Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks

Conference Paper · December 2016

DOI: 10.1109/SIMPAR.2016.7862386

12 authors, including:

Jacques Kaiser
FZI Forschungszentrum Informatik
34 PUBLICATIONS **359** CITATIONS
SEE PROFILE

J. Camilo Vasquez Tieck
FZI Forschungszentrum Informatik
28 PUBLICATIONS **224** CITATIONS
SEE PROFILE

Christian Hubschneider
FZI Forschungszentrum Informatik
12 PUBLICATIONS **150** CITATIONS
SEE PROFILE

Peter Wolf
5 PUBLICATIONS **161** CITATIONS
SEE PROFILE

Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks

Jacques Kaiser[1], J. Camilo Vasquez Tieck[1], Christian Hubschneider[1], Peter Wolf[1], Michael Weber[1], Michael Hoff[2], Alexander Friedrich[2], Konrad Wojtasik[2], Arne Roennau[1], Ralf Kohlhaas[1], Rüdiger Dillmann[1,2], J. Marius Zöllner[1,2]

***Abstract*— Spiking neural networks are in theory more computationally powerful than rate-based neural networks often used in deep learning architectures. However, unlike rate-based neural networks, it is yet unclear how to train spiking networks to solve complex problems. There are still no standard algorithms and it is preventing roboticists to use spiking networks, yielding a lack of Neurorobotics applications. The contribution of this paper is twofold. First, we present a modular framework to evaluate neural self-driving vehicle applications. It provides a visual encoder from camera images to spikes inspired by the silicon retina (DVS), and a steering wheel decoder based on an agonist antagonist muscle model. Secondly, using this framework, we demonstrate a spiking neural network which controls a vehicle end-to-end for lane following behavior. The network is feed-forward and relies on hand-crafted feature detectors. In future work, this framework could be used to design more complex networks and use the evaluation metrics for learning.**

I. INTRODUCTION

Spiking Neural Networks (SNN) are the 3rd generation of Artificial Neural Network models [1]. These networks exhibit interesting properties over the 2nd generation (rate-based models) such as computational power [2], anytime computation [3] and energy efficiency [4].

Despite these properties being important for robotics, this type of neural network does not have as much applications as the 2nd generation deep neural networks applied to robotics [5, 6]. This is because rate-based neural networks often use computing paradigms which are not biologically plausible, such as discrete-time, iterative updates (lacking temporal locality) and shared parameters and states (lacking spatial locality). While most of the research in SNN focus on deriving generic learning rules without targeting specific applications [7, 8], we take a different approach and propose a SNN application which could benefit from powerful learning rules. In this paper, we provide a proof of concept that SNN can be applied for end-to-end robot control.

This paper has two main contributions. First, we present a modular framework to evaluate neural self-driving vehicle applications. Second, we benchmark a simple spiking neural network controller on a right-lane following experiment using this framework.

[1] FZI Research Center for Information Technology, 76131 Karlsruhe, Germany. jkaiser, tieck, hubschneider, wolf, mweber, roennau, kohlhaas, zoellner, dillmann@fzi.de

[2] Karlsruhe Institute of Technology (KIT), Germany. michael.hoff, alexander.friedrich3, konrad.wojtasik@student.kit.edu

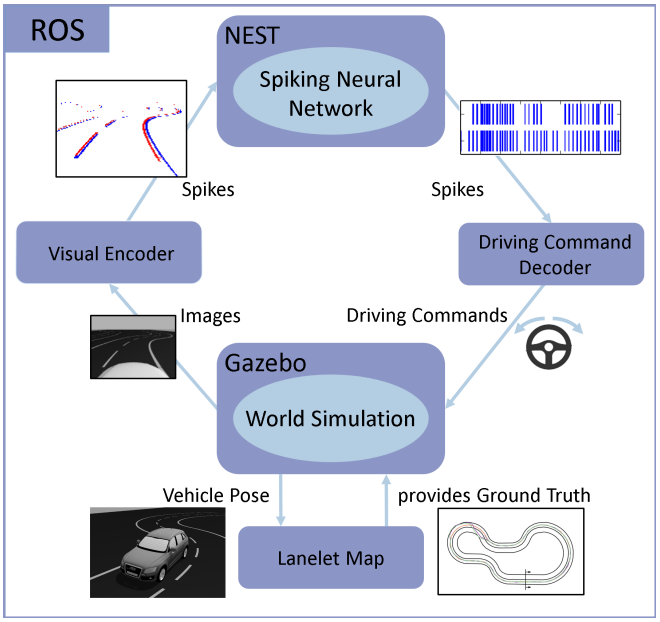


Fig. 1: Our framework for the neurorobotic simulation. ROS is used as a communication middle-ware between the different modules. Gazebo is the physic simulator, and represents the environment, including the vehicle. The vehicle sends images provided by its camera and is controlled by motor commands. The spiking neural network communicates only with spikes. The visual encoder and motor command decoder are described in Section IV-B.

