 33$, $t_b = 1.1$).

(b) Logarithmic response ($N_b = 8$, $t_b = 4.16ms$).

Fig. 6: Possible values for a spike timing code, linear (left) and logarithmic (right) scales ($FPS = 30, T = 33.3ms$).

Since brightness variations happen when a new frame is captured, spike times are referenced to the beginning of each frame. To synchronize the emitter with the receiver, we send one spike for each pixel at the beginning of the simulation ($t = 0$) and all of these encode a predefined value (in this case the maximum one). Figure 6 shows a single frame, spikes that were sent closer to the frame's beginning encode larger values. This means that if spikes were emitted at t_0 and t_1, then a pixel changed X and Y times the threshold, respectively. In our system, if $t_0 < t_1$ then $X > Y$.

We will first present the linear scale case, we can calculate the appropriate time bin C_b with

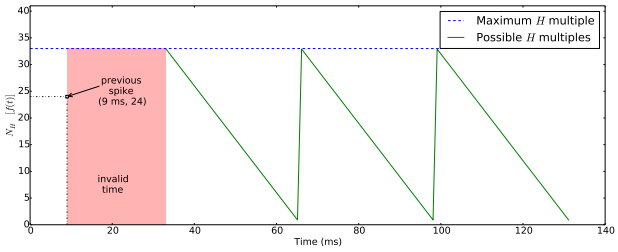
$$C_b = N_b - \min\left(N_b,\ \ N_H\right), \tag{3}$$

and the time at which the spike will be sent is given by

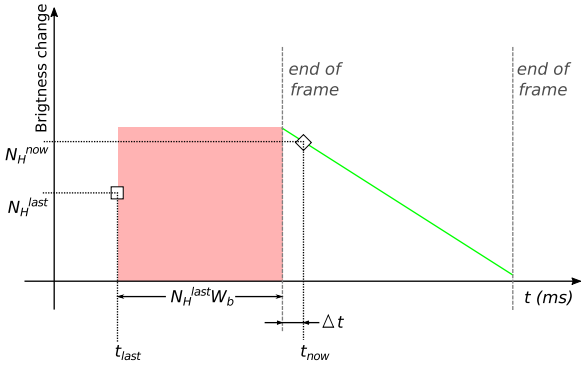
$$t_s = C_b t_b \tag{4}$$

The main advantage of this encoding is that a single spike could represent multiple (at most N_b) rate-based spikes, though the encoded values are limited by time resolution and the frame rate of the camera.

If a receiver wants to decode the spikes, it needs to figure out the time at which each frame begins, but we do not provide information about this event. A possible solution involves keeping track of the time and value from the last collected spike, by keeping these values we can calculate the end of the previous spike's frame. The green line in Figure 7a shows this spike time-to-value relation.



(a) Possible values for spikes received after another.



(b) Time differences between current and previous spikes.

Fig. 7: Spike-time coding. Decoding values in linear scale ($FPS = 30, T = 33.3ms$).

Let Δt be the difference in arrival time between the current spike and the end of the previous spike's frame (Fig. 7b, center). To calculate it we use

$$\Delta t = t_{now} - \left(t_{last} + N_H^{last} t_b\right) \tag{5}$$

where $N_H^{last} W_b$ shifts t_{last} to the end of its frame. By having Δt anchored at the beginning of a frame we can compute its bin with the remainder of the division with time period T.

$$C_b^{now} = \frac{\mathrm{mod}\left(\Delta t,\ \ T\right)}{t_b} \tag{6}$$

here, 'mod' calculates its arguments division remainder. Now that the time bin C_b is known we can compute the brightness difference with

$$\Delta B = H \cdot N_H^{now} = H\left(N_b - C_b^{now}\right) \tag{7}$$

We've mentioned before that the maximum brightness difference in linear scale is $N_b \cdot H$, by using a logarithmic scale we can encode larger brightness differences and even do so with wider time bins. This also reduces the chances of wrongly decoding spikes. If the maximum brightness difference possible is 255 and the threshold is 12, we only need 5 time bins to express the largest change. The time bin can be calculated with Equation 8. Figure 8 shows the relation of spike time and brightness change with logarithmic scale.

$$C_b = N_b - \min\left(N_b,\ \ \lfloor \log_2 N_H \rfloor\right) \tag{8}$$

The time at which the spike would be sent, with respect to the beginning of the frame would be $t_s = C_b t_b$. Since a single spike per pixel per frame is emitted, we are sending an underestimate of the desired value (i.e. previous power of two). In the following frames we send spikes to refine the received value towards the desired one [11].

