# 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 is interpreted as a neuron that will fire a number of times 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 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 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 in order 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 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 without the cost of a DVS. 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 by 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 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.

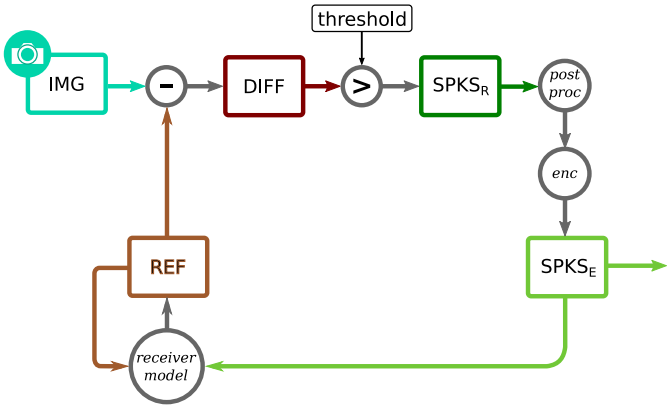



Fig. 1: DVS emulation diagram. Circles indicate operations and rectangles stages of visual information (from frames to spike trains).

Figure 1 shows the basic DVS emulation diagram, we obtain an image (IMG) from a video source and subtract from it a reference frame (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 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

In conventional video cameras pixels capture light to form an image and repeat this process in a regular time interval ($T$), these images are usually called frames. To emulate a frame-free system, we convert pixel's brightness into events. Since we are developing a real-time system, all events should be emitted between frames. For example, if the discrete time interval $T = 10ms$ and neurons (representing pixels) fire once per millisecond, then they can emit at most 10 spikes.

### A. Rate-based

As in previous emulators [5], the standard 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 $\Delta B$ by the threshold $H$ and limiting by $T$, the number of spikes ($N_s$) needed to represent $\Delta B$ will be

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

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

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

### B. Time-based

In its worst case rate-based encoding can send a spike per millisecond per pixel, which is not biologically plausible and can potentially saturate communication channels. One way to prevent this is to encode the brightness difference in the time a spike is emitted. 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].

We will again use discrete time steps, so we divide time interval $T$ into $N_b$ time bins of width $W_b$. Each bin represents a brightness change, in either a linear or logarithmic scale. We will first treat the linear scale case where earlier spikes represent larger changes in intensity and restrict pixels to emit one spike per frame. We can calculate the appropriate time bin $C_b$ with

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

finally, the time at which the spike will be sent is given by

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

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. The green line in Figure 2a shows this spike time-to-value relation.

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. Let $\Delta t$ be the difference in arrival time current spike and the end of the previous spike's originating frame (Eq. 5, Fig. 2b)

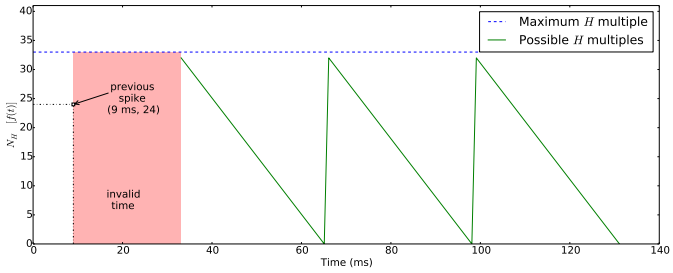
$$\Delta t = t_{now} - \left(t_{last} + N_H^{last} W_b\right) \tag{5}$$

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

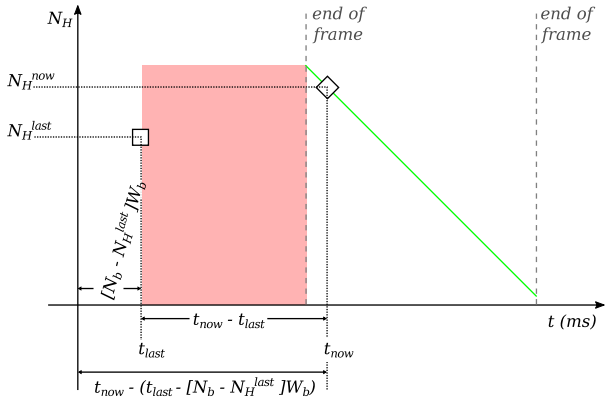
$$C_b^{now} = \frac{\mathrm{mod}\left(\Delta t,\ \ T\right)}{W_b} \tag{6}$$

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

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




(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.

By using a logarithmic scale, we can encode larger brightness differences and even do so with fewer time bins. We will study the case where earlier spikes encode larger changes, neurons fire once per frame and using base 2 logarithms. The time bin can be calculated with Equation 8, Figure 3 shows the relation of spike time and brightness change with logarithmic scale.

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

and the time at which the spike would be sent, with respect to the beginning of the frame would be $t_s = W_b C_b$.

Since a single spike is sent, we are sending an underestimate of the desired value (i.e. previous power of two), and in the following frames we send at most one spike to refine the received value towards the desired one [10].