The vehicle is equipped with a camera and is controlled by steering angle and velocity commands. To make the experiment compliant to spiking neuro-controller, we encode visual information into spikes with a novel frame-differentiation technique inspired by the Dynamic Vision Sensor (DVS) [9], or silicon retina. Decoding of spiking activity from motor neurons to steering angles and velocity commands is achieved with a simple agonist antagonist muscle model.

The SNN is feed-forward and consists of 16 neurons with static weight connections. It is inspired by the Braitenberg vehicle [10]. The concept of the Braitenberg is to have a simple neural network exhibiting seemingly intelligent behavior.

The network implementation is realized with PyNN [11] using the NEST [12] simulator.

The world model containing road circuits, the ego vehicle and simulated sensors is based on [13] and hosted in a Gazebo simulation [14]. Standard Gazebo camera sensors are used to generate artificial DVS inputs.

To connect the SNN with our simulated environment we use the Robot Operating System (ROS) [15]. Sensor data from the simulation is passed to the SNN, which, in turn, sends steering angles and target speeds as inputs to the simulated vehicle. With this separation, we are able to make the brain simulation agnostic of encoding and decoding details of the world simulation (see Fig. 1).

The setup is described in more detail in Section III, including the DVS simulation. In Section IV, we describe the spiking network architecture, how it encodes information from sensors and decode information to actuators. We benchmark this spiking network on the described experiment in Section V and show that it can drive robustly on simple circuits without intersections. Finally, we discuss how this approach could be extended with learning to solve more complicated scenarios in Section VI.

II. RELATED WORK

Spiking neural networks have already been used for applications related to robotics, such as optical flow computation [16] and pattern recognition [17]. SNN have also been used as modules on robots to either process sensory information or generate motor commands. In [18], a generic retina simulation is used to provide input to a SNN in order to smoothly move the eyes of an iCub robot to track a moving ball. However, the actual computation of the ball position was done outside of the network. This experiment was implemented in the Neurorobotics Platform [19] developed by the Human Brain Project in SP10, which provides the tools to combine a robotic simulation with an SNN simulation.

In [20], a SNN is trained in a supervised manner to learn simple arm movements. After training, the network needs to be provided with a label encoding the type of movement and the current state of the arm (proprioception) to perform the desired movements. Similarly, a network is trained in [21] with a functional model of the cerebellum to learn sinusoidal movements for a Myorobotics arm with one degree of freedom. The implementation runs on neuromorphic hardware but is currently limited to control one degree of freedom. A liquid state machine modeled using brain data from rats was trained in [22] to control a ball balancing robot in a simulation. The liquid state machine was trained in a supervised way by computing the optimal control of the robot to balance the ball.

Other works have already used SNN successfully to control a robot end-to-end from vision sensors. In [23], a liquid state machine is trained in a supervised way to learn the behavior of a simple classic robot controller. The trained liquid was able to learn a Braitenberg behavior and a basic obstacle avoidance behavior.

In this paper, we focus on an application related to the car industry by controlling a simulation of a vehicle end-to-end with a SNN. Rate-based neural networks have proven in different scenarios their ability to control a car to follow a road using camera sensor input [24]. Additionally, the car industry is also interested in biologically inspired vision sensors for their low latency and high dynamic range, making them able to operate quickly even in challenging lighting conditions [25]. Thanks to their event-based nature, SNN interface naturally with these new types of recent biologically inspired sensors [9, 26], unlike rate-based neural networks. In this paper, we simulate a Dynamic Vision Sensor (DVS) [9] to encode visual stimuli into spikes.

III. FRAMEWORK

We use the Robot Operating System (ROS) [15] as the underlying framework which also acts as message broker between the individual components. Implementing the modules as ROS nodes allows us to easily send images from the simulated sensors to the brain simulation and, in turn, steering commands generated by the spiking neural network back to control the vehicle in the simulated environment. Additionally, as information is only transferred via predefined messages, the world simulation and the brain simulation are not required to know implementation details of each other.

For evaluation purposes we generated maps of the simulated tracks using a plugin developed by [13]. With the help of these maps and the lanelet representation [27] we are able to obtain the ground truth for our evaluation. An overview of the framework which shows the interactions between the components is given in Fig. 1.

A. World simulation

In order to generate steering commands, we need to supply the spiking neural network with input data from the vehicle's environment. The latter is hosted in a Gazebo simulation [14] containing the world, the vehicle model and the camera sensors. We use an imperfect physics simulation with simplified slippage and friction, but on low speeds both can be neglected for our use case. Additionally, our vehicle is based on the Ackermann steering geometry which reduces slippage, and can be controlled via steering angle of the front axis and a signed target speed. On top of the vehicle we defined a standard Gazebo camera sensor facing forward. The recorded images are then processed by the DVS simulation described in the following section.

Circuits are designed using the world builder provided by [13]. The world builder provides predefined tiles containing straight road segments, curves with different radius and several intersection types. This way, new circuit layouts can be created easily to get more variation in training data. Each street segment template features two lanes, one in each direction with equal lane width, separated by a dashed center line. The simulated roads are flat and without any external visual disturbances. See Fig. 2a for an example from the camera's point of view.

