Active Perception With Dynamic Vision Sensors. Minimum Saccades With Optimum Recognition

Amirreza Yousefzadeh, Garrick Orchard, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco

***Abstract*—Vision processing with dynamic vision sensors (DVSs) is becoming increasingly popular. This type of a bio-inspired vision sensor does not record static images. The DVS pixel activity relies on the changes in light intensity. In this paper, we introduce a platform for the object recognition with a DVS in which the sensor is installed on a moving pan-tilt unit in a closed loop with a recognition neural network. This neural network is trained to recognize objects observed by a DVS, while the pan-tilt unit is moved to emulate micro-saccades. We show that performing more saccades in different directions can result in having more information about the object, and therefore, more accurate object recognition is possible. However, in high-performance and low-latency platforms, performing additional saccades adds latency and power consumption. Here, we show that the number of saccades can be reduced while keeping the same recognition accuracy by performing intelligent saccadic movements, in a closed action-perception smart loop. We propose an algorithm for smart saccadic movement decisions that can reduce the number of necessary saccades to half, on average, for a predefined accuracy on the N-MNIST dataset. Additionally, we show that by replacing this control algorithm with an artificial neural network that learns to control the saccades, we can also reduce to half the average number of saccades needed for the N-MNIST recognition.**

***Index Terms*—Artificial neural networks, convolutional neural networks, machine vision, neural network hardware, object recognition, robot vision systems, spiking neural networks.**

I. Introduction

INTEREST in Dynamic Vision Sensors (DVSs) has grown rapidly in recent years, while at the same time this innovative technology is becoming more accessible to researchers, highlighting such sensors' potential to enable low-latency sensing at low computational costs. However, DVSs operate differently than traditional frame-based sensors, and therefore the processing of event data requires a different approach than frame-based data.

Frame-based vision systems process frames (static images), or sequences of frames, captured by a traditional camera. Frame capture and frame processing are usually two separate steps, with the quality of the captured frame affecting the quality of the processing outcome. Acquiring high-quality frames usually involves minimizing the relative motion between the frame subject(s) and the sensor during frame exposure to eliminate motion artifacts. Conventional machine vision algorithms are designed to process each frame individually, regardless of the amount of information contained in it. This is only efficient if the rate of change in the scene is similar to the frame rate, whereas in practical applications there are sometimes moments without any movement and moments with fast motion.

DVSs are event-driven temporal contrast sensors that operate on a different principle, detecting the time and pixel location of light intensity changes in the scene. For the subject to be visible to the sensor, there has to be relative motion between sensor and subject, because without motion - and under constant lighting - there would be no pixel intensity changes for the sensor to detect.

To ensure relative motion between sensor and subject, either the sensor or the subject (or both) need to be moving. In a typical sensing scenario the motion of the subject to be sensed cannot be controlled, therefore moving the sensor is a more convenient approach. This, however, poses the question of *how* to move the sensor.

For actuated frame-based vision systems, sensor motion typically involves simply pointing the sensor towards the subject using a pan-tilt unit or by mounting it on a moving platform (such as a mobile robot). However, in biological vision - by which event-driven vision sensors are loosely inspired - there is growing evidence that the motion of the vision sensors (eyes) plays a vital role in perception, and that such movement is both well controlled (albeit subconsciously) and task-dependent [1].

Examples from nature include the jumping spider, which actively moves its retina [2], the praying mantis, which executes a peering type motion for depth perception, or pigeons, which move their heads back and forth to perceive depth. Even in humans, there is growing evidence that micro-saccades during fixation play a key role in perception [3], rather than just correcting erroneous ocular drift, as was previously believed.

Intuitively, in an action-perception loop in natural animals, it is obvious that perception influences action and that action

Manuscript received December 18, 2017; revised March 3, 2018 and April 18, 2018; accepted May 1, 2018. Date of publication June 12, 2018; date of current version August 15, 2018. This work was supported in parts by the EU H2020 under Grants 644096 ECOMODE and 687299 NEURAM3 and by the Spanish grant from the Ministry of Economy and Competitivity TEC2015-63884-C2-1-P (COGNET) (with support from the European Regional Development Fund). The work of A. Yousefzadeh was supported by an FPI scholarship from the Spanish Ministry of Economy and Competitivity. This paper was recommended by Associate Editor J. Ohta. *(Corresponding author: Bernabé Linares-Barranco.)*

A. Yousefzadeh, T. Serrano-Gotarredona, and B. Linares-Barranco are with the Instituto de Microelectrónica de Sevilla IMSE-CNM (CSIC and Universidad de Sevilla), 41092 Sevilla, Spain (e-mail: reza@imse-cnm.csic.es; terese@imse-cnm.csic.es; bernabe@imse-cnm.csic.es).

G. Orchard is with the Temasek Laboratories and Singapore Institute for Neurotechnology (SINAPSE), National University of Singapore, Singapore 119077 (e-mail: garrickorchard@nus.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBCAS.2018.2834428

influences perception. However, most works and benchmark datasets focus on how best to perceive in order to influence action, since with pre-recorded data it is not possible to influence recordings through actions.[1] This paper looks instead at how best to act to influence perception.

More specifically, this study uses the DVS, an event-driven temporal contrast sensor, to address the well-known MNIST [4] recognition task. It investigates how such a sensor should move to aid recognition in a closed action-perception loop, where the system decides what action to take next (if any) based on the sensory data it has received beforehand. We also look at whether knowledge of the action taken can be used to improve accuracy in the recognition task. For this, we mounted a DVS camera on a pan-tilt unit looking at static images. In a real-world application, for a example a moving robotic platform, the DVS should have saccadic movements that are much faster than the motion of the platform.

The MNIST dataset is well known in computer vision as a relatively easy means of testing recognition models. It contains 70,000 labeled pictures of handwritten digits, 60,000 for training and 10,000 for testing. We presented static MNIST samples in front of a DVS which was mounted on a pan-tilt unit and recognized the handwritten digit by analyzing the output events of the DVS after each saccade. When we used saccades to imitate biological eye movements and object recognition in the proposed network, an interesting question arose: how is the recognition task affected by saccade direction and how many saccades are needed to recognize an object? As expected, we noticed that each saccade can contain unique information about the object related to the direction and speed of the saccadic motion.

Since performing each saccade needs a mechanical movement plus event processing, it is desirable to reduce the number of saccades while keeping the same recognition performance. After several experiments, we designed an algorithm that can suggest the direction of the next saccade based on the current information about the object. Our results show that smartly chosen saccades can reduce the average number of saccades to half in comparison to random saccades with similar recognition accuracy. Interestingly, we noticed that a neural network can perform this task and intelligently suggest saccades with almost the same performance as an analytically developed algorithm.

Section II explains how a DVS works and previous approaches to use the MNIST dataset with these type of sensors (or simulated sensors).

In Section III we explain our proposed approach for event-driven processing and object recognition by using a DVS, the proposed algorithm for prediction of an efficient subsequent saccadic direction for better object recognition, and how a neural network can be trained to intelligently suggest a next saccade direction.

The results of the experiments are given in Section IV. In this Section, we introduce our hardware-software platform for real-time object recognition with DVS saccades to demonstrate the performance of the proposed algorithm in practice. Finally, brief conclusions are provided.

II. Background

This Section provides background information on DVSs and previous experiences with event-driven MNIST datasets.

A. Dynamic Vision Sensors

DVSs output data in the Address Event Representation (AER) format [8], where each event consists of a pixel address and a single bit. The single bit indicates whether the intensity change was positive or negative. The concept is illustrated in Fig. 1. As shown in Fig. 1(a), when logarithmic change of light intensity passes a threshold θ, an event will be generated. Fig. 1(c) shows the reconstruction of DVS events for time interval during which a propeller is rotating in front of the DVS. Reconstruction is done with the jAER software [7] where black dots indicate events with negative polarity and white dots indicate events with positive polarity.

In a DVS each pixel spikes asynchronously as soon as it detects a given change in light log-intensity. These sensors can outperform frame-based vision sensors in terms of data compression, dynamic range, temporal resolution and power efficiency. Several different event-driven temporal contrast vision sensors exist [5], [9]–[16]. For example, Samsung [17] recently presented a 640×480 pixel DVS which consumes a total of 27 mW at a data rate of 100 keps and 50 mW at 300 Meps, has better temporal resolution than a 2000 fps camera and a dynamic range of more than 80 dB. Gou *et al.* recently presented the highest resolution DVS (with on-demand image acquisition) with 768×640 pixels [18].

When something moves in front of a DVS, multiple pixels almost simultaneously generate events that evoke synchrony-based neural coding [3]. In this kind of coding, information is not spike rate coded or spike rank/order coded [19]. Although neurons spike asynchronously, information is coded not in the exact time of each individual spike but in the simultaneous firing of a group of spikes together.

