Spiking Networks for Event-Based Angular Velocity Regression

Viola Campos

Dept. Design, Computer Science, Media (DCSM)
RheinMain University of Applied Sciences

1 Introduction

Spiking neural networks (SNNs) are biologically inspired networks that are similar to current Artificial Neural Networks (ANNs) in terms of network-topology but use a different neuron model [Vre03]. The key differences to classical ANNs are, that spiking neurons process binary temporal events called *spikes* instead of numeric values and introduce the concept of time into their operating model. Like biological neurons, spiking neurons do not fire at each propagation cycle but only when their *membrane potential* – an internal state of the neuron – reaches a certain threshold value. When a neuron fires, it generates a spike that travels to other connected neurons which, in turn, increase or decrease their potentials according to the synaptic weight of the connection. If no incoming spikes arrive at a neuron, the membrane potential decays over time. Since a neuron will not generate any output spikes in the absence of incoming spikes, activity in an SNN is limited to active regions while parts of the net remain idle, which allows computation to be skipped for such neurons.

So in spiking neural systems, information is encoded as a temporal sequence of spikes (a *spike-train*) rather than numeric values. A device which provides this asynchronous data directly, is a Dynamic vision sensor (DVS) or event-based camera. It captures changes in the surrounding scene asynchronously as per-pixel brightness changes (events) and encodes visual information into an asynchronous and sparse stream of events [GDO*19]. Using event data as input for an SNN seems to be a suitable model for biological visual perception: each neuron receives incoming spikes only from a small region of the visual space (a receptive field) and spikes are only generated and transmitted for regions with a certain amount of input spikes, which models selective attention.

This work is closely related to a paper from Gehrig, Shrestha, Mouritzen and Scaramuzza from 2020 [GSMS20], which develops and trains an SNN on short event sequences from a rotating DVS to predict the 3-DOF angular velocity of the camera. Figure 1 illustrates the idea. Based on their work, we mainly investigate the following open questions:

– Gehrig et al. use a synthetic dataset with simple uniform rotations to train and evaluate their network. Since event-based sensors are prone to noise, is it possible to train the SNN on 'real world' DVS-recordings and which accuracy can be achieved?

– Which further optimizations can be used to improve the results?

In summary, the contributions of this work are:

– A dataset of 'real world' event data, recorded with a DAVIS 346 and a detailed evaluation of SNN models trained on both types of event data.

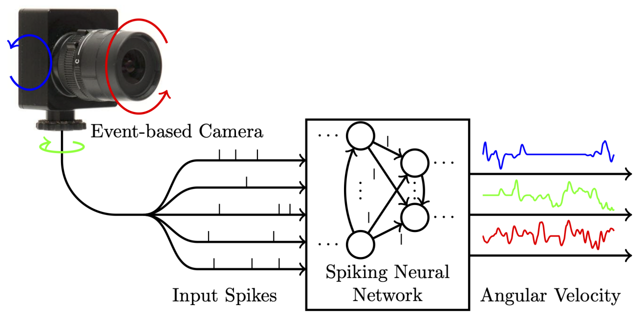


Fig. 1: Processing pipeline for angular velocity prediction using a spiking neural net and event-based vision. From [GSMS20].

– Parameter optimizations for the SNN architecture used in [GSMS20] which result in a reduction of the relative error from 26% to 17% for the original dataset.
– Formulation of an alternative approach for training SNNs with backpropagation in a discretized simulation of the continuous event-stream to achieve higher accuracy in the input representation.

Related work There exist several works which use a spiking neural network to solve low-level visual tasks given events from an event-camera. Although SNNs are a natural choice for spatio-temporal problems, they have largely been applied to classification problems like in [NMZ19], [ZG18], [SS17], [SFJ*20] and [MGND*19]. Event-based optical flow can be estimated by applying spatio-temporally oriented filters on the event stream to distinguish between different motion speeds and directions. [OBECT13] and [BTN15] define hand-crafted filters, whereas [PVSDC19] learns them from event data using Spike-Timing-Dependent Plasticity (STDP), a brain-inspired learning rule. Other works use handcrafted SNNs to solve the event-based stereo correspondence problem [OIBI17] [DFR*17] or develop systems for obstacle avoidance and target acquisition in small robots [MBR*18] [BDM*17].

To the best of our knowledge, [GSMS20] is the first paper which uses a spiking neural network to solve a temporal regression problem for DVS input data.

2 Preliminaries

2.1 Neuromorphic Vision Datasets

Event-based vision datasets consist of a stream of spike events which are triggered by intensity changes at each pixel in the sensing field of the event-based camera. These events are recorded with a precision of microseconds in two channels according to the different directions (increase or decrease) of intensity change. A sequence of such events can be represented as a $2 \times H \times W \times T$ sized event pattern, where H, W represent the height and width of the sensing field and T is the length of recording time. We distinguish two types of neuromorphic Datasets: DVS-converted, which generate artificial spike sequences from frame-based static datasets and DVS-captured datasets, which generate spike events via natural motion in a 3-dimensional surrounding.

As stated in [HWD*20], [ICL18], DVS-converted datasets lack parts of the spatiotemporal information and might not be good enough to benchmark SNNs due to the static 2-dimensional information source. Advantages of synthetically generated event data compared to real recordings are that they do not suffer from noise, which is a problem of current event-based sensors and that ground truth can be provided without further effort for such data.