For the created circuits we generated lanelet maps as ground truth. Lanelets are a representation of the drivable environment defined by the left and right boundaries of a lane segment [27]. They can be used to calculate distances between the ego vehicle and the center of a lane, as well as the offset between vehicle heading and lane orientation. This way we are able to obtain a metric for the quality of the vehicle pose.

B. DVS simulation

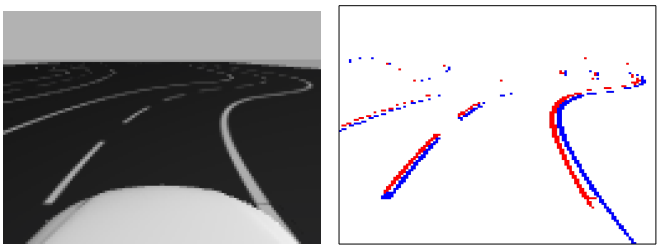
The Dynamic Vision Sensor (DVS) is a biologically inspired vision sensor [9]. Unlike a normal camera, the DVS works in a frame-less fashion and outputs a continuous stream of independent pixel events when a local difference in lighting is detected. The temporal resolution of the DVS is in the order of microseconds. Since only pixels in motion are reported, the redundant information usually contained in consecutive frames do not have to be processed. While the event-based nature of the DVS makes it complicated to use with classic robotic algorithms and rate-based neural networks, it is a great fit for SNN.

Since simulating the actual dynamics of a DVS is complicated, previously ground truth was obtained by having a real DVS observing a screen [28–30]. In this paper, we simulate the DVS by subtracting consecutive frames of a video stream. Similar frame differencing techniques have already been used for extracting moving objects [31]. After differentiation, we apply a threshold to only retain pixels that have a significant difference in light intensity, positive or negative. Differentiated pixels that have a value higher than a threshold θ are reported as ON-events, while differentiated pixels with value lower than $-\theta$ are reported as OFF-events. After a differentiation, all generated pixel events are reported with the time-stamp of the frame, see Fig. 3.

The threshold value θ is equivalent to the DVS bias, or to the slow light adaptation parameter in biology [32]. By varying the threshold value with respect to the global background luminosity, we can obtain light adaptation. In this paper, the threshold value is set to a constant since the vehicle only navigates in a single environment.

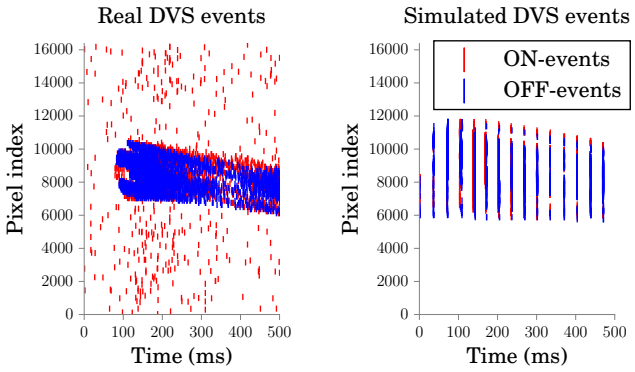
As seen in Fig. 3, the DVS simulation is less noisy than the real DVS but has a much lower temporal resolution. A drawback of this simple approach is the inability to distinguish which address event should be emitted first when many pixels are reported within the same differentiation step. Indeed, events are emitted by batches when a new frame is received (see Fig. 3b). Even if this drawback makes a poor DVS simulation, we argue that a robust SNN should be able to cooperate with such inputs. Certainly, the human visual system can still perceive fluid motion when presented discrete frames with a sufficient frame rate.

This type of encoding conveys motion information, and differs greatly from previous image encoding methods used in literature, which converts pixel values to Poisson spike rates [17]. In biology, it has been accepted that motion is processed separately from color and shape, and is strongly coupled with the pre-motor areas of the cortex [33]. It is therefore a relevant encoding for robotic control tasks.



(a) (b)

Fig. 2: Image provided by the vehicle camera in the simulation and corresponding events generated by the DVS simulation. (a) Image provided by the vehicle camera in the simulation. The camera sees the car hood. (b) Address events generated by the DVS simulation while the car is driving. Red pixels are ON-events, blue pixels are OFF-events.



(a) (b)

Fig. 3: Generated address events of a real DVS and a simulated DVS from webcam images. Both cameras are observing the same stimuli of a ball entering the field of view. The 2 dimensional image structure is flattened to a 1 dimensional pixel index. ON-events are drawn in red, OFF-events are drawn in blue. (a) Address events generated by a real DVS. The events are all independent and have individual time-stamps. (b) Address events generated by our DVS simulation from webcam video stream. The events are only generated by batch, at given time-step, when a new frame is received.

IV. SPIKING NEURAL NETWORK

In this section, we describe our spiking neural network which controls the vehicle end-to-end and how it interfaces with the simulation environment. Spiking neural networks only communicate with spikes. It is therefore required to encode robotic sensory input to input spikes, and decode output spikes to motor commands.