In this work have employed two different event-driven sensors. First, to compare accuracy with other published studies, a pre-recorded dataset (N-MNIST) was used [20] which was recorded using the Asynchronous Time-based Image Sensor (ATIS) [12]. Secondly, the IMSE-DVS [5] was used to demonstrate the approach in a closed loop system in real-time.

B. Previous Approaches to Event-Driven MNIST

MNIST is arguably the most popular dataset used thus far for event-driven vision, and some different models have been applied to different event-driven variants of the original MNIST dataset. Three main event-driven versions of the MNIST dataset have been reported. The first is a recording of a subset of MNIST with different moving digits of different sizes presented to a DVS [21]. The second approach is to convert frames to spikes by means of intensity to delay conversion [22] or Poisson distri-

[1] However, in this work we used pre-recorded data to compare our results with other works, but from a different perspective. We tried to keep the performance while using less information from the dataset, as will be explained later.

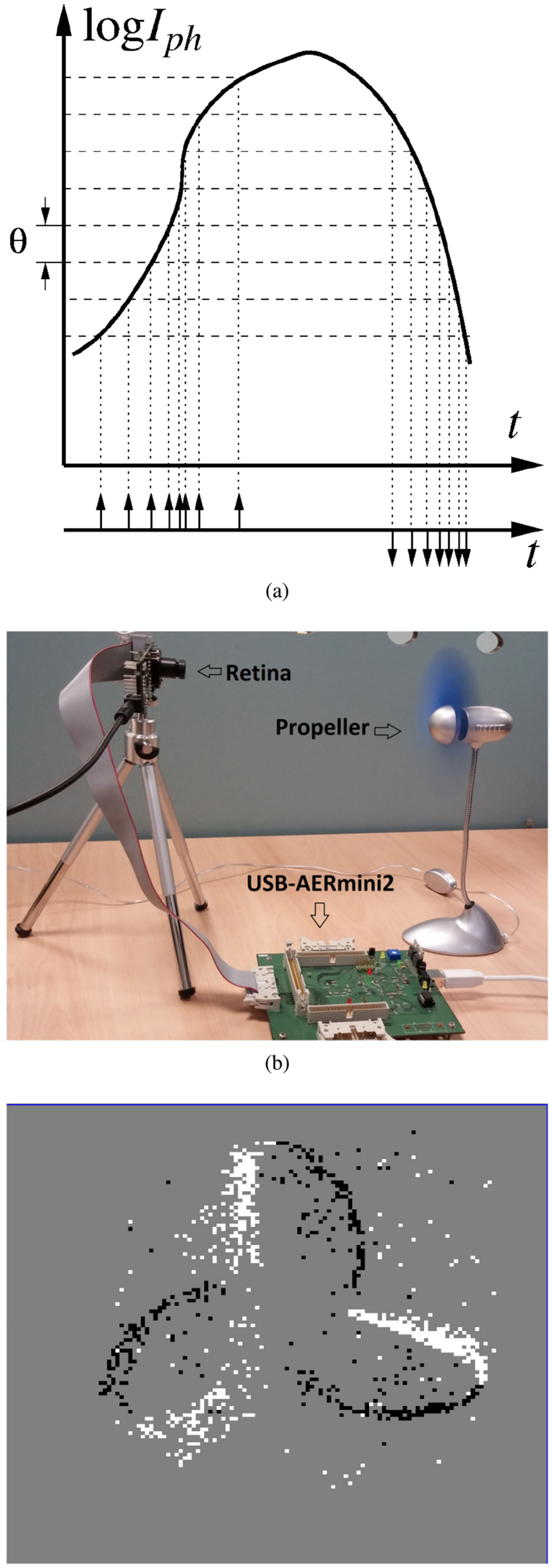


Fig. 1. (a) Event generation for a DVS pixel when changing in light intensity passes a pre-defined threshold θ. For a positive/negative change, a positive/negative event will be generated. (b) Propeller rotating in front of a DVS [5]; the USB-AERmini2 board [6] time-stamps events from the DVS and sends them to a computer through USB. (c) Reconstructed frame from DVS output with jAER software [7]. It contains 864 events collected during 624 μs (1.3 Meps - million events per second).

butions [23], [24]. The most recent approach is a full conversion of the MNIST dataset at the original pixel scale, generated by moving the sensor with fast saccades while viewing static digits [20] (dubbed N-MNIST). This dataset is captured by mounting the ATIS sensor [12] on a motorized pan-tilt unit and having the sensor move while it views the MNIST samples on an LCD monitor.

New techniques for efficient event processing are gradually being introduced. HOTS [25] is a new hierarchical machine learning technique that extracts visual features from events. HFirst [26] is a hierarchical Spiking Neural Network (SNN) for object recognition which uses a simple feed forward learning mechanism. This network has been implemented in low power parallel platforms such as FPGAs and SpiNNaker [27] and achieved 71.15% accuracy on the N-MNIST dataset [20] without optimization.

Synaptic Kernel Inverse Method (SKIM) [28], a new learning method for synthesizing SNNs, achieved 92.87% accuracy on the N-MNIST dataset. Spike Time Dependent Plasticity (STDP) is an unsupervised bio-inspired learning method for SNNs. Kheradpishe *et al.* [22] developed a multi-layer SNN equipped with STDP, achieving 98.4% accuracy on the MNIST dataset by converting all the MNIST frames to events through intensity to delay conversion. J.H. Lee *et al.* [24] developed a new method to adapt the famous error backpropagation technique for SNNs, achieving 98.66% accuracy on the N-MNIST dataset.

N-MNIST contains three saccade recordings for each MNIST sample. Based on our knowledge, no research has been done to improve the performance of recognition by considering each of these saccades as an individual source of information which is coupled to the direction of the saccade. We have used the N-MNIST dataset to benchmark performance of our proposed method but in an entirely different perspective. While it makes sense to use all the three saccades from each sample to improve recognition accuracy, in real-time robotic applications each additional saccade comes with a cost in power consumption and recognition latency. Therefore when we used N-MNIST, we considered the cost of each saccade along with the recognition accuracy and tried to use fewer saccades to recognize the handwritten digits.

III. Event-Driven Recognition With Saccades

In this Section, first, we explain the methods that we used for processing saccadic events in a neural network. This neural network is an efficient feed forward network that receives DVS events and processes them to recognize handwritten digits. The output of this network is a prediction vector that contains ten values (one for each digit). Later on, in Section III-B, we integrate this network into a closed-loop system along with a block to control the direction of saccadic motion. This block, which we call the "Next Saccade Prediction" (NSP) block tries to suggest an optimal direction for the subsequent saccade based on the current output of the feed forward handwritten digit recognition network. The NSP block executes an analytical algorithm in software to suggest the next saccade direction. In

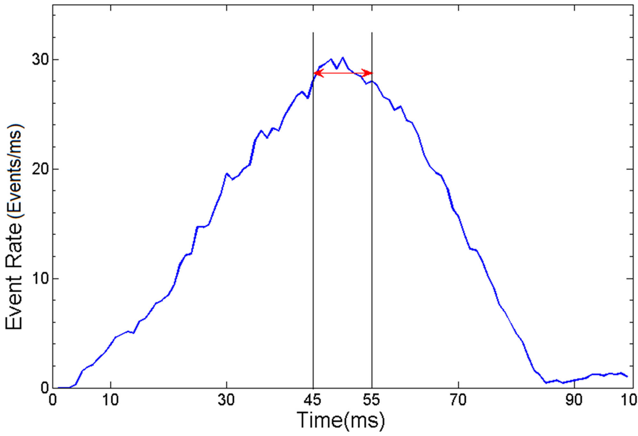


Fig. 2. Average event rate of a saccade (including all 28 × 28 pixels) in the N-MNIST dataset per millisecond. A frame is constructed by integrating the events in the time span of +/−5 ms (between 45 ms to 55 ms) around the peak average event rate at 50 ms.

Section III-C we show how the NSP block can be replaced by a neural network. In this case, both the feed forward and feedback processing will be done by using neural networks. This allows us to easily implement the system fully in hardware.

A. Feed Forward Symbol Recognition Neural Network

As mentioned earlier, a DVS is power efficient and has considerably less latency than conventional frame-based sensors. However, it is generally harder to extract information from the DVS events by using conventional image processing methods. To extract information efficiently, we propose processing groups of events that are generated close together in time rather than processing individual events. This is not a new idea and has been used in efficient hardware implementations [29], [30]. There is evidence that this kind of processing also takes place in biological cortex [31]. A block which is called "frame-maker" was therefore designed to group events occurring close together in time into a packet (equivalent to a frame in conventional image processing).[2] By using such a "frame-maker" after DVS sensing, it was possible to create an automatic adaptive frame-rate camera for our system input.

To build a frame from each saccade, one approach is to put a fixed number of events in a frame [30]. This method may result in multiple frames for a saccade. Also, each stimulus may need a variable number of events to construct a precise frame. For example, digit '8' is bulkier than digit '1' and will require more events to have a clear frame.

