Neural Networks 121 (2020) 21–36

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle

Zhenshan Bing [a,b], Claus Meschede [b], Guang Chen [c], Alois Knoll [b], Kai Huang [a,*]

[a] School of Data and Computer Science, Sun Yat-Sen University, China
[b] Department of Computer Science, Technical University of Munich, Germany
[c] School of Automotive Studies, Tongji University, China

ARTICLE INFO

Article history:
Available online 9 July 2019

Keywords:
Spiking neural network
End-to-end learning
R-STDP
Lane keeping

ABSTRACT

Building spiking neural networks (SNNs) based on biological synaptic plasticities holds a promising potential for accomplishing fast and energy-efficient computing, which is beneficial to mobile robotic applications. However, the implementations of SNNs in robotic fields are limited due to the lack of practical training methods. In this paper, we therefore introduce both indirect and direct end-to-end training methods of SNNs for a lane-keeping vehicle. First, we adopt a policy learned using the Deep Q-Learning (DQN) algorithm and then subsequently transfer it to an SNN using supervised learning. Second, we adopt the reward-modulated spike-timing-dependent plasticity (R-STDP) for training SNNs directly, since it combines the advantages of both reinforcement learning and the well-known spike-timing-dependent plasticity (STDP). We examine the proposed approaches in three scenarios in which a robot is controlled to keep within lane markings by using an event-based neuromorphic vision sensor. We further demonstrate the advantages of the R-STDP approach in terms of the lateral localization accuracy and training time steps by comparing them with other three algorithms presented in this paper.

1. Introduction

Utilizing robots to carry out complicated tasks with autonomy has been a realistic prospect for the future, e.g. in the fields of unmanned vehicles, social humanoid robots, and industrial inspection. In order to acquire this advanced intelligence and operate in the real-life scenes, robots have to be able to sense their environment with sensors, which usually produce high-dimensional or large-scale data. Nowadays, inspired by the biological nervous system deep learning architectures have become a promising solution, due to their superiorities for processing multi-dimensional non-linear information from training data. Yet, they differ a lot from the brain-like intelligence in both of the structural and functional properties, which make them incompatible with neuroscience findings. Meanwhile, due to their nature of deep architecture and substantial data, training and operating them is energy-intensive, time-consuming, and latency-sensitive. Taking self-driving cars as an example, the overall computation consumes a few thousand watts, as compared to the human brain, which only needs around 20 watts of power (Drubach, 2000). These are considerable disadvantages, especially in mobile applications where real-time responses are important and energy supply is limited.

A possible solution to some of these problems could be provided by event-based neural networks or spiking neural networks (SNNs) that mimic the underlying mechanisms of the brain much more realistically (Bing, Meschede, Röhrbein, Huang, & Knoll, 2018; Kasabov, 2018). In nature, information is processed using impulses or spikes, making seemingly simple organisms able to perceive and act in the real world exceptionally well and outperform state-of-the-art robots in almost every aspect of life (Brette, 2015). SNNs are able to transmit and receive large volumes of data encoded by the relative timing of only a few spikes, which leads to the possibility of very fast and efficient computing, both in terms of accuracy and speed. For example, human brains can perform visual pattern analysis and classification in just 100 ms, despite the fact that it involves a minimum of 10 synaptic stages from the retina to the temporal lobe (Thorpe, Delorme, & Rullen, 2001).

On the other hand, training these kinds of networks is notoriously difficult. The error back-propagation mechanisms commonly used in conventional neural networks cannot be directly transferred to SNNs due to the non-differentiabilities at spike

* Corresponding author.
E-mail addresses: bingzs@mail.sysu.edu.cn (Z. Bing), claus.meschede@tum.de (C. Meschede), guangchen@tongji.edu.cn (G. Chen), knoll@in.tum.de (A. Knoll), huangk36@mail.sysu.edu.cn (K. Huang).

https://doi.org/10.1016/j.neunet.2019.05.019
0893-6080/

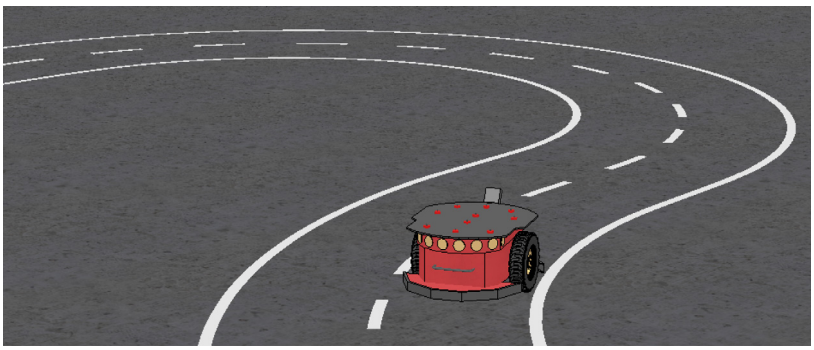


Fig. 1. Robot task: lane keeping.