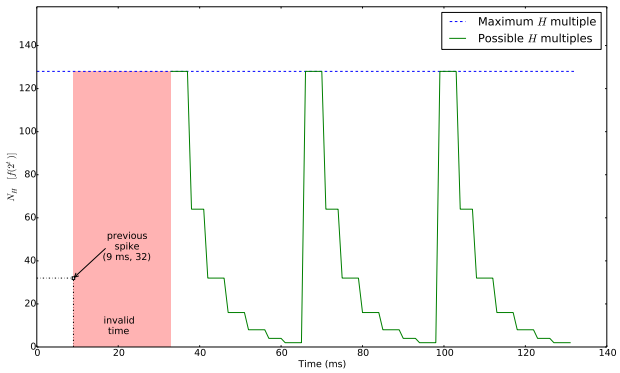


Fig. 8: Spike-time coding. Decoding values in logarithmic scale. Frame rate was 30 FPS and we used 7 bits (time bins).

Decoding (Fig. 8) can be done in a similar way to the linear case, but the receiver can only record an approximate version of N_H, that is $\tilde{N}_h = 2^{\lfloor \log_2 N_H \rfloor}$. The time difference with respect to the end of frame for the previous spike is

$$\Delta t = t_{now} - \left(t_{last} + \log_2 \tilde{N}_H t_b\right) \tag{9}$$

The current spike's time bin is calculated using Eq. 6 and the decoded value with

$$\Delta B = H \cdot N_H^{now} = H \cdot 2^{(N_b - \lfloor C_b^{now} \rfloor)} \tag{10}$$

If we allow many spikes to be sent per pixel per frame, the receiver gets a closer approximation of N_H. The downside to this is that this is unlikely to be biologically plausible. Decoding multiple spikes per frame has to be split in two cases: first, if the new spike arrives before the current frame period is finished, then we accumulate its value to the current decoded value; otherwise we keep the newly decoded value.

B. History decay

Initially we described a system where we assume every spike sent will be captured and properly decoded by the receiver, but this is not always the case. To cope with failures in transmission, 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 for the reference frame in the calculation of its new value, the update rule becomes

$$R_{now} = D \cdot R_{last} + N_H \cdot H \tag{11}$$

If no spikes are sent, the values in the reference frame tend to 0 which corresponds to black in our tonal scale. This can be seen as "forgetting" the information stored in the reference frame.

C. Adaptive threshold

Slow-changing pixels in cameras would capture small brightness values each frame thus the difference would never be enough to generate a spike. Meanwhile DVSs' 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 on a per-pixel basis, so we reduce the threshold if a pixel did not emit a spike. Since threshold values may be lowered, they also have to be increased if a spike was generated. These changes in the threshold effectively add a temporal low-pass filter to the system.

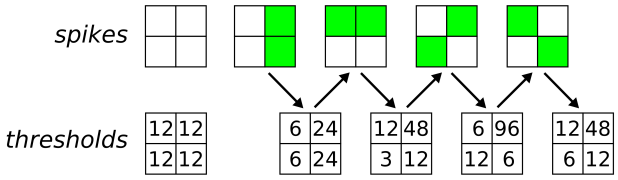


Fig. 9: Adaptive threshold behaviour.

D. Local inhibition

In mammalian retinas inhibitory circuits play an important role. Some researchers have suggested that they reduce the number of spikes needed to represent what the eye is sensing [12]. Our emulator's inhibition mechanism follows a similar idea; since neighbouring pixels have similar values, we assume that they are transmitting redundant information. The inhibitory behaviour is simply a local MAX operation (similar to complex cells in the HMAX model [13]) of pixel areas. An example is shown in Figure 10, we chose a 2-by-2 area since it's the smallest 2D one. Initially pixels whose difference was 77, 31, -16 (left side in green and red) where to emit events,

but after inhibition only the maximum value (right side in green; 77) will generate a spike, while other values (right side in brown; 0, 31, and 16) are blocked.