Decoding 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 $\tilde{N}_h = \log_2 N_H$. The time difference with respect to the end of frame for the previous spike is

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

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 each frame, the receiver gets a closer approximation of $N_H$. The downside

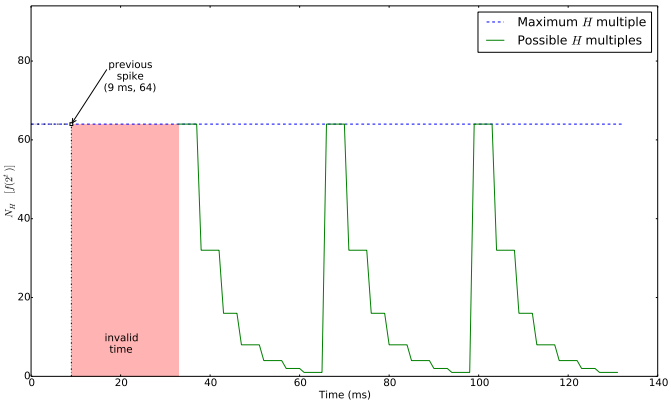



Fig. 3: Exponential time-based encoding of thresholds ($H$) exceeded. Frame rate was 30 FPS and we used 8 bits (time bins).

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. To test encodings we converted a handwritten digit image from the MNIST database [11]. We scaled the image down to $8 \times 8$ pixels for clarity and the reference frame's values were initialized at half the scale range, Figure 4 shows the spike representation using the proposed encodings.

## IV. Additional Behaviours

In this section 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, the constant threshold has been replaced for an adaptive version and an inhibitory block (*max*) completes the list of changes.

### A. History decay

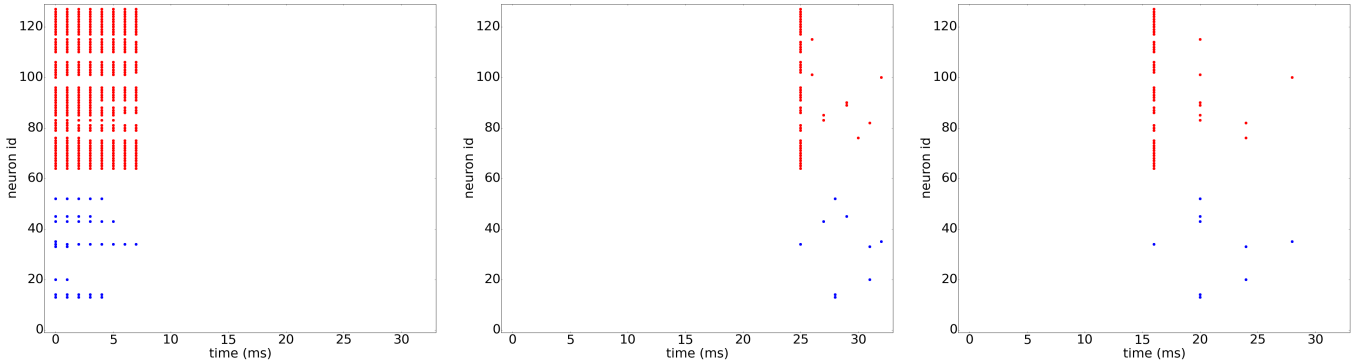
Initially we described a system where every spike that is sent will be captured by the receiver, but this is not always the case. To cope with the latter cases, we now introduce a history decay mechanism which will allow the receiver to, in the long term, recover from missing spikes. Let $D \in \mathbb{R} = (0, 1]$ be the weight 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 frame.

### B. Adaptive threshold

A subtle detail 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 DVSs' pixels receiving insufficient light to immediately trigger




(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.

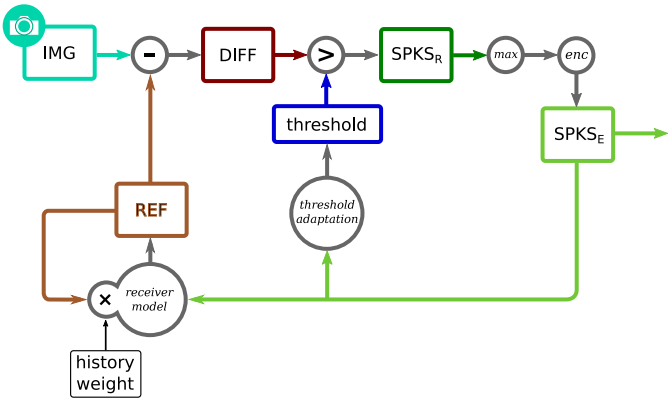



Fig. 5: DVS emulation with adaptive thresholds, local inhibition and history decay.

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, that is reduce it if a pixel did not create 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 low-pass filter to the system.