2.2 Spiking Neurons

Spiking neurons model the dynamics of a biological neuron. They receive spikes, short binary signals, whose magnitudes and signs are determined by the synaptic weight of the incoming connection and form the *Post-Synaptic Potential (PSP)*. The accumulation of all incoming signals in a neuron determines the neuron's state, the *membrane potential* $u(t)$. Whenever the membrane potential exceeds a threshold ϑ, the spiking neuron generates an outgoing spike which is distributed to all connected neurons. Immediately after the spike, the neuron suppresses its membrane potential and ignores further incoming spikes so that the spiking activity is regulated. This self-suppression mechanism is called *refractory response*. While a neuron receives no incoming spikes, its synaptic potential decays towards the neuron's resting potential. Figure 2 illustrates the process.

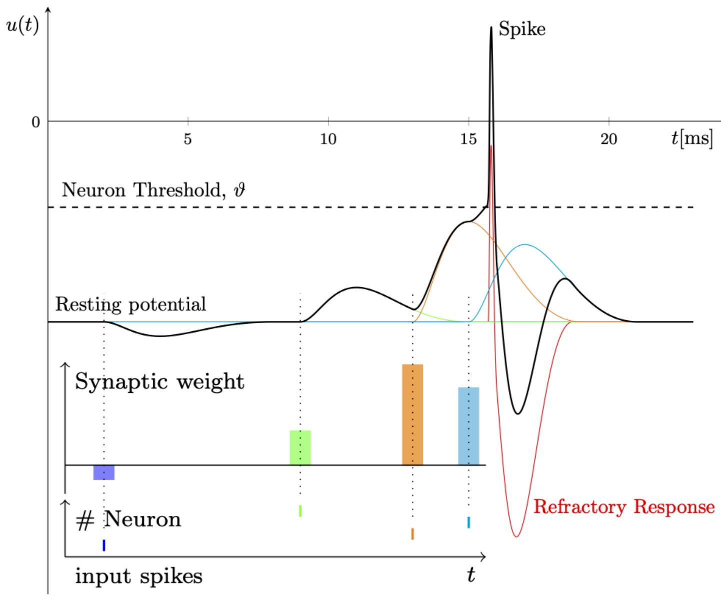


Fig. 2: Dynamics of a spiking neuron. From [GSMS20].

There are various mathematical models in neuroscience describing the dynamics of a spiking neuron with different level of detail, from the simple leaky integrate and fire (LIF) neuron [GK02] to the complex Hodgin-Huxley neuron [HH52]. In this work, we use the Spike Response Model (SRM) from [Ger95]. In SRM the PSP response is modeled by a spike response kernel, $\epsilon(t)$, which is scaled by the synaptic weight of the connection and distributes the effect of incoming spikes in time. Similarly, the refractory response is described by a refractory kernel $\nu(t)$. SRMs are simple, but versatile neuron models as they can represent various characteristics of spiking neurons with appropriate kernels.

Input to a neuron is a sequence of spikes, a *spike-train*: $s_i(t) = \sum_{t^{(f)} \in \mathcal{F}} \delta(t - t_i^{(f)})$. Here $t_i^{(f)}$ is the time of the f^{th} spike of the i^{th} input, $\mathcal{F}$ is the set of times of the individual spikes and δ is the dirac delta function with $\delta(t - t_0) = 0 \ \forall \ t \neq t_0$. In SRM, the incoming spikes are converted into a *spike response signal* $a_i(t)$ by convolving $s_i(t)$ with the spike response kernel $\epsilon(\cdot)$ [SO18]:

$$a_i(t) = (\epsilon * s_i)(t) \tag{1}$$

Similarly, the refractory response of a neuron is $(\nu * s)(t)$ where $\nu(\cdot)$ is the refractory kernel and $s(t)$ is the neuron's output spike-train. Each spike response signal is scaled by a synaptic weight w_i to generate the post synaptic potential. The neuron's state (*membrane potential*) $u(t)$ is the sum of all PSPs and refractory responses

$$u(t) = \sum w_i(\epsilon * s)(t) + (\nu * s)(t) = \boldsymbol{w}^\intercal \boldsymbol{a}(t) + (\nu * s)(t) \tag{2}$$

Whenever $u(t)$ reaches a predefined threshold ϑ, an output spike is generated. Hence the *spike function* $f_s(\cdot)$ is defined as

$$f_s(u) : u \to s, s(t) := s(t) + \delta(t - t^{(f+1)}) \text{ where } t^{(f+1)} = min\{t : u(t) = \vartheta, t > t^{(f)}\} \tag{3}$$

The derivative of of the spike function is undefined, which is an obstacle for backpropagating errors in an SNN.

2.3 Feedforward Spiking Neural Networks

Connecting spiking neurons through synapses constructs a Spiking Neural Network (SNN) model. One of the advantages of SNNs is their ability to process event-data directly, as events can be interpreted as incoming spikes which are fed to the network without further preprocessing. Our SNN model has two input channels for each pixel location to account for the polarity of incoming events and one input neuron per pixel location. The network is a feedforward SNN with n_l layers.