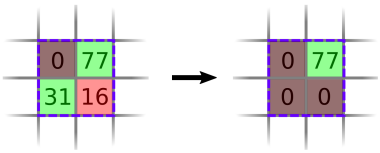
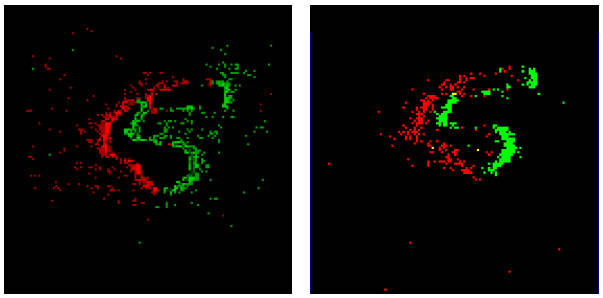


Fig. 10: Local inhibition mechanism. Quantities represent the absolute value of the difference between an image and the reference. Green and red mean a spike event, brown pixels did not spike

IV. 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 is available at https://github.com/chanokin/pyDVS.

The initial goal was to provide an alternative for computational neuroscientists who required spiking visual input but could not afford a DVS. To test the emulator compatibility with neuromorphic hardware, we created a PyNN-compatible [14] code template that communicates to the SpiNNaker platform [15] over Ethernet. We tested the behaviour of a DVS [4] and our emulator with a PS3Eye camera [16]. Both sensors were pointed to a 60 Hz LED monitor and were shown an MNIST digit traversing from left to right. Figure 11 shows a visual comparison of the behaviour of the emulator (left) and the DVS (right); the emulator's output has some noise due to automatic mechanisms (e.g. exposure, gain control, white balance).

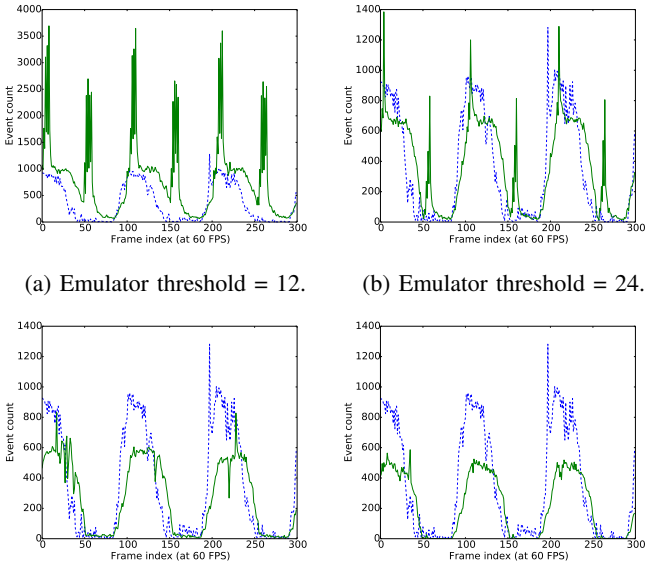


(a) Recording from DVS emulator.

(b) Recording from silicon retina [4].

Fig. 11: Visual comparison of an MNIST digit traversing a computer screen horizontally.

With the same setup we performed tests using different thresholds for the emulator running at 60 FPS. We counted the number of events for every frame in the case of the emulator and accumulated events every $16ms$ for the DVS using the jAER suite. Figure 12 shows the comparison when the emulator's threshold was set to 12, 24, 36 and 48. Events generated by the DVS are plotted with a dashed blue line and the emulator's are shown with a green line.



(a) Emulator threshold = 12.

(b) Emulator threshold = 24.

(c) Emulator threshold = 36.

(d) Emulator threshold = 48.

Fig. 12: Comparison of generated events (blue dashed - DVS, green - Emulator).