A. Network architecture

The complete network architecture is visualized in Fig. 4. The network is feed-forward and consists of 16 neurons

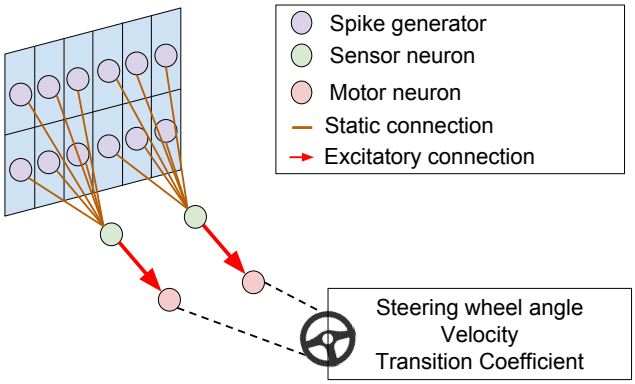


Fig. 4: The feed-forward spiking neural network controlling the vehicle end-to-end. Spike generators propagate spikes from visual stimuli to sensor neurons. They are responsible for specific areas in the provided image and connect to left or right sensor neurons, with the weights described in Fig. 5. Sensor and motor neurons are leaky integrate-and-fire implementing a Braitenberg. Motor neurons are used to decode steering angle (Section IV-C), velocity (Section IV-D), and dynamic transition coefficient (Section IV-E).

with static weights. It has 12 input spike generator neurons, which propagate spikes from the DVS simulation, and 4 leaky integrate-and-fire neurons.

The spike generators have equally sized receptive fields and taken together cover the whole image from the vision sensor. A spike generator will emit a spike for each address event in the covered pixel array. Spike generators are divided into two groups that are connected to two left and right sensor neurons respectively. The weights of these connections are static and are tuned to balance sensory input to stay on the right lane of the road (see Fig. 5). Specifically, we gave more weights to the pixels on the bottom and extremities of the image since they represent lane markup that is closer and on the side. The reason why the weights are not symmetric is because the middle lane marker is discontinuous and thinner than boundary lane markers on the surface of the road.

The two sensor neurons are connected one-to-one to the two motor neurons with excitatory weights. This part of the network can be interpreted as a Braitenberg vehicle with an aggressive connection pattern, driving towards sensor input (see Fig. 6). In a classic Braitenberg setup, the activity of the motor neurons is used to control the wheels of the vehicle. In this paper, we use the activity of the motor neurons to contract virtual muscles attached to the steering wheel (see Fig. 7). When the vehicle enters a turn, the sensory input will be stronger in the direction of the turn. Therefore, the motor neuron on the side of the turn will have a stronger activity than the other, yielding the vehicle to take the turn.

B. Encoding and decoding

Since spiking neural networks only communicate with spikes, it is required to encode robotic sensory input to input spikes, and decode output spikes to motor commands. For

Fig. 5: Representation of the weights of the static connections between spike generators and sensor neurons. The left pixels represent connection weights to left sensor neuron, and the right pixels represent connection weights to the right sensor neuron. The reason why the weights are slightly asymmetrical is to compensate for the thinner discontinuous middle lane markup. The weights are $\left(\begin{smallmatrix}25 & 20 & 0 & 0 & 20 & 25\\ 70 & 5 & 0 & 0 & 20 & 70\end{smallmatrix}\right)$.

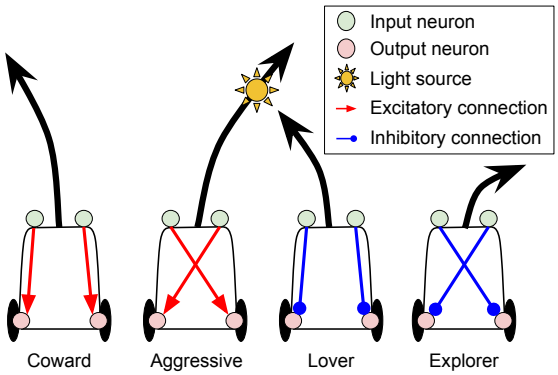


Fig. 6: A Braitenberg-vehicle in its most simple form is made with a network of two light-sensitive sensor neurons and two motor neurons controlling the wheels. The behavior of the Braitenberg-vehicle can be changed based on the wiring of the network. Excitatory connections increase wheel speed whereas inhibitory connections decrease it. In this work we use an *aggressive* connection pattern. Image adapted from [34].

the encoding part, each spike generator is responsible for a region of the pixel array of the DVS simulation. Each event (both ON and OFF) in this region of pixels is transformed into a spike in the respective spike generator at the same time-stamp. The spike generators cover the whole pixel array as seen in Fig 4.

The activity of a neuron is decoded by normalizing the spike-rate of the last simulation step. Let n^i_{spikes} be the number of spikes of neuron i at time t during simulation time T,

$$n^i_{\text{spikes}} = |\{\, \tau \mid \tau \in F_i, \quad t - T < \tau \leq t \,\}| \qquad (1)$$

with $F_i = \{\tau^1, \tau^2, ..., \tau^n\}$ the spike times of neuron i.