In the N-MNIST dataset [20], each saccade takes about 100 ms. A saccadic movement has the highest velocity in the middle of the saccade. Therefore, the output event reaches a maximum around 50 ms after the start of a saccade. Fig. 2 shows the average event rate of one saccade in the N-MNIST dataset. Our experiments show that collecting events during a short time when the event rate is high will result in a sharp frame. As illustrated in Fig. 2, a frame for each N-MNIST saccade can be created by collecting all the events which are generated in a time span of 10ms around the center of a saccade. The events outside this time will not be processed.[3] Fig. 3 shows three frames that are generated by three saccades of a sample in the N-MNIST dataset. Directions of these three saccades are shown in Fig. 4, and are labeled SAC_H, SAC_DU, SAC_DD. Throughout the paper we will use the same labels to refer to different "concepts": for example, in Fig. 3 use them for labeling "frames", each corresponding to one of the three saccades; later on we will use them to label three "memories" or three "flag bits", always each corresponding to one of the three saccades.

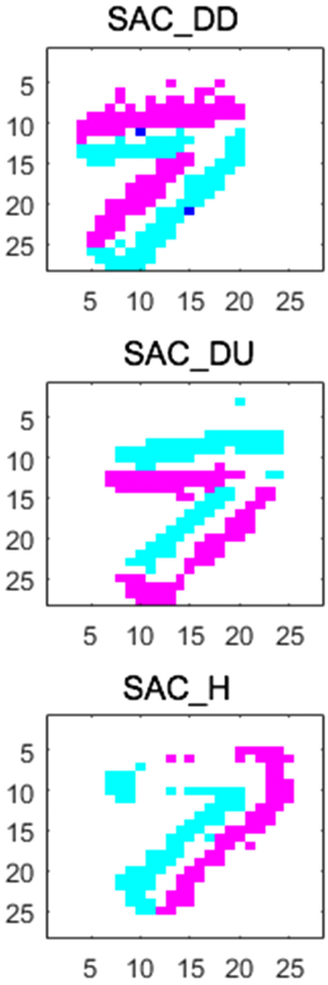


Fig. 3. Three saccades captured from sample '80' of test set in N-MNIST dataset. The colors show the polarity of the events. Blue is for negative events and purple is for positive events. Dark blue indicates places where both positive and negative events occurred.

Fig. 5 shows the block diagram of the system that has been used to perform symbol recognition with a DVS through saccadic movements. The output of the "frame-maker" block is

[2] We report elsewhere [32] that the delay of processing such frames in hardware is in the order of 20 μs. Consequently, the latency of processing the frame once the events are available is negligible with respect to the time of collecting them (10 ms).

[3] This optimum 10 ms interval is dependent on the mechanical saccadic movement used for these experiments. Speed and displacement have been kept fixed, only changing the angle. For different speeds and displacements, one may need to change this optimum 10 ms slice.

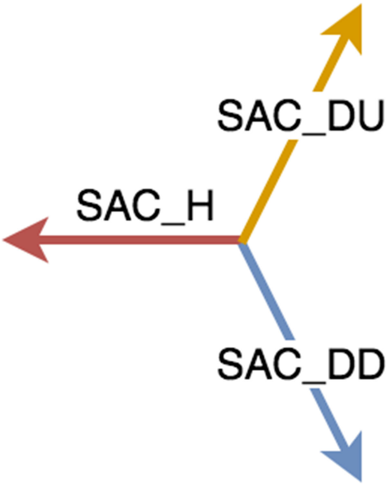


Fig. 4. Direction of saccades in the N-MNIST dataset, SAC_DD (Diagonal Down Saccade), SAC_DU (Diagonal Up Saccade) and SAC_H (Horizontal Saccade).

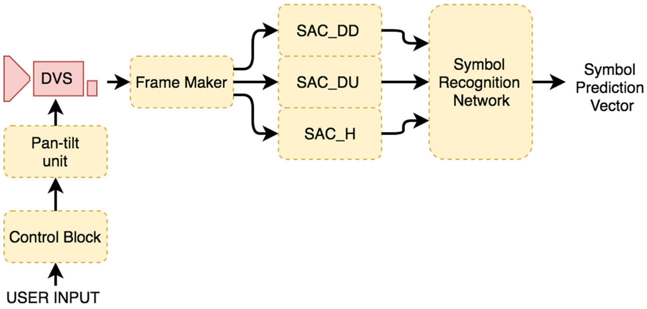


Fig. 5. Block diagram of proposed feed forward system for symbol recognition with saccades, using a DVS. The DVS is connected to a moving pan-tilt unit. The "frame-maker" block assembles a frame after each saccade and then stores that frame in its corresponding memory (SAC_DD, SAC_DU or SAC_H). Here we limit the direction of movements to three to remain compatible with the N-MNIST dataset. The Control Block is responsible for controlling the direction and speed of the pan-tilt unit movements based on the user input.

a 28×28 pixel binary frame,[4] if polarity information is not used; otherwise the output frame is $28 \times 28 \times 2$ bits. After the "frame-maker" block, a conventional Artificial Neural Network (ANN) is implemented to process the frames and recognize the handwritten digits. This ANN is called "Symbol Recognition Neural Network" (SRNN) and receives a hyper-frame as its input. The hyper-frame is a frame with 28×84 ($28 \times 28 \times 3$) pixels (assuming polarity bit is ignored) and can contain all the frames made by the three saccades of each sample. This network should be able to work with one, two or three saccades of each input sample, as illustrated in Fig. 6. Therefore, during the training phase, SRNN was trained to accommodate all possibilities. This means that the SRNN can use one, two or all three saccades of a sample to predict the presented digit.[5] We

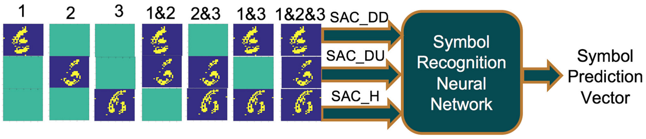


Fig. 6. "Symbol Recognition Neural Network" (SRNN) with one, two and three saccades for a sample of N-MNIST dataset. Each column represents a hyperframe with three frames, each for one saccade direction. When a saccade direction is not available, its frame is left blank.

used a blank input in the position of the non-available frames to construct the 28×84 pixels of a hyper-frame. The SRNN can have an arbitrary number of layers with various architectures (convolutional, fully connected, etc.). In this work, we tried a few small but accurate enough neural networks to perform our experiments.

For a given input sample i, we interpret the SRNN output $\hat{Y}_i = (\hat{y}_{i0}, \hat{y}_{i2}, \ldots \hat{y}_{ij}, \ldots, \hat{y}_{i9})$ as the vector of prediction probabilities $\hat{y}_{ij}$ for each class j. The recognized digit is the one with the highest prediction value. The sum of all the values in a prediction vector is normalized to one. Therefore, each value can be interpreted as a probability. The quality of the prediction vector can be measured by calculating the following "prediction loss function"

$$\mathcal{L}_i = \frac{||\hat{Y}_i - Y_i||}{2} \tag{1}$$

where $\mathcal{L}_i$ is the loss for sample i, $\hat{Y}_i$ is called the prediction vector for sample i and Y_i is the ground truth label for sample i represented using one hot encoding ($y_{ij} = 1$ if the class is j, and zero otherwise).

B. Closed Loop Recognition With Analytical Algorithm

In biological vision, it is the movement of the retina (saccades and micro-saccades) that enables one to see clearly [3]. In our study, we used a standard pan-tilt platform to move the DVS while the object in front of it was fixed. The information obtained from saccades is determined by the movement parameters. A movement can be described in terms of its velocity, distance, and direction. The movement velocity can affect the rate at which events are generated. For a clear saccade to be captured, the movement has to be sufficiently fast over a short distance. The recognition task can also be affected by the direction of the movement. For example, horizontal saccades intensify vertical edges but suppress horizontal ones. This also influences the relative positions of positive and negative events, leading to different perceptions of the same object (see Fig. 3).

Intuitively, two strong enough saccades of a DVS with different directions should be sufficient to retrieve all the information from a two-dimensional picture. For objects without any prominent edges parallel to the direction of the saccade, just one saccade could be enough for recognition. An extra saccade increases power consumption and delay in recognition, but it may also provide additional information about the object. In real applications, a robot can choose to perform an extra saccade or

[4] There is only one bit for each pixel in this frame, so multiple events with the same address will not carry additional information. To save power, it is recommended to adjust the DVS parameters (like threshold, refractory period, ...) and pan-tilt unit parameters (like the velocity and range of movement) in a way that each pixel generates maximum one event for each frame.

[5] Recognition accuracy increases (on average) when more saccades are provided.

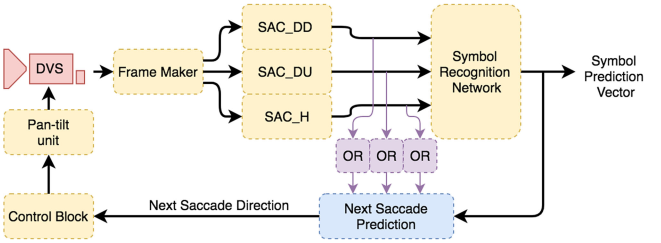


Fig. 7. Block diagram of the closed-loop recognition system. The NSP block calculates the best direction for the next saccade, if it is needed.

not, depending on its current knowledge of the object. The same decision can be made regarding the direction of the saccade.