Considering a layer l with N_l neurons, synaptic weights $(W)^{(l)} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{N_{l+1}}]^\intercal \in \mathbb{R}^{N_{l+1} \times N_l}$ between layer l and $l+1$, the network forward propagation is described as:

$$\boldsymbol{s}^{(0)}(t) = \boldsymbol{s}_{in}(t) \tag{4}$$

$$\boldsymbol{u}^{(l+1)}(t) = (W)^{(l)}(\epsilon * \boldsymbol{s}^{(l)})(t) + (v * \boldsymbol{s}^{(l+1)})(t) \tag{5}$$

$$\boldsymbol{s}^{(l)}(t) = f_s(\boldsymbol{u}^{(l+1)}(t)) = \sum_{t^{(f)} \in \{t | \boldsymbol{u}^{(l)}(t) = \vartheta\}} \delta(t - t^{(f)}) \tag{6}$$

where the inputs $\boldsymbol{s}^{(0)}(t) = \boldsymbol{s}_{in}(t)$ and outputs $\boldsymbol{s}^{(out)}(t) = \boldsymbol{s}_{n_l}(t)$ are spike-trains rather than numeric values.

The spike response kernel and refractory kernel are defined as

$$\epsilon(t) = \frac{t}{\tau_s} e^{1-\frac{t}{\tau_s}} \mathcal{H}(t) \tag{7}$$

$$\nu(t) = -2\vartheta e^{1-\frac{t}{\tau_r}} \mathcal{H}(t) \tag{8}$$

$\mathcal{H}(\cdot)$ is the Heaviside step function, τ_s and τ_r are time constants of the spike response kernel and refractory kernel. The spike response kernel (equation 5 and 7) distributes the effect of input spikes over time, peaking slightly delayed and exponentially decaying after the peak, where τ_s defines the position of the peak. The effective temporal range of the kernel allows a short term memory mechanism in an SNN.

3 Method

3.1 Input Data

We train our network with two types of input data. A synthetic dataset from [GSMS20] which was generated with ESIM, an event-camera simulator [RGS18]. ESIM renders images along a camera trajectory to interpolate a continuous visual signal at each pixel position. This signal is used to emulate the operating principle of an event camera and generate events with a user-chosen contrast threshold. The dataset consists of sequences which were generated from a subset of 10000 panorama images from the publicly available Sun360 dataset[1] with random constant rotational motion covering all axes.

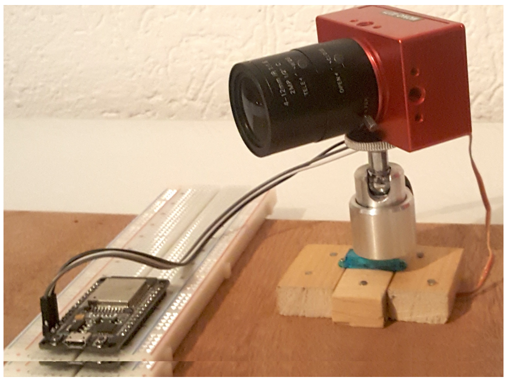
The second dataset consists of real-world event data, recorded with a DAVIS346. The recordings include sequences using a tripod where the camera is rotated around one axis only as well as sequences with a freely rotating camera. To prevent overfitting, the sequences show a variety of scenes in different lighting conditions. The IMU sensor of the camera is used to provide ground truth angular velocity. While the angular velocity does not change drastically during a sequence in the simulated event dataset, the recorded dataset also contains samples with changing velocities.

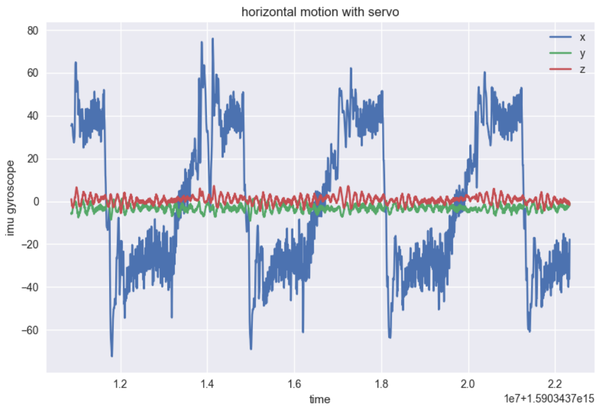
In order to record sequences at a constant rotational speed, the camera was mounted on a servo motor (RoVoR S0307) which was controlled by an Arduino board. It turned out that the recordings were unsuitable as training sequences, as the motor did not move the camera continuously, but in small steps. Figure 3 shows the setup and the IMU output of a sample sequence.

3.2 Preprocessing

SNNs operate on time-continuous data streams, so they must be discretized for simulation on GPUs. Choosing the discretization time steps means a trade-off between accuracy of the simulation (time steps as small as possible) and availability of memory and computational resources. We chose time steps of 1 millisecond and restricted the simulation time to 100 milliseconds.