By knowing the simulation time T and the refractory period of the neuron τ_{refrac}, the neuron's activity can be normalized and decoded as

$$n_{\text{spikes}} = \frac{n^i_{\text{spikes}}}{n^{\max}_{\text{spikes}}} \in [0,1] \quad \text{where} \quad n^{\max}_{\text{spikes}} = \frac{T}{\tau_{\text{refrac}}}. \qquad (2)$$

In this paper, the spiking neural network is simulated for a time interval of $T = 20\,\text{ms}$, and the refractory period of all neurons is set to $\tau_{\text{refrac}} = 1\,\text{ms}$.

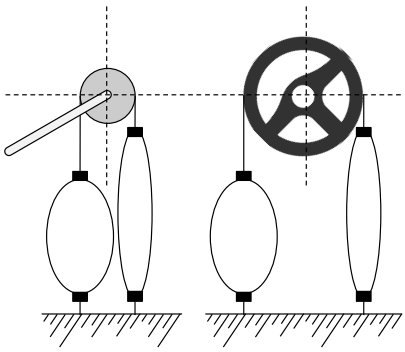


Fig. 7: Muscle modeling for single joint actuation used to actuate the steering wheel.

In the next sections, we refer to l_{motor} and r_{motor} as the normalized activity of the left and right motor neuron respectively,

$$\begin{aligned} l_{\text{motor}} &= \text{decode}(n_{\text{spikes}}^{\text{leftmotor}}), \\ r_{\text{motor}} &= \text{decode}(n_{\text{spikes}}^{\text{rightmotor}}) \end{aligned} \tag{3}$$

C. Muscle modeling

In vertebrates, a single joint is actuated by a set of muscles by means of contraction and flexion of the agonist and antagonist muscle respectively. In a simple muscle model the neural signals are sent as spikes from the brain and the amount of spikes regulate the muscle activation. The muscle fibers translate the spikes into movement. A high spiking activity yields a high activation of the muscle fibers, which translates to a strong contraction. Muscles are only able to actively contract themselves and need an external force for extension. This extension is usually produced by another muscle contracting on the other side of the joint (Fig. 7). This is called the agonist antagonist principle. The steering wheel of the car was modeled using the agonist and antagonist principle as a joint actuated by two muscles and each muscle is driven by the two motor neurons described in Fig. 4. To turn right the muscle on the right side of the steering wheel has to be actuated and vice-versa to go left. Our framework is agnostic to any type of muscle model (see Fig. 1) and more complex models can be integrated in the future.

Formally, the left motor and right motor neurons can be seen as a couple of agonist and antagonist muscles. The steering angle s_t is therefore

$$s_t = k \cdot a_t \quad \text{where} \quad a_t = (l_{\text{motor}} - r_{\text{motor}}) \in [-1, 1], \tag{4}$$

with a constant scaling factor defining the range of the angle which can be spanned in the simulation.

D. Curve braking

Additionally to control the steering angle of the vehicle, the motor neurons can also be used to regulate its speed. We set the speed of the vehicle to be inversely proportional to the steering angle. This has the effect to slow down the vehicle in curves.

Formally, we have

$$v_t = -|a_t| \cdot (v^{\max} - v^{\min}) + v^{\max}, \tag{5}$$

with $v^{\max}$ the maximum speed of the car and $v^{\min}$ the minimum speed of the car. When $a_t = 0$, the car drives straightforward and therefore $v_t = v^{\max}$. the speed of the car decreases linearly with $|a_t|$.

E. Smooth transition

Since roads are generally continuous, steering angles and speed of a car intending to stay on the road should also be continuous. A robust algorithm for driving on the road should therefore have memory of its past motor commands.

Usually, memory in neural networks is achieved with recurrent connections. In this paper, we simply merge the current command sent by the network with the previous command. Specifically, the final motor command (both steering angle and speed) sent to the car is a weighted sum between the current decoded motor command and the previous motor command:

$$\text{ctrl}_t = c \cdot (s_t, v_t) + (1 - c) \cdot \text{ctrl}_{t-T} \tag{6}$$

We compute the coefficient of the weighted sum dynamically at every time-step based on the activity of the motor neurons:

$$c = \sqrt{\frac{l_{\text{motor}}^2 + r_{\text{motor}}^2}{2}}. \tag{7}$$

The dynamic computation of this coefficient has two advantages. First, the current motor command has more weight when there is a lot of visual information. Second, when there is no visual information, the network will still apply previous steering commands instead of driving straight. This allows the car to recover when it drives off the road. The benefit of this dynamical smooth transition is evaluated in Section V against a constant coefficient.

V. EVALUATION

In this section we evaluate the whole system and the performance of the SNN on different circuits. Specifically, we show that the car stays on the right lane of the road in simple situations when driving at a speed of up to 12km/h.

A. Experimental setup

To evaluate the car's driving we designed circuits with different layouts and generated lanelets [27] to provide a ground truth in regard to the car's position and orientation (see Fig. 1).

First, we evaluate the car on a simple closed-loop circuit at three different maximum speeds $v^{\max}$: slow (9km/h), medium (12km/h) and fast (15km/h). The minimum speed of the car for all simulations is set to $v_{\min} = 2$km/h, see Equation 5. The medium speed is then evaluated on the same circuit with two different setups: with a constant transition coefficient $c = 0.3$ (see Section IV-E), and without curve braking (see Section IV-D). We then evaluate the medium speed on

two more complex circuits involving X-shaped and T-shaped intersections.