As shown later in Section IV, we noticed that more than 94% of the test samples in the N-MNIST dataset could be recognized correctly with only one saccade. Therefore, performing an extra saccade for those samples represented a waste of time and power. In this Section, we look at how we can predict the need for an additional saccade and the best direction for it. We added another block to Fig. 5, called NSP. This block closes the loop of our system, as shown in Fig. 7.

Fig. 7 shows the inputs and output of the NSP block. One of the inputs is $\hat{Y}_i$ which is generated by processing one or more saccades. The other three inputs to the NSP block indicate which saccades have contributed to recognition. This information is important to avoid suggesting an already performed saccade again. An 'OR' logic can determine which memory block contains non-zero pixels. Memory blocks which correspond to the saccades that have not been executed contain all zero values.

The output of the NSP block determines the next saccade direction. The next saccade direction can be one of these four possibilities (see Fig. 7):

1) No extra saccade
2) SAC_DD
3) SAC_DU
4) SAC_H

The NSP block is implemented in a closed loop with the SRNN to perform the following tasks in the order shown:

1) Receive events from the first saccade and make the first guess about the object
2) Predict the best direction for the next saccade, if necessary
3) Command the pan-tilt unit to perform the next saccade
4) Combine information from all the saccades performed so far to improve recognition accuracy
5) Continue from step 2)

Our experimental results showed that if the SRNN is not sure about the recognition results, the NSP block should request extra saccades. To quantify the amount of uncertainty in a prediction vector, we used the definition of entropy in information theory

$$\mathcal{H}_i = -\sum_j \hat{y}_{ij} log_2(\hat{y}_{ij}) \tag{2}$$

Fig. 8 illustrates the relationship between entropy and prediction loss (see (1)) for the test samples in the N-MNIST dataset for a specific SRNN. It shows that having a small entropy cannot guarantee a correct recognition. Sometimes, it is possible that the SRNN can be very confident but the answer is wrong. Our experiments show that in these cases, performing extra saccades cannot help to find the correct answer and the only solution is to improve the SRNN.[6] High entropy in a prediction vector means less certainty about the results and this is the time when performing extra saccades might be helpful.[7]

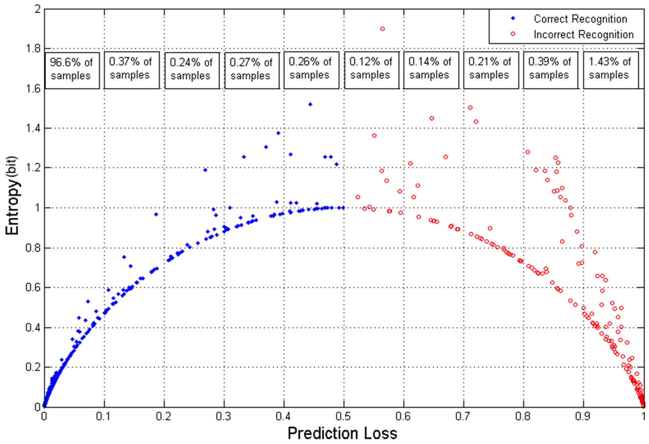


Fig. 8. Relationship between entropy and prediction loss for the 10,000 test samples of the N-MNIST dataset after presentation of all the three saccades. A two layer SRNN has been used. A sample is classified as correctly recognized when the position of the maximum value in its prediction vector correctly shows the class of the presented digit. For each 0.1 interval in the x-axis, we indicate the percent of test samples within this interval (see boxes on the top part of the figure).

The NSP block can decide to ask for an extra saccade when the entropy is higher than a predefined threshold θ_H. This θ_H is a user-defined parameter and depends on the cost of an additional saccade. Obviously, choosing a smaller θ_H will result in performing more saccades on average and increasing the average recognition accuracy. In Section IV the relationship between θ_H, the average number of saccades and the recognition accuracy for our experiments will be explained in detail.

If the NSP block asks for an extra saccade, another mechanism will also be needed to define the best choice among different saccade directions. For this purpose, we decided to extract some statistics from the training set of the N-MNIST dataset.

Before performing any saccade, our system does not have any information about the presented object. In this case, the NSP block chooses the saccade that shows the best average performance among all the three saccades during training. For each of the possible next saccade directions, we have defined a vector which is called "Confidence Coefficient Vector" (CCV).

To explain how to calculate the CCVs, suppose that the first saccade is SAC1. The possible next saccades are SAC2 and SAC3. The CCV for SAC2 and SAC3 is a 10-element vector

[6] In this work we do not intend to improve the capability of the SRNN with novel techniques, rather we would like to use the available network as efficiently as possible.

[7] For example, when the prediction vector is [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] entropy is zero, while it is maximum (3.32) when the prediction vector is [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]. In the first case, the network is confident that the presented sample is digit '0' while in the second case, the network is not sure about any of the classes.

which is computed using the following equations:

$$A = \{i | \mathcal{L}_i(SAC1, SAC2, Blank) < \mathcal{L}_i(SAC1, Blank, SAC3)\}$$

$$B = \{i | \mathcal{L}_i(SAC1, Blank, SAC3) < \mathcal{L}_i(SAC1, SAC2, Blank)\}$$

$$CCV(SAC1 \rightarrow SAC2) = mean(\hat{Y}_i(SAC1)) \text{ for all } i \in A$$

$$CCV(SAC1 \rightarrow SAC3) = mean(\hat{Y}_i(SAC1)) \text{ for all } i \in B$$

where:

1) i is the sample number
2) $\hat{Y}_i(SAC1)$ is the prediction vector for sample i after presentation of SAC1
3) $CCV(SAC1 \rightarrow SAC2)$ is the Confidence Coefficient Vector when SAC2 is performed after SAC1
4) $CCV(SAC1 \rightarrow SAC3)$ is the Confidence Coefficient Vector when SAC3 is performed after SAC1
5) $\mathcal{L}_i(SAC1, SAC2, SAC3)$ is the prediction loss function in (1) when SAC1, SAC2 and SAC3 is performed. If one of the inputs is $Blank$, it means the prediction vector from which the $\mathcal{L}_i$ is calculated, did not have information about that specific saccade.

In other words, the CCV is the average of the prediction vectors for the training set samples with the minimum prediction loss for a pre-defined next saccade. CCVs are calculated once, after training the SRNN. This method can easily be extended for more than three saccades.

The NSP block chooses the next saccade which has the most similar CCV to the current prediction vector. In this way, we hope that the upcoming saccade will be in a way that is best for the existing guess of the SRNN. The similarity of two vectors can be calculated by measuring their Euclidean distance. This method, even though it is not a very precise method for choosing the next saccade direction, is nevertheless quite simple.

The algorithm for the NSP block can be summarized as follows:

1) The first saccade is always the one that shows the best average results for the training samples.
2) Once the entropy (see eq. (2)) of the prediction vector of the first saccade has been calculated, it is compared with θ_H to see whether an extra saccade is needed or not.
3) If an extra saccade is needed, the best saccade can be found using the CCVs
4) Calculate the entropy again and compare it with θ_H to see whether an extra third saccade is needed or not.
5) Continue from step 3)

C. Closed-Loop Recognition With a Neural Network

Next, we investigated the use of an ANN to predict which saccade should be performed next. In this case, both feed forward and feedback paths will be equipped with neural networks which reduce the complexity of the system. Therefore we replaced the proposed algorithm by a neural network inside the NSP block in Fig. 7. Note that using a Neural Network to predict the next saccade simplifies the implementation complexity (in hardware) and scales well as datasets become more complex. An algorithmic implementation is not so obvious to implement in hardware and may need significant changes when complicating the dataset.

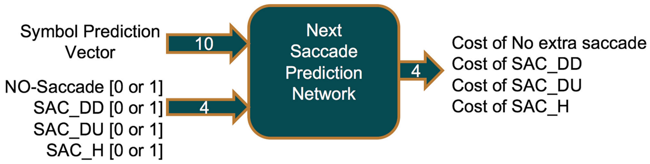


Fig. 9. Inputs and outputs of “Next Saccade Prediction Network” (NSPN).

Fig. 9 shows the inputs and outputs of the “Next Saccade Prediction Network” (NSPN). Inputs and outputs of this network are highly compatible with the previously presented NSP block. The “Symbol prediction vector” is the output of the SRNN. The other four inputs indicate the currently executed and available saccades. Previously, in the NSP block we only used three inputs for providing this information, which was enough. However, during training, we noticed that the neural network can learn better if the inputs are normalized. We wanted to train the network to predict the best saccade direction for the initial movement as well, so we decided to add an active input for this case which is called “NO-Saccade”. The “NO-Saccade” input is equal to the ‘NOR’ of the other three inputs (SAC_DD, SAC_DU, SAC_H).