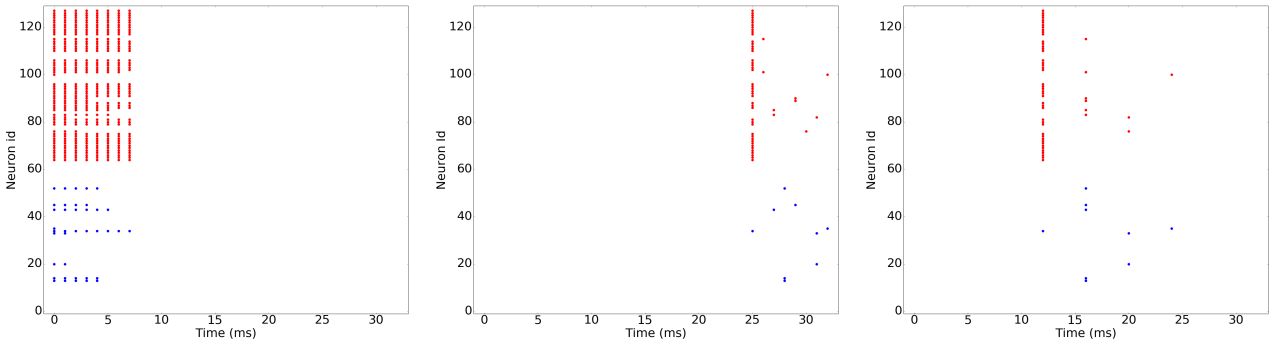
It is noticeable that the emulator generates peaks of activity when the image appears or has left the field of view of the camera (Fig. 12a and 12b). We believe this is caused by automatic exposure and gain mechanisms which we were unable to turn off. As we increase the threshold less noise is captured but we generate fewer spikes as well, thus we may lose relevant data. Although having these activity peaks is clearly different to what we expect from a DVS, we could think of them as a way to refresh the value of these pixels or adding noise to the spike representation.

Using conventional cameras, and thus our emulator, enforces certain limitations which we summarize in Table I.

TABLE I: Emulator limitations.

	DVS	Camera (PS3 Eye)
Power consumption	24 mW	$\tilde{1}$20 mW
Response time	3.6 $\rightarrow$ 15 us	8 $\rightarrow$ 33 ms
Dynamic range	120 dB	60 dB

The emulator can encode video (live or pre-recorded) 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 database [17] we demonstrate the emulator's extensions. To test encodings we converted an MNIST digit image which we scaled down to 8×8 pixels for clarity and the reference frame's values were



(a) Rate coded spikes.

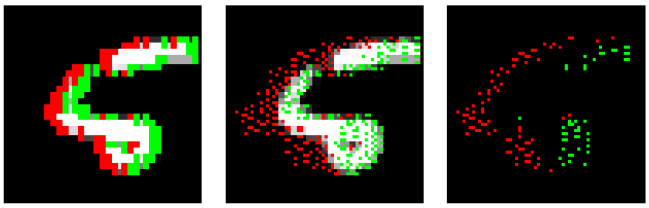
(b) Time coded spikes (linear).

(c) Time coded spikes (logarithmic).

Fig. 13: Difference between spike encodings for the first wave of spikes after the presentation of an MNIST digit.

initialized at half the scale range, Figure 13 shows the spike representation using the developed output encodings.

The inhibitory behaviour will reduce the number of spikes that the emulator produces per frame, while keeping some of the information needed to represent the visual input. Figure 14a shows the detected spikes as the digit traverses to the right. After the inhibition step fewer spikes remain while keeping the overall shape (Fig. 14b).



(a) Raw spikes.

(b) Inhibited spikes.

(c) Spikes from previous frames.

Fig. 14: Difference between raw and local inhibited spikes from a traversing image.

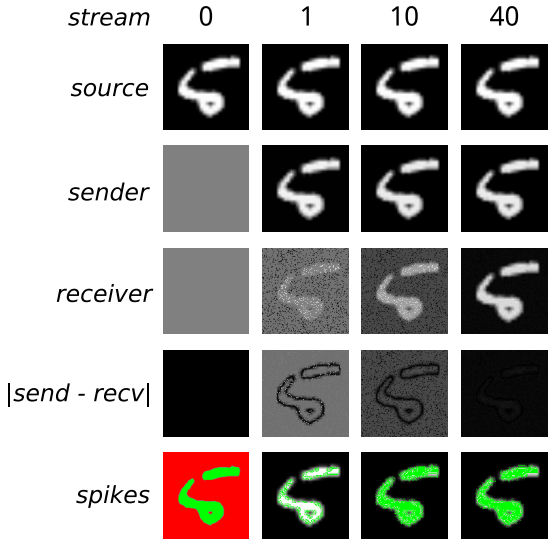


Fig. 15: History decay helps to remove transmission errors (1 spike-per-pixel with linear spike-time encoding).