The spiking neural network is simulated for time intervals of $T = 20\,\mathrm{ms}$, and the refractory period of all neurons is set to $\tau_{\mathrm{refrac}} = 1\,\mathrm{ms}$.

We use two metrics to monitor the performance of the network: distance to the center of the right lane, and the angle between current orientation of the car and the right lane. Both of these metrics are calculated with lanelets.

The simulated camera is attached to the top of the car looking down to the road. Like the DVS, it has a resolution of 128x128 pixels. However, the frame-rate is set to 30fps which is much slower than the latency at which the real DVS delivers events. The camera images are pre-processed with the DVS simulation (see Section III-B) and information between the SNN and the sensors/actuators is encoded and decoded as described in Section IV-B. The system used for these experiments was a common consumer-grade desktop PC running Kubuntu 14.10 (64bit) with 15G of RAM. and the simulation was ran in Gazebo 6.

B. Results

As seen in Fig. 8, the car manages to stay on the right lane of the road for more than 5 laps at slow and medium speed. At higher speed (15km/h), the car eventually crosses the middle lane, and crashes during its second lap. Both with constant transition coefficient (see Section IV-E) and without braking in curve the car crashes during its second lap at medium speed.

The car does not drive straight on the road but rather wiggles from one side of the lane to the other, see 9. This wiggling is caused in part by the discontinuous middle lane markup and intensifies with respect to the speed of the car. Additionally, this can cause the spike generators at the bottom corners to lose sight of the lane markings altogether, especially in or at the beginning of a curve. At higher speeds, this might result in the car leaving the road. This is observed when comparing the spike-trains of the generators before a failure in Fig. 11a and Fig. 11b. Indeed, these spike generators are crucial due to their strong weights in the feature layer (Fig. 5). Potential solutions to this limitation are discussed in Section VI.

For more complex circuits involving intersections, the car drives off the road even when driving at medium speed. Indeed, when reaching the X-shaped intersection the car drives off the road by attempting to average the two road possibilities, (see Fig. 10a). It manages to recover for an additional lap thanks to the dynamic computation of the transition coefficient (IV-E). Indeed, despite that the car drives off the road and has no more visual inputs, it continues steering back to the road. For the T-shaped intersection, the car also tries to average the road possibilities and crashes before a single lap is completed.

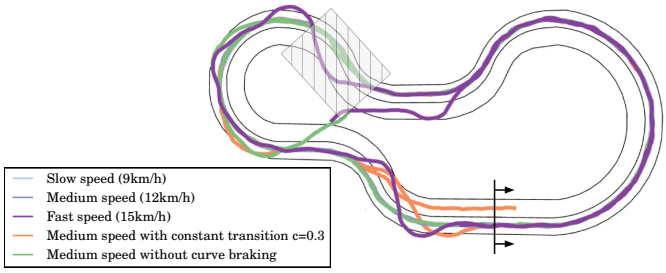


Fig. 8: Positions of the car on a road without intersections. The car drives at 3 different maximum speeds: slow (9km/h), medium (12km/h) and fast (15km/h). Both dynamic transition (see Section IV-E) and curve braking (see Section IV-D) are enabled. In the last two simulations, the car drives at medium speed with a constant transition coefficient of $c = 0.3$ (see Equation 7) and without curve braking. The car manages to stay on the right lane of the road for all configurations except at high speed, or at medium speed without braking in curves.

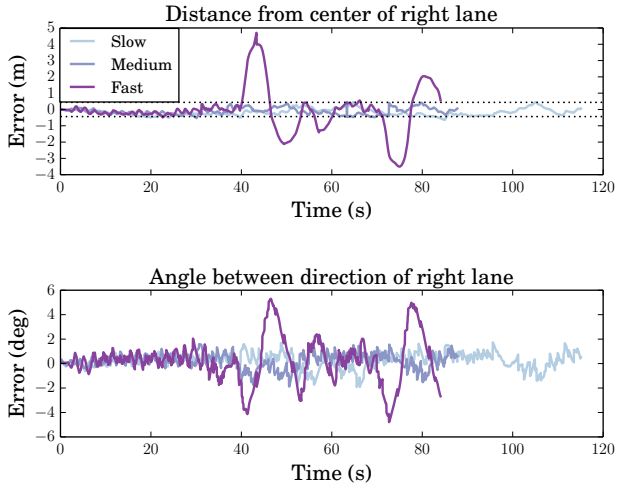
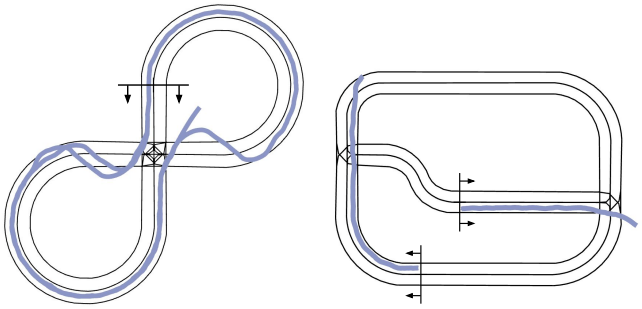


Fig. 9: Metrics to evaluate the performance of the car during 1 lap on the circuit without intersections. The top plot is the distance in meters from the car to the center of the right lane. The dotted lines represent the boundary of the right lane. The bottom plot is the angle in degrees between the orientation of the car and the direction of the right lane. These two metrics are computed from the lanelet map. For slow and medium speed, the car stays within the lane but wiggles. It completes the lap approximately at the same time both with medium and fast maximum speed. In the latter, the car drives off the road a few times and brakes more often.