The NSPN outputs are four values which represent a “cost” for each action. In the current implementation, the pan-tilt unit will move in the direction with minimum predicted cost (Hard-Threshold).[8] For example, when the value of output “Cost of No extra saccade” is the minimum, it means the NSPN is suggesting not to do any further saccade, because the current information about the object might be enough.

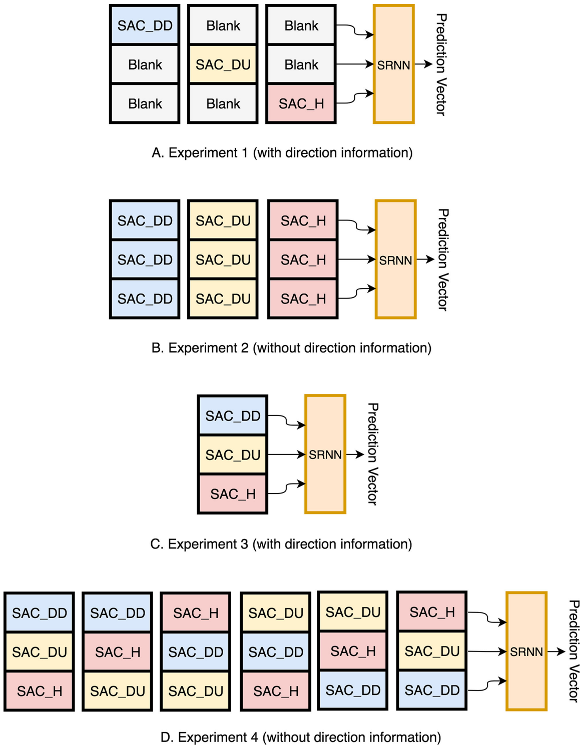
To train this network, we calculated a “cost” for each action for all the possible combinations of inputs and used it for supervised learning. To calculate this “cost”, first, we determined the power and latency cost of an additional saccade. This value, which we call “saccade cost” (or “mechanical cost”) S_C, is equivalent to the θ_H in the previously introduced NSP block. When this value is high, the NSPN is more likely to suggest no extra saccade for the next action. Here, we used the same S_C for all three saccades. Another critical parameter to determine the cost of each action is how much this action will help to reduce the “prediction loss” (see eq. (1)).

We define the “cost” of each action as the sum of the “prediction loss” value (which is calculated after performing the action) and the S_C

$$Cost_i = S_C + \mathcal{L}_i \tag{3}$$

For the “No extra saccade”, S_C value is zero. Otherwise, it is a predefined constant value. From the four possible next actions, the system picks the one whose output predicts the minimum cost. Our goal is that the system learns to reduce the total

[8] Another suggestion is not to restrict the DVS to move in the direction of one of these three saccades, but to let it move in a mixture of directions based on the cost of each saccade (Soft-Threshold).



A. Experiment 1 (with direction information)

B. Experiment 2 (without direction information)

C. Experiment 3 (with direction information)

D. Experiment 4 (without direction information)

Fig. 10. Different experimental configurations for testing whether feeding direction information can be helpful for learning.

TABLE I
ACCURACY OF EXPERIMENTS IN FIG. 10 FOR DIFFERENT NETWORK SIZES, USING ONLY POSITIVE EVENTS, AFTER '3' EPOCHS

	1-Layer	2-Layer	3-Layer
Experiment 1	91.9%	95.5%	97.7%
Experiment 2	70.5%	83.9%	89.2%
Experiment 3	95.0%	96.9%	97.2%
Experiment 4	82.1%	95.0%	97.3%

TABLE II
ACCURACY OF EXPERIMENTS IN FIG. 10 FOR DIFFERENT NETWORK SIZES, USING ONLY POSITIVE EVENTS, AFTER '50' EPOCHS

	1-Layer	2-Layer	3-Layer
Experiment 1	92.6%	97.3%	98.3%
Experiment 2	85.2%	89.4%	97.9%
Experiment 3	95.4%	97.2%	98.6%
Experiment 4	89.4%	96.4%	98.1%

number of saccades required to achieve the same recognition performance.

IV. EXPERIMENTAL SETUPS AND IMPLEMENTATION RESULTS

Section IV-A describes the results of our experiments for saccade-based recognition with the DVS moving in a predefined direction. Later on, we describe the results for closed loop recognition, when DVS movement was controlled by the NSP block (Section IV-B) and the NSPN (Section IV-C). For all Neural Network blocks we used Tensorflow python library on GPUs for off-line training. The systems were then implemented as real-time interactive platforms with the DVS on a pan-tilt unit, while using MATLAB for all the processing.

A. *Feed Forward Recognition With Saccades*

This Section reports only the test results for the open loop system in Fig. 5 when the DVS moves in a predefined direction (i.e., no saccade prediction). For these experiments we used the pre-recorded N-MNIST dataset [20].

Our goal here is to study under what condition it is relevant to provide specific information about direction of provided saccades to improve recognition. To do so, we designed the experiments shown in Fig. 10. In experiments '1' and '3', input hyper-frames are built by allocating saccades in the same positions, while for experiments '2' and '4', saccade positions are intentionally shuffled. Let us call these saccade positions "channels".

To see the effect of network size, three different network sizes for the SRNN were implemented: a 3-Layer network (3C5x5-128FC-10FC), a 2-Layer network (3C5x5-10FC) and a 1-layer network (10FC).[9]

The spikes' polarity bits reveal information about the movement direction. To find out the effect of using spikes' polarity, we have done all the experiments with and without using spikes' polarity. Each experiment was carried out once using only the positive polarity events and once using both polarity events. When only the positive polarity was used, input size was 28 × 28 × 3, while when using both polarities input size was 28 × 28 × 3 × 2. In the second case, the network size was therefore larger.

By comparing experiments '1' and '2' in Fig. 10 we wanted to find out if it helps or not to feed all saccades through the same channel. To have a fair comparison, we always used the same input size, but the hyper-frame arrangements are different. In this case, accuracy is calculated by averaging the prediction vectors of all three saccades.

By comparing experiments '3' and '4' in Fig. 10 we wanted to find out the effect of feeding explicit information about direction when all three saccades are available. In experiment '3' a hyper-frame contains only one arrangement of saccades (SAC_DD/SAC_DU/SAC_H) while in experiment '4' all six possible shufflings are provided.

We also tested the speed of learning. We report the accuracy of the networks after 3 training epochs and also after 50. It should be noted that the number of training samples in each epoch in the different experiments was not equal. While there are 60,000 training samples in the N-MNIST dataset, in experiments '1' and '2' we had 180,000 (60,000 × 3), in experiment '3' we had 60,000 and in experiment '4' we had 360,000 (60,000 × 6) hyper-frames in each epoch.

Tables I, II, III and IV show the results obtained in all the Fig. 10 experiments. Based on the results we concluded the following:

[9] FC indicates a Fully Connected layer while C indicates a Convolutional layer. For example, 3C5x5 means a convolutional layer with three feature maps and kernel size of 5 × 5 and 10FC means a fully connected layer of 10 neurons.

TABLE III
ACCURACY OF EXPERIMENTS IN FIG. 10 FOR DIFFERENT NETWORK SIZES, USING BOTH POSITIVE AND NEGATIVE EVENTS, AFTER '3' EPOCHS

	1-Layer	2-Layer	3-Layer
Experiment 1	94.8%	97.4%	97.2%
Experiment 2	74.7%	84.2%	95.1%
Experiment 3	96.3%	95.8%	97.8%
Experiment 4	80.1%	96.6%	97.6%

TABLE IV
ACCURACY OF EXPERIMENTS IN FIG. 10 FOR DIFFERENT NETWORK SIZES, USING BOTH POSITIVE AND NEGATIVE EVENTS, AFTER '50' EPOCHS

	1-Layer	2-Layer	3-Layer
Experiment 1	95.4%	97.8%	98.6%
Experiment 2	89.0%	95.2%	98.6%
Experiment 3	96.5%	98.0%	98.8%
Experiment 4	89.8%	97.0%	98.1%

TABLE V
ACCURACY DIFFERENCE BETWEEN '50' AND '3' TRAINING EPOCHS, FOR EXPERIMENTS '1' AND '2' IN FIG. 10, WHEN USING ONLY POSITIVE EVENTS

	1-Layer	2-Layer	3-Layer
Experiment 1	0.7%	1.8%	0.6%
Experiment 2	14.7%	5.5%	8.7%

TABLE VI
DIFFERENCE OF ACCURACIES BETWEEN EXPERIMENTS '1' AND '2' AND BETWEEN EXPERIMENTS '3' AND '4' IN FIG. 10 AFTER 50 EPOCHS WHEN USING BOTH POLARITIES

	1-Layer	2-Layer	3-Layer
Experiment (1)-(2)	6.4%	2.6%	0.0%
Experiment (3)-(4)	6.7%	1.0%	0.70%

1) Feeding explicit direction information accelerates learning. This can be seen by comparing the accuracy of the networks after 3 and 50 epochs. For experiments '1' and '2' which had the same number of hyper-frames in each epoch, Table V shows the difference between accuracies after 3 and 50 epochs. It can be seen that by using explicit direction information (experiment '1'), the training process converges faster.