[1] `http://people.csail.mit.edu/jxiao/SUN360/`


(a) setup



(b) Output of camera's integrated IMU during rotation

Fig. 3: DAVIS 346 mounted on small servo motor

3.3 Network Architecture

The network architecture is a convolutional spiking neural network, consisting of 5 convolutional layers, followed by an average pooling and a fully connected layer. The convolutional layers perform spatial downsampling with stride 2 while doubling the number of channels with each layer, starting with 16 channels in the first layer and ending with 256 channels in layer five. The spatial convolution alternates with a convolution in time in each layer, where every neuron distributes its incoming spikes through time to compute the membrane potential as described in equation (5) and (7). The time constants for the decay rate of the kernels τ_s and τ_r are set layer-wise.

The convolutional layers are followed by a global average pooling step. For each output channel i of the last convolution layer, the average number of spikes per timestep $g_i(t)$ is computed as follows:

Let $\boldsymbol{S}_i(t, x, y)$ denote a spatial spike-train resulting from the i-th channel of the previous layer

$$\boldsymbol{S}_i(t, x, y) = \sum_{t^{(f)} \in \mathcal{F}_i(x,y)} \delta(t - t^{(f)}) \tag{9}$$

where $\mathcal{F}_i(x, y)$ is the set of spike times in the i-th channel at the spatial location (x, y). Then $g_i(t)$ is i-th output of the pooling operation and is described as

$$g_i(t) = \sum_{\substack{x \in \{0,\ldots,W-1\} \\ y \in \{0,\ldots,H-1\}}} \boldsymbol{S}_i(t, x, y) \tag{10}$$

where W and H are width and height of previous channel.

The last fully-connected layer connects these averaged spike-trains to three, non-spiking output neurons for regressing the angular velocity prediction in time.

As we use the SNN for a regression task, we need to convert the output spike-train to numeric values. This is done by convolving the spikes with the spike response kernel $\epsilon(\cdot)$, similar to the computation of the post synaptic potential. To ensure a smooth function, it is reasonable to choose a high value for τ_s in this layer. So the prediction of the angular velocity $\boldsymbol{\omega}$ is

$$\boldsymbol{\omega}(t) = \boldsymbol{W}^{(n_l)}(\epsilon * \boldsymbol{s}^{(n_l)})(t) \tag{11}$$

3.4 Loss function

The loss function L is taken from [GSMS20] and is defined as the time-integral over the euclidean distance between the predicted angular velocity $\boldsymbol{\omega}(t)$ and ground truth angular velocity $\hat{\boldsymbol{\omega}}(t)$:

$$L = \frac{1}{T_1 - T_0} \int_{T_0}^{T_1} \sqrt{\boldsymbol{e}(t)^\intercal \boldsymbol{e}(t)} dt \tag{12}$$

where $\boldsymbol{e}(t) = \boldsymbol{\omega}(t) - \hat{\boldsymbol{\omega}}(t)$. To give the system some settling time, the evaluation of the error function is only activated after processing the first 50 ms of the event sequence.

3.5 Training Procedure

The training of the SNN is based on first-order optimization methods, similar to backpropagation in ANNs. Therefore, we must compute gradients of the loss function with respect to the parameters of the network. This is done with the publicly available PyTorch implementation [2] of SLAYER [SO18], a backpropagation mechanism for supervised learning in deep spiking neural networks.

To perform backpropagation in an SNN, two problems need to be solved: the state of the system is determined by the sequence of previous spikes, so it is uncertain how much each spike contributes to the error. The second problem is the non-differentiability of the spike function, so that an approximation for the derivative is needed.

Further details on how error is assigned to previous time-points and how the derivative of the spike function is approximated with SLAYER can be found in [SO18].

4 Processing spikes with accurate timestamps

Actually, an SNN is a time-continuous dynamic system. To simulate it on GPUs, it must be discretized, whereby the choice of step size means a trade-off between accuracy of the simulation and consumption of computational resources. We propose a new input format together with a modified processing pipeline to achieve higher accuracy in a discretized simulation.

We discretize the system with a sampling time T_s such that $t = nT_s, n \in \mathbb{Z}$ and let N_s denote the total number of samples in the period $t \in [0, T]$.

[2] `https://bitbucket.org/bamsumit/slayer/src/master/`

4.1 Spike Representation

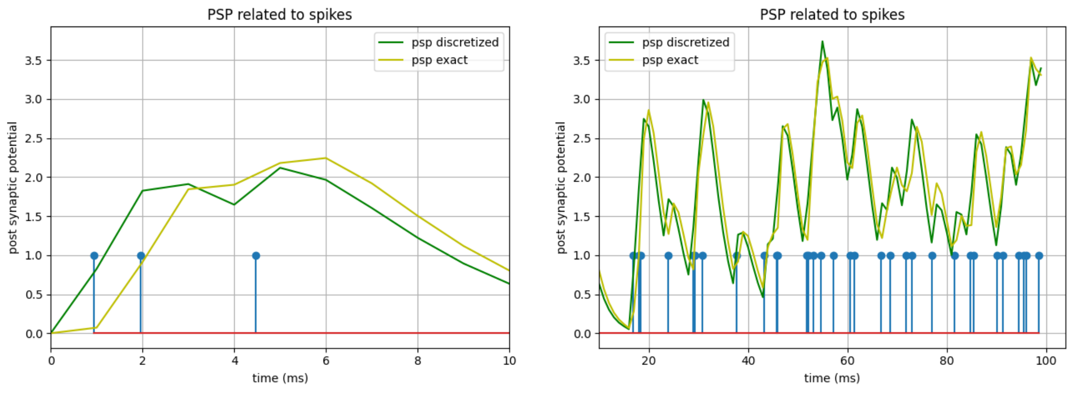
Gehrig et al. use a binary discretized spike representation in [GSMS20]. Incoming events are assigned to sampling times. These 'time bins' are stacked to a 4-dimensional vector of size $C \times H \times W \times N_s$ where C denotes the number of channels and $H \times W$ are the input dimensions. Its entries are 1 if an event of the given channel is found at the corresponding pixel position during a time slice or 0 if no event was found. When this representation is used to compute the spike response signal to equation (1), the discrete time steps are used as spike times during convolution with the spike response kernel ϵ.