(a) (b)

Fig. 10: Different roads to test car behavior driving at medium speed. (a) Positions of the car on a road with a X-shaped intersection. (b) Positions of the car on a road with a T-shaped intersection.

VI. DISCUSSION

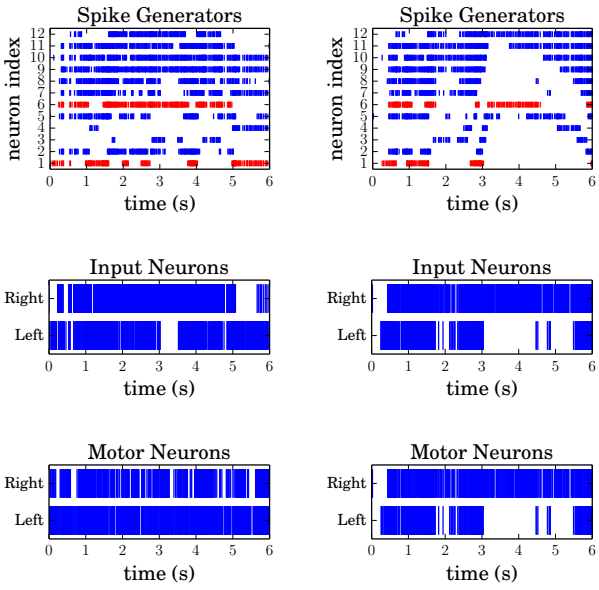
The contribution of this papers is twofold. Firstly, we presented a modular framework to evaluate neural self-driving vehicle applications. It enables us to define experiments utilizing the ground truth lane position provided by lanelets [27]. Using this framework, we designed a neurorobotics experiment for right lane following. Novel biologically inspired pre-processing based on the DVS is used to encode visual information into spikes. To decode spiking activity to driving commands, we used a simple agonist antagonist muscle model.

Secondly, we presented a simple SNN which controls the car end-to-end. This novel feed-forward network can solve simple circuits in our experiments, despite having hand-crafted features. Unlike current research in SNN, we start by presenting an application before deriving new learning rules. Indeed, this simple network admits many limitations:

- **L.1** It does not drive straight but wiggles from one side of the lane to the other;
- **L.2** It does not work for T-intersections;
- **L.3** It manages to follow the road only at moderate speed;
- **L.4** It is not robust to any visual disturbances (both on-road and off-road);
- **L.5** It is not possible to choose a desired direction at an intersection.

We argue that a network with more neurons could overcome these limitations. The wiggling problem **L.1** is caused in part by the fact that the middle lane is discontinuous. To ensure that the method does detect the middle lane markers when they are distant, we should increase the spatial and temporal resolution. Currently, spatial resolution is limited by the small number of spike generators, each having a big receptive field over the 128x128 pixel array of the simulated DVS. Increasing the number of spike generators would require a new tuning of the weights. Temporal resolution would be increased with a higher camera frame rate or a real DVS, which would allow the car to drive at higher speed.

To solve the other limitations, a substantial number of neurons should be added. However, it becomes non-trivial to manually tune the weights of such network so that it would achieve the desired behavior. Instead, the weights should be learned with respect to a learning rule. Specifically, in a reinforcement learning setup, one could use the error metrics given by the lanelets presented in Section III-A to define an objective function. By deriving the appropriate learning rule, the network would adjust its weights to minimize this objective function. Such setup has recently proven very successful with complex networks of rate-based neurons [35]. Minimizing such objective function would make the SNN more robust to complex circuits and solve Limitation **L.2**. Adding a term penalizing slow speed in this objective function would solve Limitation **L.3**. Choosing a direction at intersections could also be encoded in the objective function to solve Limitation **L.5**. Moreover, Limitation **L.4** could be solved by training the network on circuits with visual disturbances.



(a) (b)

Fig. 11: Spike-trains of all neurons during a 800 ms time period, when the car drives on the strong curvature of the circuit without intersection. The indexes of the spike generators are mapped to the image in the following way: $\left(\begin{smallmatrix}7&8&9&10&11&12\\1&2&3&4&5&6\end{smallmatrix}\right)$. The neuron 1 and 6 (in red) are responsible for the bottom corners and have strong weights, see Fig 5.
(a) Medium maximum speed (12km/h).
(b) Fast maximum speed (15km/h).

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement n.720270 (Human Brain Project SGA1).