2) When network size is smaller, feeding explicit direction information (experiments '1' and '3') improves recognition accuracy. This indicates that larger networks with more learning capacity can extract saccade directions from each frame themselves without the need of explicit information[10]. Table VI shows the difference between accuracies of experiments '1' and '2' and between accuracies of experiments '3' and '4'. It shows that when network size is larger, the difference between accuracies drops.

3) Using polarity of spikes improves the recognition accuracy. This can be because of multiple reasons. First, the network size is larger in this case (input size is twice, which results in more synaptic connections for the input layer). Second, there is additional information about the object in the negative polarity spikes, as the edges producing positive events are not identical to the edges producing negative events. The third reason is that using both polarities at the same time contains information about the movement direction.

4) In experiments '1' and '2', the final prediction vector for each sample is the average of the prediction vectors for all three saccades. In experiments '3' and '4', the neural network receives all the saccades and mixes them. Experimental results show that, in general, averaging the prediction vectors for all three saccades is not always the best strategy and a neural network itself may find a more optimized way to mix the prediction vectors.

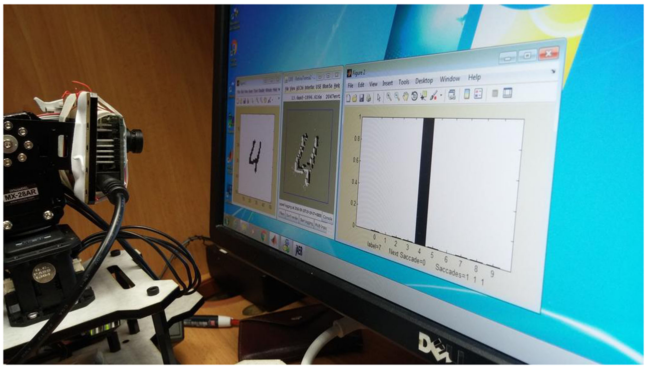


Fig. 11. Hardware setup for moving the DVS with a pan-tilt unit.

TABLE VII
ACCURACY OF TWO-LAYER(5C5X5-10FC) SRNN FOR THE DIFFERENT COMBINATIONS OF INPUT SACCADES WITH N-MNIST DATASET (ONLY POSITIVE POLARITY EVENTS ARE USED)

Input saccades	Training Accuracy	Testing Accuracy
SAC_DD	95.8%	94.4%
SAC_DU	95.5%	93.6%
SAC_H	96.6%	94.5%
SAC_DD and SAC_DU	98.4%	96.9%
SAC_DD and SAC_H	98.7%	97.1%
SAC_DU and SAC_H	98.7%	97.0%
All Saccades	99.3%	97.7%

B. Using Closed-Loop Next Saccade Prediction Algorithmic Block

Fig. 11 shows the hardware setup with the DVS mounted on a mechanical pan-tilt unit so that it can be moved in a desired direction. We have used this setup for a real-time demo [33], however, for reporting the next results, we have used the N-MNIST dataset, so that interested readers can reproduce them.

For the closed loop recognition experiments, we selected a 2-layer SRNN (5C5x5-10FC) with the configuration of experiment '3' in Fig. 10 and we used only positive polarity events. Table VII shows the average accuracies of the SRNN for each saccade combination of the N-MNIST dataset. As can be seen, adding an extra saccade always increases accuracy.

From Table VII, it can be seen that SAC_H has the best average accuracy among all three saccades of the training samples.

[10] To investigate more about this fact, we have done another experiment. In this test, we trained three neurons (each corresponds to one saccade direction) to predict the direction of a saccade. The input of each neuron was a 28 × 28 pixel frame. We found out that this network could predict the saccade direction (which the input frame is made of) with more than 98% accuracy without using spikes polarity.

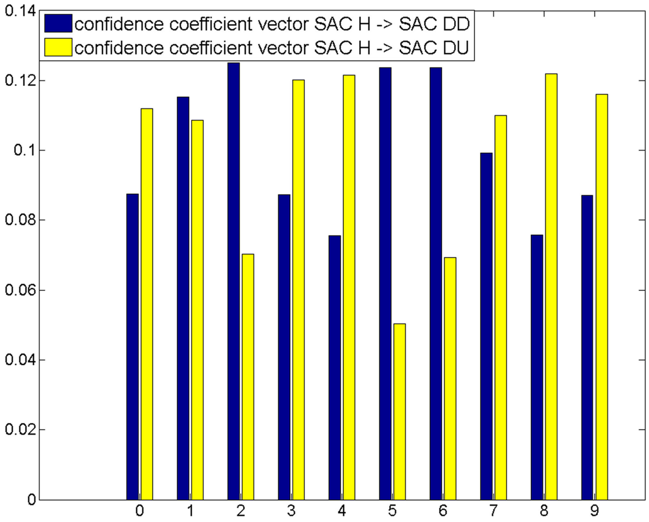


Fig. 12. Confidence coefficient vectors (CCVs) of SAC_H→SAC_DD and SAC_H→SAC_DU.

Therefore, the NSP block always choses SAC_H as the initial saccade. The next action can be "No extra saccade", "SAC_DD" or "SAC_DU".

If the entropy of the prediction vector is higher than θ_H, the NSP block should choose the "SAC_DD" or "SAC_DU" as the second saccade. In Section III-B we explained the method to extract CCVs. The confidence vector for SAC_DD after SAC_H, for example, can be calculated by averaging the prediction vectors[11] of those training samples that showed the best results when SAC_DD was performed after SAC_H. Fig. 12 shows the CCVs of SAC_H→SAC_DD and SAC_H→SAC_DU. These vectors show which saccades are better for which classes. In Fig. 12, for example, it can be seen that for digit '5' SAC_H→SAC_DD has more confidence than SAC_H→SAC_DU.

Fig. 13 shows the results of the analytical approach to saccade prediction using different θ_H. As expected (Fig. 13 A and B) the average number of saccades and the accuracy decrease by increasing θ_H. Fig. 13(C) shows the relationship between accuracy and the average number of saccades for the different θ_H. For example, if θ_H is set at '0.09' the average number of saccades will be '1.54', while accuracies for training and testing samples will be 99.24% and 97.57%, respectively. By looking at Table VII, we notice that this is almost equal to the result of the open loop recognition with three saccades per sample. This means that by using the NSP block, it is possible to reach the highest possible accuracy of the SRNN while only performing half of the number of saccades on average.

C. Using Closed-Loop Next Saccade Prediction Neural Network

This Section shows the results obtained with the closed loop network configuration with a neural network in the feedback path. In this experiment, we used the same SRNN which was used in Section IV-B and we only replaced the NSP block by a neural network (which we call "Next Saccade Prediction Network" NSPN). This network is shown in Fig. 9 and was trained using the training samples of the N-MNIST dataset[12]. We used a small but fully connected 4-layer network (50FC-50FC-50FC-4FC) for the NSPN.

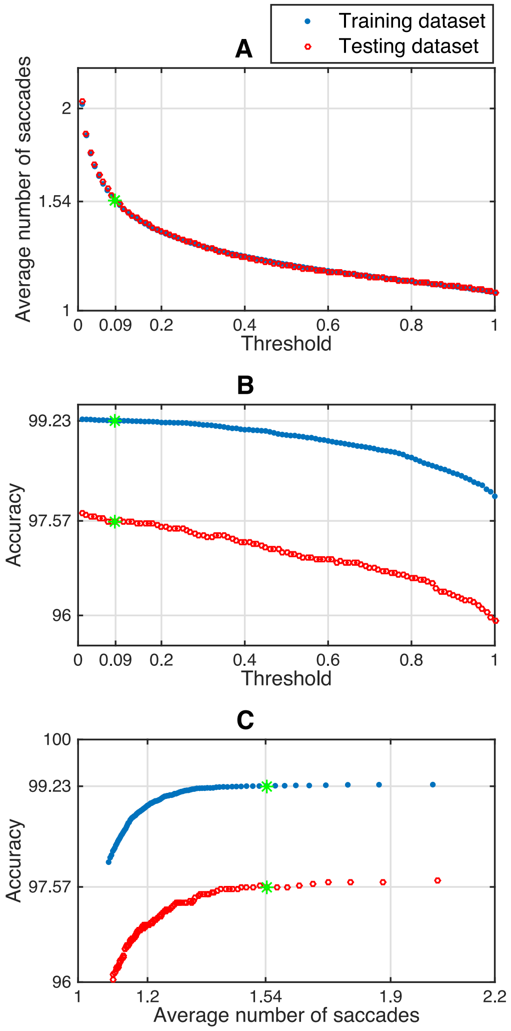


Fig. 13. Results of analytical approach to saccade prediction with different entropy thresholds (θ_H). Green marks show the accuracy and average number of saccades when θ_H is '0.09'. For this θ_H accuracy is close to the highest accuracy of our SRNN, while instead of using all 3 saccades per sample, only 1.54 saccades in average are used.