$$\epsilon(t) = \frac{t}{\tau_s} e^{1-\frac{t}{\tau_s}} \mathcal{H}(t),\ t \in nT_s,\ n \in [0, N_s] \tag{13}$$

Since the exact timestamps of the events are known (in μs-resolution), we use the entries in the input vector to store the relative spike time Δt during the time slice for events and zero otherwise. These exact spike times are used as additional information to compute more accurate samples of the spike response signal. The discretized version of the spike response for a spike with relative spike time Δt becomes:

$$\epsilon(t, \Delta t) = \frac{t}{\tau_s} e^{1-\frac{t}{\tau_s}} \mathcal{H}(t),\ t \in nT_s - \Delta t,\ n \in [0, N_s] \tag{14}$$

Figure 4 shows the difference: determining the post synaptic potential with exact timestamps instead of sampling times leads to a slightly different PSP curve.



(a) detail view of 10ms (b) Full sequence of 100ms

Fig. 4: Discretized simulation of development of the post synaptic potential for one spiking neuron. The green curve assumes spikes at sampling times, the yellow curve uses exact spike times. Incoming spikes are plotted in blue.

The spike function $f_s(u(t))$ needs to be adjusted as well. If at a sampling time nT_s a PSP value $u(nT_s) > \vartheta$ is found, the exact spike time t with $(n-1)T_s < t < nT_s$ must be determined to provide spikes in the expected input format to the next layer. We perform a linear regression between $u((n-1)T_s)$ and $u(nT_s)$ to approximate the relative spike time Δt.

4.2 Average Pooling

The authors of [GSMS20] use an operation which they denote as *global average spike pooling (GASP)* to compute an averaged spike-train $g_i(t)$ (described in section 3.3).

$$g_i(t) = \frac{1}{W \cdot H} \sum_{\substack{x \in \{0,\dots,W-1\} \\ y \in \{0,\dots,H-1\}}} \boldsymbol{S}_i(t, x, y) \tag{15}$$

where $\boldsymbol{S}_i(t, x, y)$ denotes the spatial spike-train resulting from the i-th channel of the previous layer and W and H are width and height of the previous channel.

This pooling operation is pointless on spike-trains in our extended format as the average value of an unspecified number of relative timesteps has no reasonable interpretation in the system. Instead, the pooling operation is modified to compute the average number of spikes per channel and timestep as intended.

5 Experiments

Based on the reference implementation[3] from [GSMS20] we performed several experiments to further improve the accuracy of the regression.

Table 1 compares the root mean square error and the relative error of the predicted angular velocities for simulated and real world data. While the absolute root mean squared error (RMSE) is slightly lower for the recorded event sequences than for the generated ones, the relative error is much higher. The reason for this is, that the mean angular velocity in the 'real world' dataset is lower than in the simulated dataset. Achieving low relative error at low absolute speed is difficult as the relative error converges towards infinite near zero angular velocity.

	mean		SNN	
	simulated data	real world data	simulated data	real world data
RMSE (deg/s)	226.9	72.94	66.3	44.06
relative error	100%	100%	26%	54%

Table 1: Comparison of errors on different test sets. The naive mean prediction serves as a baseline.

Figure 6 illustrates the predicted angular velocities and ground truth for different samples of event sequences of 100ms. The plots show that the network needs some settling time before it predicts good values which corresponds with the training method: evaluation of the error loss, backpropagation and weight updates are only performed after 50ms of the sequence during training.

[3] `https://github.com/uzh-rpg/snn_angular_velocity`

To evaluate whether the starting time of loss evaluation influences the result, the SNN was trained in different settings where evaluation of loss and penalizing errors started after 20ms, 30ms, 40ms and 50ms. At the same time the error between predicted and ground truth angular velocity was evaluated over different time periods starting from the whole sequence to the last 20ms only. Tables 2 and 3 give a comparison of errors after 40 epochs of training for these scenarios on simulated and recorded event data. When trained on simulated data, the start time for training does not seem to to have a significant effect on the resulting accuracy of the network. Best results are achieved when training starts at 40ms but the results do only differ by a few degrees. Using 'real world' data, the results vary more, which is probably related to the quality of the input data.

Regarding the influence of the point in time at which the error evaluation begins, it is noticeable that after a short settling time, the network already provides comparatively good predictions even before the training has started. Here, the recursive character of the network is evident: in each spiking layer the same operation is performed for every time step in the sequence.