times. Therefore, there has been a dearth of practical learning rules to train SNNs (Lee, Delbruck, & Pfeiffer, 2016a). Initially, SNN-based control tasks were performed by manually setting network weights, e.g. in Indiveri (1999), Lewis, Etienne-Cummings, Cohen, and Hartmann (2000), and Ambrosano et al. (2016). Although this approach is able to solve simple behavioral tasks, such as wall following (Wang, Hou, Tan, Wang, & Hu, 2009) or lane keeping (Kaiser et al., 2016), it is only feasible for lightweight networks with few connections. On the level of single synapses, experiments have shown that the precise timing of pre- and post-synaptic spikes seems to play a crucial part in the change of synaptic efficacy (Song, Miller, & Abbott, 2000). With this spike-timing-dependent plasticity (STDP) learning rule, networks have been trained in various tasks. For example, Wang constructed a single-layer SNN using proximity sensor data as conditioned stimulus input and then trained it in tasks such as obstacle avoidance and target reaching (Wang, Hou, Lv, Tan, & Wang, 2014; Wang, Hou, Zou, Tan, & Cheng, 2008). However, it is still not clear how the brain assigns credit as efficiently as back-propagation does, even some preliminary research has tried to bridge the gap by combining back-propagation with SNNs Bengio, Mesnard, Fischer, Zhang, and Wu (2017), Bogacz (2017), Lee et al. (2016a), Whittington and Bogacz (2017).

Furthermore, some research has attempted to implement biologically plausible reinforcement learning algorithms based on experimental findings in SNNs. Reward-modulated spike-timing-dependent plasticity (R-STDP) (Florian, 2007; Legenstein, Pecevski, & Maass, 2008a, 2008b), which is a learning rule that incorporates a global reward signal in combination with STDP, has recently been a research focus. This approach intends to mimic the functionalities of those neuromodulators which are chemicals emitted in human brain, e.g. dopamine. Therefore, R-STDP can be very useful for robot control, because it might simplify the requirements of an external training signal and leads to more complex tasks.

However, practical robotic implementations based on R-STDP are rarely found due to its complexity in feeding sensor data into SNNs, constructing and assigning the reward to neurons, and training the SNNs. Specifically, typical sensor data is time-based, such as data from proximity sensor and conventional vision sensor, rather than event or spike-based. In order to feed the data into an SNN, it has to be converted into spikes somehow. In addition, the reward should be carefully assigned to the SNN, a value that is either too high or too low will make the learning instable. The network weights are critical for learning as well, otherwise the learning process will consume more time or even cause failures.

On the basis of our previous work (Bing et al., 2018), this paper aims to explore the training algorithms for spiking neural networks from two different ends and implement them for end-to-end control in the robotics domain. We conduct our research in four parts. First, we construct a simulated lane scenario and adapt it with different lane patterns for evaluating different algorithms, in which a pioneer robot mounted with a dynamic vision sensor (DVS) (Lichtsteiner, Posch, & Delbruck, 2008) is deployed to perform the task. The DVS directly outputs event-based spikes when there is a change of illumination on the pixel level. Thus, it fits SNNs well due to its spike-based nature and offers some great advantages over traditional vision senors, such as speed, dynamic range, and energy efficiency (Lichtsteiner et al., 2008). Second, in an indirect training setup, a conventional ANN is trained in a classic reinforcement learning setting using the Deep Q-Learning (DQN) algorithm. Afterwards, the learned policy is transferred to train an SNN on a state–action dataset created by collecting data from the RL scenarios using supervised learning. Third, an event-based neural network is constructed using the STDP dopamine synapse model and directly trained by the R-STDP learning rule. The reward given to the SNN is defined for each motor individually as a linear function of the lane center distance. Finally, we compare the training performances of all four networks by running them in the training and testing scenarios.