The NSPN was trained with different S_C. Fig. 14 shows the network accuracy versus the average number of saccades for different S_C. As the S_C is decreased, the network elicits more saccades and accuracy increases. With a S_C of 0.002 (when the average prediction loss is 0.05 for the training samples), for example, the system needs to perform on average 1.71 saccades per sample to achieve an accuracy of 97.6% for the testing data and 99.2% for the training data. These results are very similar to the results of the NSP block in Section IV-B. This experiment shows that if the S_C is adjusted to a reasonable value (like

[11] These prediction vectors are calculated only after presenting SAC_H.

[12] The NSPN is using the output of the SRNN. Therefore, the NSPN will be trained after the SRNN.

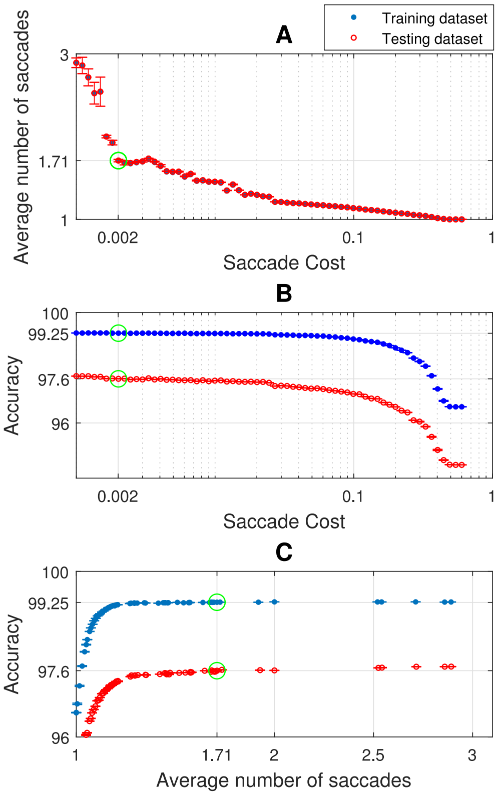


Fig. 14. Network accuracy for N-MNIST dataset and average number of saccades per sample for different S_C. The network was trained with 50 epochs for each S_C, and training was repeated 10 times. Bars indicate the spread of the average number of saccades. The reason that sometimes the plots are not monotonic is because training neural network is a stochastic process and starts from different random states. The average trend is similar to what is expected. Green marks show the accuracy and average number of saccades when S_C is 0.002, for both training and test samples. In this case, accuracy is close to the highest accuracy of SRNN while rather than using 3 saccades per sample, 1.71 saccades are used in average.

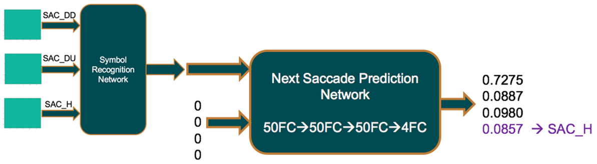


Fig. 15. The NSPN outputs for the initial saccade.

0.002), the SRNN can maintain its accuracy (see Table VII), requiring on average around half of the saccades (1.71 saccades rather than three saccades).

Next we study the features of the NSPN more carefully and provide more results. For the following experiments a S_C of 0.002 is used.

As seen in Fig. 15, the NSPN always chooses SAC_H as the initial saccade. Table VII shows that SAC_H is the best saccade, on average, for both training and testing samples.

Table VIII shows the NSPN actions after the first saccade. For more than 41% of the samples one single saccade was sufficient, and the error rate for this category was low (0.41%). This means the saccade prediction network correctly determined the samples that were easy to recognize from the first saccade. For the other samples, the network suggested an extra saccade. Since the initial saccade was SAC_H, the NSPN did not recommend SAC_H again for the second saccade, as expected.

TABLE VIII
SECOND SACCADE CHOICE FROM THE NSPN FOR TESTING SAMPLES

Saccade	Percentage of samples	Error rate
SAC_N	41.80%	0.41%
SAC_DD	20.18%	5.17%
SAC_DU	38.02%	3.85%
SAC_H	0.00%	-
Overall	100.00%	2.68%

'SAC_N' means no extra saccade was chosen. Error rate is computed after performing the second saccade (except for 'SAC_N').

TABLE IX
SACCADE CHOICE STATISTICS FROM THE SACCADE PREDICTION NETWORK FOR TESTING SAMPLES

Saccade	Percentage of samples	Error rate	Error rate of first saccade
SAC_H	41.80%	0.41%	0.41%
SAC_H→SAC_DD	15.51%	1.70%	7.28%
SAC_H→SAC_DU	36.82%	1.49%	4.22%
All SAC	5.12%	31.81%	58.07%
All SAC+	0.75%	31.81%	58.07%
Overall	100%	2.40%	5.60%

'All SAC+' indicates samples that needed more than three saccades.

After the second saccade, the network may decide that a third saccade is required for some samples. Table IX shows the percentage of test samples for each combination of saccades after all necessary saccades have been performed. For more than 41% of the samples, the system only asked for one saccade and recognized them with 99.59% accuracy. This means that recognition of these samples was an easy task for the SRNN.

For around 52% of the samples (15.51 + 36.82), the NSPN asked for one additional saccade. These samples have been recognized with 98.44% accuracy while the same samples have been recognized with 94.87% accuracy before performing the second saccade.

For around 5.12% of the samples, the NSPN asked for two additional saccades. These samples have been recognized with 76.88% accuracy while the same samples have been recognized with only 41.93% accuracy after the first saccade. These samples were hard to recognize and, as expected, the NSPN requested three saccades for them.

When recognition loss is high, the NSPN sometimes asks to perform more than three saccades, repeating one of the previous saccades. In Table IX, for 0.75% of the samples, the NSPN requested more than three saccades. The 31.81% error rate in this group of samples indicates that these samples were very difficult to recognize. In real scenarios (i.e., not a pre-recorded dataset), repeating a saccade may provide additional information.

V. CONCLUSIONS AND DISCUSSIONS

Our aim in this paper was to answer the question of how to perform saccades with a DVS to improve accuracy, speed and power consumption in a robotic platform. The first step was to mount a DVS on a motorized pan-tilt unit to perform object recognition with saccadic movements. In this step, the objective was to determine the effect of saccade direction, velocity and distance on the information captured by the DVS.

Our experimental results show that to achieve better object recognition the internal parameters of the recognition system should ideally match the saccade velocity while the distance of movement should be sufficiently short. The best saccade direction depends on the shape of the object. The first saccade can be random or move in the direction that, on average, is optimal for all cases. In our experiment, most of the objects could be recognized with the first saccade, although in some cases the system needed to perform an extra saccade to gain enough information for recognition. A proposed analytical approach and later on, an Artificial Neural Network, were used to predict the need for an extra saccade and also predict the best direction for the next saccade based on the information obtained from previous saccades. The schemes were shown to halve the number of saccades required while preserving the accuracy of the network.

When solving specific problems with a neural network, there is always the open question of how to choose the number of layers and the number of neurons per layers, or whether to use a fully connected topology or a convolution-like arrangement. In this paper we had to face this issue with two sub neural networks, the SRNN and the NSPN. During our explorations we tested many possible topologies, although we did not exhaustively try all possibilities. Consequently, we cannot claim that the proposed topologies are the optimum for our problem. Our strategy was always to look for the smallest possible networks that provided reasonable performance on the N-MNIST dataset. On the other hand, we were intentionally not looking for the optimum topology, as we wanted to see some improvements when playing the number of saccades. For the SRNN we reported and compared three different topologies, a 3-layer one, a 2-layer one, and a 1-layer one. For the first two we decided to use for the first layer a convolution processing as this, in general, helps to improve accuracy while having fewer parameters. For the NSPN, we observed that a slightly deeper network with 4 fully-connected layers gave good results while yielding a relatively small network. We observed that if we made the network deeper or wider, it would result in overfitting and performance was not improving.

Regarding datasets, in this paper we focused on the N-MNIST dataset for two reasons. The first one, because it is recorded with a spiking retina sensor, the DVS, providing a natural spiking input for our setup. The second one, because it is starting to be used by other researchers world-wide, thus providing a good reference for further comparisons. Definitely, the choices we made with respect to network topologies and parameters are somehow tuned to this specific problem of N-MNIST digit recognition. However, extending the methodology for other more complex datasets should not be too difficult. Depending on the complexity of the dataset, maybe using more saccades helps in general to retrieve more information. Additionally, the general experience is that for more complex datasets, deeper and wider neural networks are required. On the other hand, our intuition tells us that using the NSPN instead of the NSP could help when changing to more complex datasets, because adapting an algorithm might be more difficult than training a neural network.