							Evaluate error after:	
	0ms		30ms		50ms		80ms	
start training:	RMSE	relative error	RMSE	relative error	RMSE	relative error	RMSE	relative error
30 ms	...	...	56.37	0.69	47.73	0.57	46.78	0.56
40 ms	...	...	45.11	0.56	61.67	0.73	43.42	0.53
50 ms	49.47	0.63	41.25	0.51	44.10	0.54	43.31	0.54

Table 2: Comparison of different training scenarios for 'real world' event sequences of 100ms. Root mean square error in deg/s and median of relative error are reported after 40 epochs of training for different combinations of starting times for error evaluation and backpropagation.

							Evaluate error after:	
	0ms		30ms		50ms		80ms	
start training:	RMSE	relative error	RMSE	relative error	RMSE	relative error	RMSE	relative error
30 ms	99.68	0.35	69.24	0.25	69.07	0.25	68.96	0.25
40 ms	98.06	0.35	66.89	0.25	66.79	0.25	66.62	0.25
50 ms	100.03	0.36	70.96	0.26	70.67	0.26	70.51	0.26

Table 3: Evaluation from table 2 on simulated event sequences.

5.1 Hyperparameter selection

As stated in [TGK*19], the choice of suitable hyperparameters in spiking neural networks is crucial for their learning ability. In the SRM neurons used in this work, the threshold value ϑ

in the spike generation function (3) influences how spikes propagate through the network. A high value for ϑ reduces the rate of outgoing spikes compared to incoming ones, while a low ϑ increases the spike rate. The spike generation process in a layer is further influenced by τ_s and τ_r, temporal constants which define the decay rate of the spike response and the refractory kernel from equations (7) and (8). To ensure spike propagation until the output layer, ϑ, τ_s and τ_r must be chosen appropriately in relation to expected input spike rates to prevent both, vanishing and exploding signals.

Starting from the parameter set used in [GSMS20], we performed a grid search to optimize the hyper-parameters of the SNN for both simulated and recorded event data. Since Gehrig et al. increase the temporal parameters τ_s and τ_r with network depth and explain this choice as important to compensate for high event rates at the input layer and lower dynamics in deeper layers, the search focused on combinations of parameters which ensure a nearly constant number of spikes per neuron in the individual layers. As this approach did not lead to significant improvements compared to the results from [GSMS20], the search was continued with more randomized parameter combinations in a second step.

Against expectations, we achieved the best accuracy results with a set of constant parameters for all convolutional layers. In the fully connected layer which connects the pooling layer with the three output neurons, a rather high value for τ_s worked best, as it lead to smoother predictions. After 40 epochs of training, our best model (Setup B in table 4) achieved a root mean square error of 49,2 deg/s which is a relative error of 17.1 % compared to 66.3 deg/s RMSE or a relative error of 26% from the original paper.

Figure 5 shows the development of the loss function for training and test data with exemplary parameter settings and table 4 lists the hyperparameters used in each model in more detail. All models are trained on the recorded event data set.

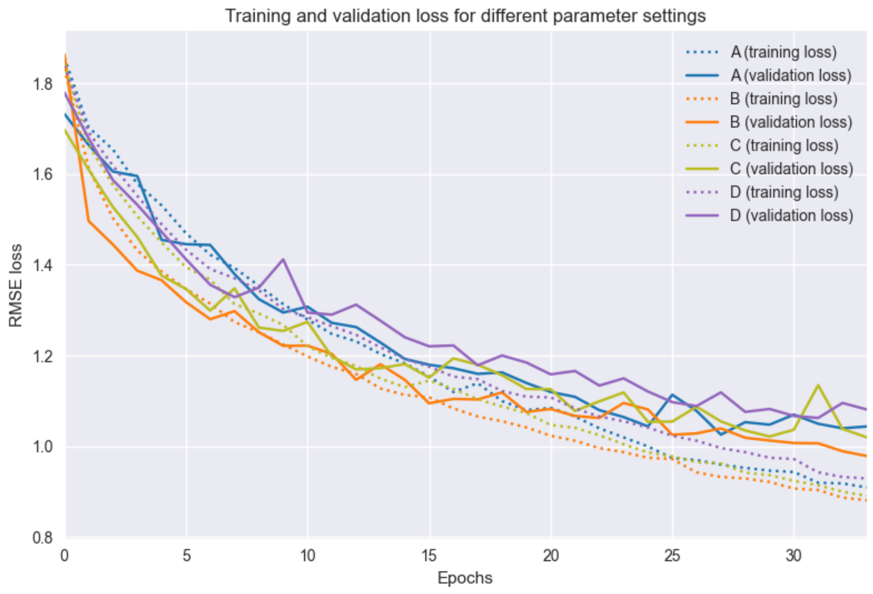


Fig. 5: Training and validation loss for different SNNs with same architecture and different hyperparameters. For parameter settings see table 4.