Our main contributions to the literature are summarized as follows. First, our indirect approach utilizes the learned knowledge from a classical reinforcement learning setting and successfully transfers it into an SNN-based controller. This transition offers a way to quickly build up an applicable spike-based controller on the basis of conventional ANNs in robotics, which can be further executed on a neuromorphic hardware to achieve fast computation. Second, our direct approach trains the SNN with the R-STDP learning rule in a biologically plausible way and demonstrates fast and accurate learning process when taking the advantages of an event-based vision sensor. This approach resembles the neural modulation process, which serves as one of the main functionalities in brains and is responsible for strengthening the synaptic connections and then reinforcing desired behaviors or actions. Third, by comparing the performances of all controllers, we demonstrate the superiorities of the R-STDP approach in terms of the training time steps, lateral localization accuracy, and adaption to unknown challenging environment. Those advantages make this method very suitable for being used in mobile robots applications, which usually require quick learning ability and environmental adaptation.

The remainder of the paper is organized as follows: Section 2 introduces the simulation setups for performing the lane-keeping tasks. Section 3 present the indirect learning method for transferring the policy from DQN to SNN. Section 4 presents the implementation details of the direct training based on R-STDP. In Section 5, the training details of each controller are presented. In Section 6, we provide the experimental results and make a comparison to other controllers. Section 7 summarizes our study and presents the future work.

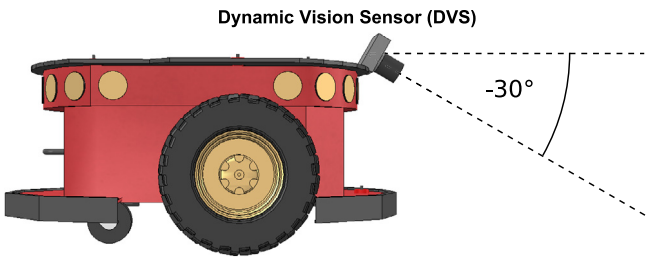


Fig. 2. Pioneer P3-DX robot with dynamic vision sensor (DVS).

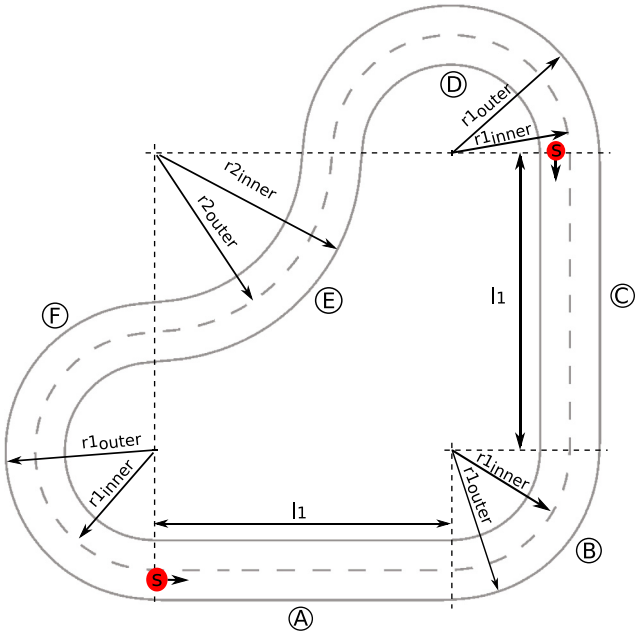
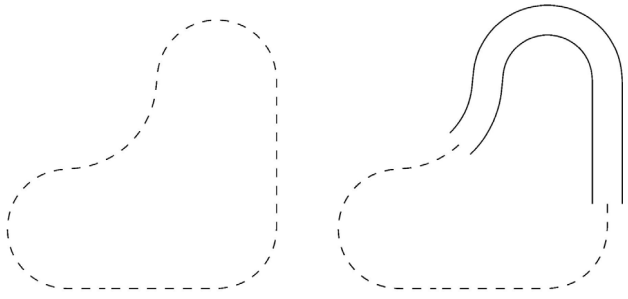


Fig. 3. Scenario 1: The simple lane-keeping scenario consists of a road with two lanes (inner and outer) and six different sections A, B, C, D, E, and F. Starting positions are marked with *S*. Dimensions: $r1_{inner} = 1.75$ m, $r2_{inner} = 3.25$ m, $r1_{outer} = 2.25$ m, $r2_{outer} = 2.75$ m, $l_1 = 5.0$ m.



(a) Scenario 2 (b) Scenario 3

Fig. 4. (a) Scenario 2: Single lane pattern without boundaries. (b) Scenario 3: Lanes with two different patterns.