REFERENCES

[1] W Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] H. Paugam-Moisy and S. Bohte, "Computing with Spiking Neuron Networks," *Handbook of Natural Computing*, pp. 335–376, 2012.

[3] W. Maass, "Liquid State Machines: Motivation, Theory, and Applications," *Computability in Context: Computation and Logic in the Real World*, pp. 275–296, 2010.

[4] P. U. Diehl, G. Zarrella, A. Cassidy, *et al.*, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," *arXiv preprint arXiv:1601.04187*, 2016.

[5] J. Mahler, F. T. Pokorny, B. Hou, *et al.*, "Dex-Net 1.0: A Cloud-Based Network of 3D Objects for Robust Grasp Planning Using a Multi-Armed Bandit Model with Correlated Rewards," *International Conference on Robotics and Automation*, 2016.

[6] A. Giusti, J. Guzzi, D. C. Cire, *et al.*, "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," *IEEE Robotics and Automation Letters*, vol. PP, no. 99, pp. 1–1, 2015.

[7] M. N. Galtier and G. Wainrib, "A Biological Gradient Descent for Prediction Through a Combination of STDP and Homeostatic Plasticity," *Neural computation*, vol. 25, no. 11, pp. 2815–2832, 2013.

[8] M. Schiess, R. Urbanczik, and W. Senn, "Somatodendritic Synaptic Plasticity and Error-backpropagation in Active Dendrites," *PLoS Computational Biology*, vol. 12, no. 2, pp. 1–18, 2016.

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 dB 15µs Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[10] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[11] A. P. Davison, "PyNN: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, no. January, p. 11, 2008.

[12] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[13] M. Zofka, F. Kuhnt, R. Kohlhaas, *et al.*, "Simulation framework for the development of autonomous small scale vehicles," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, IEEE, 2016, Forthcoming.

[14] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems*, IEEE, vol. 3, 2004, pp. 2149–2154.

[15] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[16] M. Giulioni, X. Lagorce, F. Galluppi, *et al.*, "Event-based computation of motion flow on a neuromorphic analog neural platform," *Frontiers in neuroscience*, vol. 10, 2016.

[17] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity.," *Frontiers in computational neuroscience*, vol. 9, no. August, p. 99, 2015.

[18] A. Ambrosano, L. Vannucci, U. Albanese, *et al.*, "Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform," in *Conference on Biomimetic and Biohybrid Systems*, Springer, 2016, pp. 16–27.

[19] E. Falotico, L. Vannucci, A. Ambrosano, *et al.*, "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform," 2016.

[20] A. Bouganis and M. Shanahan, "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2010, pp. 1–8.

[21] C. Richter, S. Jentzsch, R. Hostettler, *et al.*, "Scalability in neural control of musculoskeletal robots," *arXiv preprint arXiv:1601.04862*, 2016.

[22] D. Probst, W. Maass, H. Markram, *et al.*, "Liquid computing in a simplified model of cortical layer iv: Learning to balance a ball," in *International Conference on Artificial Neural Networks*, Springer, 2012, pp. 209–216.

[23] H. Burgsteiner, "Training networks of biological realistic spiking neurons for real-time robot control," in *Proceedings of the 9th international conference on engineering applications of neural networks, Lille, France*, 2005, pp. 129–136.

[24] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to End Learning for Self-Driving Cars," *arXiv:1604*, pp. 1–9, 2016.

[25] S. Mafrica, A. Servel, and F. Ruffier, "Towards an automatic parking system using bio-inspired 1-d optical flow sensors," in *International Conference on Vehicular Electronics and Safety (ICVES)*, IEEE, 2015, pp. 96–103.

[26] S. Mafrica, S. Godiot, M. Menouni, *et al.*, "A bio-inspired analog silicon retina with michaelis-menten auto-adaptive pixels sensitive to small and large changes in light," *Optics express*, vol. 23, no. 5, pp. 5614–5635, 2015.

[27] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Intelligent Vehicles Symposium Proceedings*, IEEE, 2014, pp. 420–425.

[28] E. Mueggler, B. Huber, and D. Scaramuzza, "Event-based , 6-DOF Pose Tracking for High-Speed Maneuvers," IEEE, 2014.

[29] G. Orchard, A. Jayawant, G. K. Cohen, *et al.*, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, no. NOV, pp. 1–15, 2015.

[30] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details," *Frontiers in Neuroscience*, vol. 9, no. December, pp. 1–10, 2015.

[31] N. Singla, "Motion Detection Based on Frame Difference Method," *International Journal of Information & Computation Technology*, vol. 4, no. 15, pp. 1559–1565, 2014.

[32] J. M. Valeton, "Photoreceptor light adaptation models: An evaluation," *Vision Research*, vol. 23, no. 12, pp. 1549–1554, 1983.

[33] N. Kruger, P. Janssen, S. Kalkan, *et al.*, "Deep hierarchies in the primate visual cortex: What can we learn for computer vision?" *Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1847–1871, 2013.

[34] G. Tatai and L. Gulyás, "Neural network controlling architectures in autonomous agents," in *Proceedings of the Hungarian National Conference on Agent Based Computing*, 1998.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, no. July, pp. 1–14, 2015.