Model		Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC	relative error
A (original paper)	τ_s	2.00	2.00	4.00	4.00	4.00	8.00	22.4%
	τ_r	1.00	1.00	4.00	4.00	4.00	-	
	ϑ	0.30	0.30	0.30	0.40	0.40	-	
	# spikes / neuron	2.11	2.84	2.36	1.68	2.13	-	
B (equal values)	τ_s	2.00	2.00	2.00	2.00	2.00	2.00	17.1 %
	τ_r	1.00	1.00	1.00	1.00	1.00	-	
	ϑ	0.30	0.30	0.30	0.30	0.30	-	
	# spikes / neuron	1.23	1.93	3.51	2.43	5.83	-	
C	τ_s	2.00	2.00	4.00	4.00	4.00	8.00	29.1 %
	τ_r	1.00	1.00	4.00	4.00	4.00	-	
	ϑ	0.23	0.28	0.30	0.3	0.25	-	
	# spikes / neuron	1.25	1.31	1.82	1.14	0.89	-	
D	τ_s	2.00	2.00	4.00	4.00	5.00	8.00	31.4 %
	τ_r	1.00	1.00	4.00	4.00	6.00	-	
	ϑ	0.30	0.30	0.30	0.30	0.30	-	
	# spikes / neuron	0.93	2.10	1.62	1.13	1.55	-	

Table 4: Hyperparameters for examplary SNN models. The table shows the relative error for each model after 40 epochs of training and additionally the mean number of spikes per neuron for each layer.

5.2 Processing spikes with accurate timestamps

We trained a spiking neural network with our extended processing pipeline on the simulated event dataset for 40 epochs which led to a prediction accuracy similar to the original paper. Table 5 shows the resulting error, figure 6 plots some sample predictions.

The model uses the set of hyperparameters which performed best for the original SNN. However, inspection of the dataflow shows much higher spike rates in deeper layers than in the original version, suggesting that the results could be further improved by hyperparameter optimization.

	original input representation	representation with exact timestamps
RMSE (deg/s)	66.3	65.78
relative error	26%	25.4%

Table 5: Comparison of errors in SNNs using different forms of input representation.

6 Conclusions

In this work, we investigated an existing spiking neural network which uses backpropagation-based training to regress angular velocities on event-based data. It was shown that the network is able to learn and predict angular velocities for recordings from a DVS instead of the simulated input data from the original work. Although the 3-DOF motion was more complex and the sensor introduced noise into the input data, the SNN was able to learn the angular velocities, however with less accuracy. We evaluated the influence of hyperparameters and training settings and found better parameters for the proposed network. Further, a version of the SNN was developed, which processes event data in a new input format which reduces errors in the input representation.

However, the accuracy of the predictions is still low compared to state-of-the-art ANNs in classical computer vision while the backpropagation-based training approach is expensive in terms of memory consumption and processing time. Combining spiking neural nets with backpropagation as learning method does not feel like a natural choice. Backpropagation requires to unroll the SNN in time at discrete time steps which is expensive, introduces problems of recurrent networks like vanishing and exploding gradients and does not correspond to the time-continuous nature of the system. Furthermore, in each time step the state of the whole network needs to be updated, which does not allow to address localized activity in the network and skip computations in idle areas.

We are still convinced, that the combination of spiking neural networks and even-based vision is promising. Nevertheless, in order to exploit the full potential of the technology, new learning approaches are needed.

References

BDM*17. Blum H., Dietmüller A., Milde M., Conradt J., Indiveri G., Sandamirskaya Y.: A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor. *Robotics Science and Systems, RSS 2017* (2017).

BTN15. Brosch T., Tschechne S., Neumann H.: On event-based optical flow detection. *Frontiers in neuroscience 9* (2015), 137.

DFR*17. Dikov G., Firouzi M., Röhrbein F., Conradt J., Richter C.: Spiking cooperative stereo-matching at 2 ms latency with neuromorphic hardware. In *Conference on Biomimetic and Biohybrid Systems* (2017), Springer, pp. 119–137.

GDO*19. Gallego G., Delbrück T., Orchard G., Bartolozzi C., Taba B., Censi A., Leutenegger S., Davison A. J., Conradt J., Daniilidis K., Scaramuzza D.: Event-based vision: A survey. *ArXiv abs/1904.08405* (2019).

Ger95. Gerstner: Time structure of the activity in neural network models. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics 51 1* (1995), 738–758.

GK02. Gerstner W., Kistler W. M.: *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press, 2002.

GSMS20. Gehrig M., Shrestha S. B., Mouritzen D., Scaramuzza D.: Event-based angular velocity regression with spiking networks. *arXiv preprint arXiv:2003.02790* (2020).

HH52. Hodgkin A. L., Huxley A. F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology 117*, 4 (1952), 500.

HWD*20. He W., Wu Y., Deng L., Li G., Wang H., Tian Y., Ding W., Wang W., Xie Y.: Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. *arXiv preprint arXiv:2005.02183* (2020).

ICL18. Iyer L. R., Chua Y., Li H.: Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv preprint arXiv:1807.01013* (2018).

MBR*18. Milde M. B., Bertrand O. J., Ramachandran H., Egelhaaf M., Chicca E.: Spiking elementary motion detector in neuromorphic systems. *Neural computation 30*, 9 (2018), 2384–2417.

MGND*19. Mozafari M., Ganjtabesh M., Nowzari-Dalini A., Thorpe S. J., Masquelier T.: Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognition 94* (2019), 87–95.

NMZ19. Neftci E. O., Mostafa H., Zenke F.: Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine 36*, 6 (2019), 51–63.

OBECT13. Orchard G., Benosman R., Etienne-Cummings R., Thakor N. V.: A spiking neural network architecture for visual motion estimation. In *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (2013), IEEE, pp. 298–301.