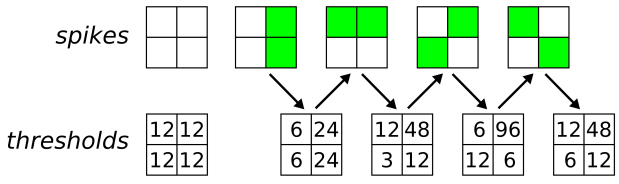



Fig. 6: Adaptive threshold behaviour.

### C. Local inhibition

In mammalian retinas inhibitory circuits play an important role. Some researchers have suggested that 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 resembling 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 [13]) of pixel areas. An example is shown in Figure 7, the maximum value (in green; 77) will generate a spike, while other values (in brown; 0, 31, and 16) are blocked.

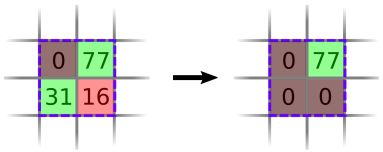
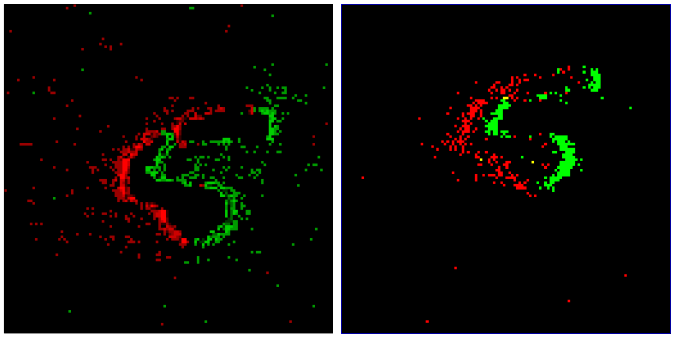



Fig. 7: Local inhibition mechanism, quantities represent the absolute value of the difference between an image and the reference. Green and red mean a spike event, brown pixels did not spike

## 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 $128 \times 128$-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 who required visual input but could not afford a real 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]. Figure 8 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 exposure mechanisms.



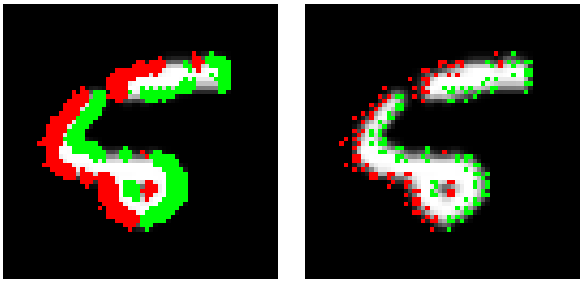
(a) Recording from DVS emulator.

(b) Recording from silicon retina [4].

Fig. 8: Visual comparison of an MNIST digit traversing a computer screen horizontally.

The emulator can 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 handwritten digits we demonstrate the emulator's behaviours.

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 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 between frames, 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 [17], 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 the receiver can recover from a loss of spikes is shown in Figure 10; of particular interest are the $sender$ and $receiver$ rows, which show their reference frames (i.e. what both "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), the receiver cannot reconstruct the picture correctly. After 40 waves of spikes (rightmost column), the absolute difference ($|send - recv|$ row) between the reference images only has PERCENT non-zero 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 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.

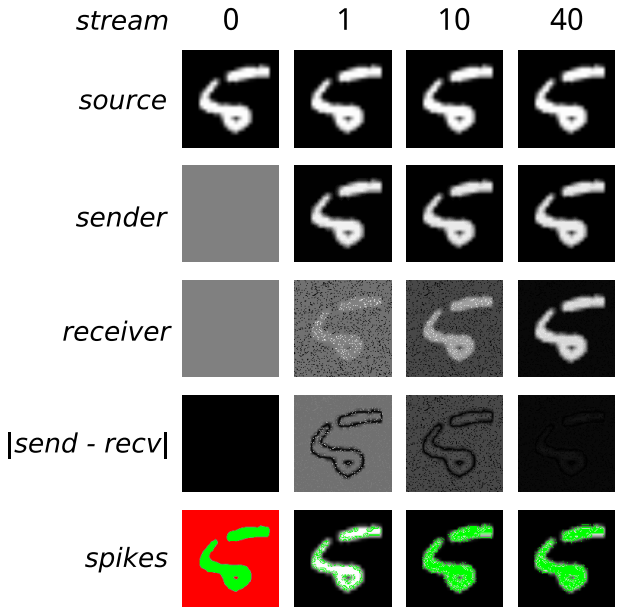



Fig. 10: History decay helps to remove transmission errors (1 spike-per-pixel with linear time encoding).

## VI. Conclusion

An important contribution of 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.

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 to send more information per spike and lowers bandwidth requirements. Time-coded spikes 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 using the parallel processing nature of Graphics Processing Units it was possible to encode images at higher resolutions (we have tested up to 1080p video at 30 FPS); 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 [12]. 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 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] 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.

[11] 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.

[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, "Playstation eye — wikipedia, the free encyclopedia," 2016, [Online; accessed 4-July-2016]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PlayStation_Eye&oldid=726565219

[17] 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.