In this paper we have focused on analyzing the impact of reducing the number of saccades when performing object recognition with a DVS camera, and using the N-MNIST dataset as benchmark. It is always interesting to compare against other methods (where saccades don’t play any role), and thus have some comparison with other techniques. It should be noticed, however, that during our work, our goal was not to obtain an optimum recognition, and thus optimize the architecture for that. Our goal was always to study under which conditions one could reduce the number of saccades. To compare with other methods, let us take as reference for this work the result shown in Table IV for the 3-layer case (3C5x5-128FC-10FC) and when presenting all 3 saccades. In this case the accuracy on the N-MNIST test set was 98.8% (or 1.2% error rate). Therefore, let us compare with respect to other works that use 3 layers and do not apply any pre-processing to the dataset (like distortions, expansion of the dataset, etc). Focusing first on other works for spiking neural networks, we can find the following in the literature. Recently, Mostafa [34] reported a 3-layer FC spiking network trained directly in the spiking domain by a clever adaptation of backpropagation, and where each neuron is allowed to fire just one spike (which would be higly beneficial if implemented in hardware). He used the original MNIST dataset, properly adapted to a 1-spike-per-neuron representation. He reports an error rate of 2.45%. Also recently, Wu *et al.* [35] have reported another smart adaptation of the backpropagation training technique for the spiking spatio-temporal domain, and they report results for both, the MNIST (converted into spikes through Bernoulli sampling from intensity to spike rate) and N-MNIST datasets. Using a 3-layer architecture, they report for the MNIST case an error rate of 1.11% and for the N-MNIST case 1.22%. In another recent work, Lee *et al.* [36] also propose a spiking domain version of backpropagation, reporting an error rate of 1.34% for the N-MNIST dataset when using a FC 3-layer MLP. Stromatias *et al.* [37] obtained 2.23% on the N-MNIST dataset for a 2-layer system, where the first layer was an untrained convolutional layer of spiking Gabor filters, and the second layer was a FC classifier trained from the spiking outputs of the first layer. For the case of non-spiking neural networks (typically referred to as ANNs - Artificial Neural Networks) benchmarking MNIST on similar architectures, we can mention the original work of LeCun [4] using the Lenet4 convolutional neural network with 1.1% error rate, or the 3-layer FC network by Hinton [38] with 1.54% error rate. A full list of reported results on the MNIST (but using any architecture and technique, except spiking ones) is maintained at [39].

Acknowledgment

The authors would like to thank T. Masquelier for helpful comments on the draft, and the SINAPSE Institute for hosting A. Yousefzadeh during his stay at Singapore.

References

[1] R. Engbert, “Microsaccades: A microcosm for research on oculomotor control, attention, and visual perception,” *Progress Brain Res.*, vol. 154, pp. 177–192, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0079612306540099

[2] A. Bruckstein, R. J. Holt, I. Katsman, and E. Rivlin, “Head movements for depth perception: Praying mantis versus pigeon,” *Auton. Robots*, vol. 18, no. 1, pp. 21–42, 2005. [Online]. Available: http://dx.doi.org/10.1023/B:AURO.0000047302.46654.e3

[3] T. Masquelier, G. Portelli, and P. Kornprobst, “Microsaccades enable efficient synchrony-based coding in the retina: A simulation study,” *Sci. Rep.*, vol. 6, pp. 1–12, 2016, doi: 10.1038/srep24086.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[5] T. Serrano-Gotarredona and B. Linares-Barranco, “A 128 × 128 1.5% contrast sensitivity 0.9% *FPN* 3 μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers,” *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.

[6] R. Serrano-Gotarredona *et al.*, “CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking,” *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[7] F.Corradi, S. Bamford, L. Longinotti, and T. Delbruck, *jAER Software*. [Online]. Available: https://sourceforge.net/projects/jaer. Accessed on: May 2018.

[8] K. A. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[9] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, “Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output,” *Proc. IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[10] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128 × 128 120 dB 15μs latency asynchronous temporal contrast vision sensor,” *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[11] J. A. Lenero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, “A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor,” *IEEE J. Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, Jun. 2011.

[12] C. Posch, D. Matolin, and R. Wohlgenannt, “A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS,” *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011.

[13] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, “A 240×180 130 dB 3μs latency global shutter spatiotemporal vision sensor,” *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.

[14] C. Li *et al.*, “Design of an RGBW color VGA rolling and global shutter dynamic and active-pixel vision sensor,” in *Proc. 2015 IEEE Int. Symp. Circuits Syst.*, May 2015, pp. 718–721.

[15] M. Yang, S. C. Liu, and T. Delbruck, “A dynamic vision sensor with 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding,” *IEEE J. Solid-State Circuits*, vol. 50, no. 9, pp. 2149–2160, Sep. 2015.

[16] D. P. Moeys *et al.*, “A sensitive dynamic and active pixel vision sensor for color or neural imaging applications,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 123–136, Feb. 2018.

[17] B. Son *et al.*, “4.1 A 640x480 dynamic vision sensor with a 9μm pixel and 300Meps address-event representation,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2017, pp. 66–67.

[18] M. Guo, J. Huang, and S. Chen, “Live demonstration: A 768×640 pixels 200Meps dynamic vision sensor,” in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2017, doi: 10.1109/ISCAS.2017.8050397.

[19] R. V. Rullen and S. J. Thorpe, “Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex,” *Neural Comput.*, vol. 13, no. 6, pp. 1255–1283, Jun. 2001.

[20] G. Orchard, A. Jayawan, G. K. Cohen, and N. Thakor, “Converting static image datasets to spiking neuromorphic datasets using saccades,” *Frontiers Neurosci.*, vol. 9, pp. 1–11, 2015, Art. no. 437.

[21] T. Serrano-Gotarredona and B. Linares-Barranco, “Poker-DVS and MNIST-DVS. their history, how they were made, and other details,” *Frontiers Neurosci.*, 2015.

[22] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, “STDP-based spiking deep neural networks for object recognition,” *CoRR*, vol. abs/1611.01421, 2016. [Online]. Available: http://arxiv.org/abs/1611.01421

[23] P. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers Comput. Neurosci.*, vol. 9, p. 99, 2015. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fncom.2015.00099

[24] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Frontiers Neurosci.*, 2016.

[25] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, “HOTS: A hierarchy of event-based Time-Surfaces for pattern recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1346–1359, Jul. 2016.

[26] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, “HFirst: A temporal approach to object recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 2028–2040, Oct. 2015.

[27] G. Orchard, X. Lagorce, C. Posch, S. B. Furber, R. Benosman, and F. Galluppi, “Real-time event-driven spiking neural network object recognition on the SpiNNaker platform,” in *Proc. 2015 IEEE Int. Symp. Circuits Syst.*, May 2015, pp. 2413–2416.

[28] G. K. Cohen, G. Orchard, S. H. Leng, J. Tapson, R. Benosman, and A. van Schaik, “Skimming digits: Neuromorphic classification of spike-encoded images,” *Frontiers Neurosci.*, vol. 10, pp. 1–11, 2016, Art. no. 184.

[29] A. G. Andreou *et al.*, “Real-time sensory information processing using the TrueNorth Neurosynaptic System,” in *Proc. 2016 IEEE Int. Symp. Circuits Syst.*, May 2016, pp. 2911–2911.

[30] A. Amir *et al.*, “A low power, fully event-based gesture recognition system,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, Jul. 2017, pp. 7388–7397.

[31] A. Luczak, B. L. McNaughton, and K. D. Harris, “Packet-based communication in the cortex,” *Nature Rev. Neurosci.*, vol. 16, no. 12, 2015. [Online]. Available: http://dx.doi.org/10.1038/nrn4026

[32] A. Yousefzadeh, G. Orchard, E. Stromatias, T. Serrano-Gotarredona, and B. Linares-Barranco, “Hybrid neural network, an efficient low-power digital hardware implementation of event-based artificial neural network,” in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.

[33] A. Yousefzadeh, “Real time demo, symbol recognition and saccade prediction network,” https://youtu.be/Tw5Jbi_on-4, 2016.

[34] H. Mostafa, “Supervised learning based on temporal coding in spiking neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: 10.1109/TNNLS.2017.2726060.

[35] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, “Spatio-temporal backpropagation for training high-performance spiking neural networks,” *CoRR*, vol. abs/1706.02609, 2017. [Online]. Available: http://arxiv.org/abs/1706.02609

[36] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Frontiers Neurosci.*, vol. 10, pp. 1–13, 2016, Art. no. 508.

[37] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, “An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data,” *Frontiers Neurosci.*, vol. 11, pp. 1–17, 2017, Art. no. 350.

[38] G. Hinton, (unpublished) http://www.cs.toronto.edu/˜hinton. Accessed on: May 2018.

[39] Y. Lecun, http://yann.lecun.com/exdb/mnist. Accessed on: May 2018.