OIBI17. Osswald M., Ieng S.-H., Benosman R., Indiveri G.: A spiking neural network model of 3d perception for event-based neuromorphic stereo vision systems. *Scientific reports 7* (2017), 40703.

PVSDC19. Paredes-Vallés F., Scheper K. Y. W., De Croon G. C. H. E.: Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception. *IEEE transactions on pattern analysis and machine intelligence* (2019).

RGS18. Rebecq H., Gehrig D., Scaramuzza D.: Esim: an open event camera simulator. In *Conference on Robot Learning* (2018), pp. 969–982.

SFJ*20. Samadzadeh A., Far F. S. T., Javadi A., Nickabadi A., Chehreghani M. H.: Convolutional spiking neural networks for spatio-temporal feature extraction. *arXiv preprint arXiv:2003.12346* (2020).

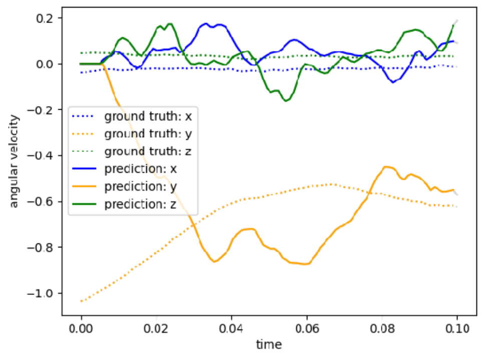
SO18. Shrestha S. B., Orchard G.: Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems* (2018), pp. 1412–1421.

SS17. Shrestha S. B., Song Q.: Robust learning in spikeprop. *Neural Networks 86* (2017), 54–68.

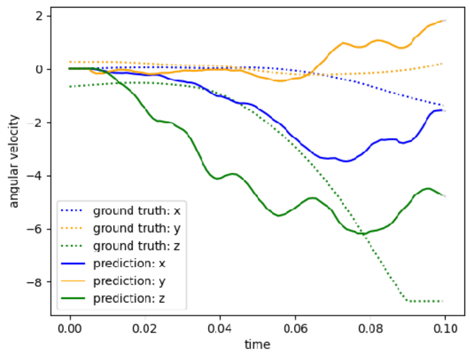
TGK*19. Tavanaei A., Ghodrati M., Kheradpisheh S. R., Masquelier T., Maida A.: Deep learning in spiking neural networks. *Neural networks : the official journal of the International Neural Network Society 111* (2019), 47–63.

Vre03. Vreeken J.: Spiking neural networks, an introduction.

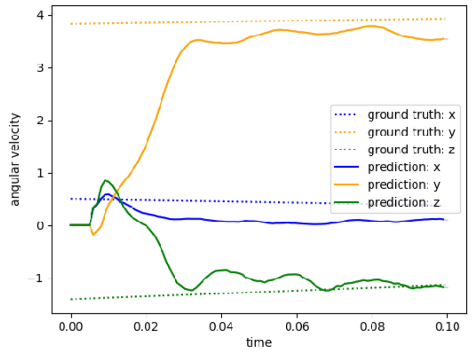
ZG18. Zenke F., Ganguli S.: Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation 30*, 6 (2018), 1514–1541.



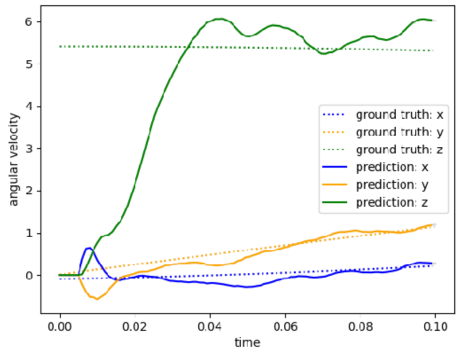
(a) DAVIS recording 1



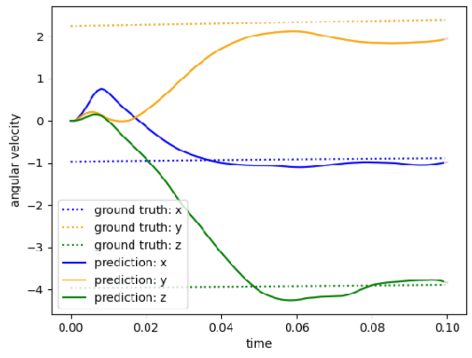
(b) DAVIS recording 2



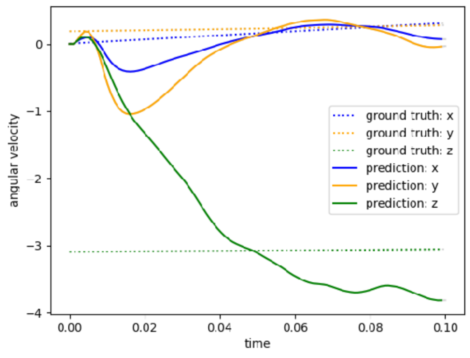
(c) Synthtic data 1



(d) Synthetic data 2



(e) Synthetic data, using exact timestamps 1



(f) Synthetic data, using exact timestamps 2

Fig. 6: Continuous-time angular velocity predicted by the SNN and corresponding ground truth. Upper row: 'real world' data, center: synthetic event data, lower row: synthetic data processed with a model, using spike representation from section 4 .