Since not all the information is sent on the first frame and objects in video generally do not move fast enough to disappear between frames, there is a *persistence of vision*-like effect of spikes for succeeding frames. To asses this we removed the spikes that would be generated in the current frame without inhibition (Fig. 14a) from the inhibited spike generation (Fig. 14b), the result (Fig. 14c) shows spikes that are the result of inhibition in previous frames. This shows a figure that's quite similar to the what we should have detected in preceding frames.

Finally, we tested the weight decay mechanism, an example of how the receiver can recover from a loss of spikes is shown in Figure 15; of particular interest are the $sender$ and $receiver$ rows, which show their respective reference frames (i.e. what both "see"). The leftmost column illustrates how the system starts and what spikes are going to be sent ("spikes" row).

If some of the spikes are lost (second column), the receiver cannot reconstruct the picture correctly. After 40 waves of spikes (rightmost column), pixels in the absolute difference between the reference images ($|send - recv|$ row) have an 8.6 average value which is bellow the threshold. 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 removed since the reference values are continuously lowered; this effect furthers the difference between the image and the reference up to a point at which spikes are produced.

V. Conclusion

An important contribution to the field of computer vision research has been the development of image and video databases. To utilize them in spiking neural networks without pointing a DVS at a monitor, we developed the emulator presented here. To encode images we developed a "virtual camera" that simulates movement so the pictures can be

captured by the DVS. Two of these motions are inspired by eye movements known as saccades.

Emulating a DVS provides the flexibility to modify the system's behaviour in software. One such change is to encode values represented by spikes using time instead of rate. This allows more information to be sent per spike and lowers bandwidth requirements.

Our inhibitory component splits spike emission into, at most, four frames and it also presents a side-effect similar to persistence of stimuli. If the adaptive threshold block is used, the emulator should filter out fast changing pixels thus reducing the number of spikes emitted. After considering that not all spikes will arrive to their destination, we developed a history decay mechanism for the reference update. This has the effect of fixing transmission errors and continuously sending spikes if pixels' difference of brightness values are larger than the threshold, even without moving images.

VI. Future work

While the image resolution of the current version of our emulator is low, an OpenCL version has been developed. By using the parallel processing nature of Graphics Processing Units it was possible to encode images at higher resolutions (we have tested up to 1920×1080 video at 30 FPS); the main problem with this large imagery is that serializing and transmitting such numbers of spikes has proven a hard task.

Research on encoding spikes using convolution kernels is ongoing. We have explored, using a kernel based on Carver Mead's original silicon retina connectivity [2] and biologically inspired difference of Gaussian kernels [12]. These types of encoding could prove to be more efficient since a single spike would represent a region of the image instead of a single pixel.

Acknowledgement

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme(FP/2007-2013) / ERC Grant Agreement n. 320689 and from the EU Flagship Human Brain Project (FP7-604102).

Garibaldi Pineda García is funded by a National Council for Science and Technology of Mexico PhD Scholarship.

Patrick Camilleri and Steve Furber are supported by EPRSC project EP/K034448/1 (PRiME).

The authors would like to thank the reviewers for taking the time to review this document and provide useful feedback on the work. They would also like to thank Jim Garside and Viv Woods for help and useful discussions during the course of this paper.

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 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] J. Hamkins, *Pulse Position Modulation*. John Wiley & Sons, Inc., 2007, pp. 492–508.

[9] W. Singer, "Time as coding space?" *Current Opinion in Neurobiology*, vol. 9, no. 2, pp. 189 – 194, 1999.

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

[11] E. L. Zuch, "Where and when to use which data converter: A broad shopping list of monolithic, hybrid, and discrete-component devices is available; the author helps select the most appropriate," *IEEE Spectrum*, vol. 14, no. 6, pp. 39–43, June 1977.

[12] 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.

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

[14] 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.

[15] 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.

[16] Wikipedia. (2016) Playstation eye. Accessed July-2016. [Online]. Available: https://en.wikipedia.org/wiki/PlayStation_Eye

[17] 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.