2. Lane-keeping tasks

In order to provide a simple and flexible environment to test and compare different algorithms, simulated lane-keeping tasks with different lane patterns for a Pioneer robot are set up using Virtual Robotics Experiment Platform (V-REP) (Rohmer, Singh, & Freese, 2013) (see Fig. 1). All the sensor messages and motor commands between the simulator and the neural networks are transmitted via ROS (Quigley et al., 2009).

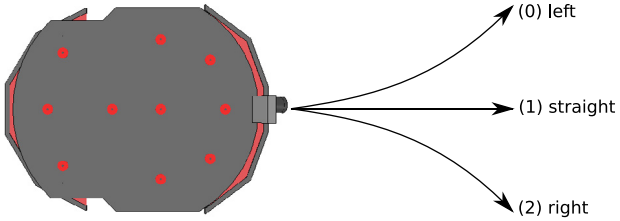


Fig. 5. Action space in lane-keeping task with three discrete actions: (0) Turn left: Set left motor speed to $v_s - v_t$ and right motor speed to $v_s + v_t$. (1) Go straight: Set left and right motor speed to v_s. (2) Turn right: Set left motor speed to $v_s + v_t$ and right motor speed to $v_s - v_t$.

Instead of using the on-board ultrasonic sensors, a DVS camera is attached to the front of the robot with a 30° depression angle as shown in Fig. 2. For further validating the effectiveness and adaptability of the proposed algorithms, three scenarios with different lane patterns are shown in Fig. 4. The first scenario in Fig. 3 consists of a closed loop course with a two-lane road. The road is comprised of two solid lines and a uniformly dashed line in the middle. From the starting position onwards, the outer lane can be divided into six sections: (*A*) *straight*, (*B*) *left*, (*C*) *straight*, (*D*) *left*, (*E*) *right*, (*F*) *left*. During each episode in the training, the robot will switch the start position and moving direction between inner and outer lane at each reset. Therefore, it will experience both left and right turns equally and with different radii as well.

Based on the same layout and dimensions, a second scenario was implemented, testing the algorithms on a different road pattern where the left and right solid lines are missing (see Fig. 4(a)). In a third scenario, two different road patterns have to be learned in parallel (see Fig. 4(b)).

3. Indirect learning based on DQN

In this section, we will first solve the lane-keeping tasks using a classic Deep Q-Learning (DQN) reinforcement learning algorithm. Then we will introduce the indirect training method by transferring the learned policy from the DQN to an SNN within the framework of supervised learning. All the simulation parameters can be found in the tables in the appendix.

3.1. Lane keeping as MDP

Reinforcement learning tasks are usually described as a Markov decision process (MDP), which is defined as a 5-tuple of actions, states, transition probabilities, rewards, and discount factor. While the transition probabilities can be ignored when using model-free algorithms such as Q-learning, other components of the MDP have to be carefully chosen to ensure fast and stable learning.

Fig. 5 shows three discrete actions that the robot can take for these tasks. It can go straight, letting left and right motors run at the same speed, or it can take a turn by adding or subtracting speed to both motors depending on the desired moving direction.

3.2. DVS input generation

In similar reinforcement learning tasks using conventional cameras (Lillicrap et al., 2015; Mnih et al., 2015), scaled images could be directly used as state input for the MDP; this is more difficult when using a DVS device. Dynamic Vision Sensor, as an emerging neuromorphic sensor, generates sparse, event-based

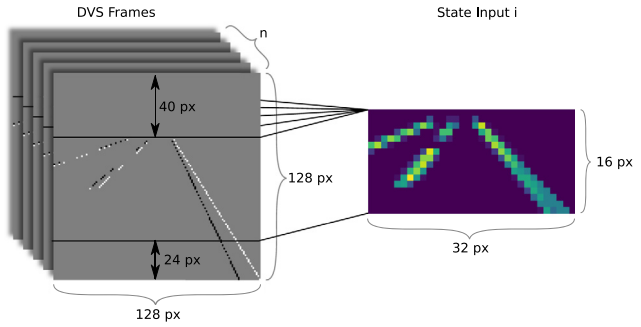


Fig. 6. Conversion of consecutive DVS frames into state input for reinforcement learning. This is done by dividing the original 128 × 128 DVS frames into small 4 × 4 regions and counting every event over consecutive frames regardless of the polarity. Furthermore, the image is cropped at the top and bottom, resulting in a 32 × 16 image.

output that represents the positive and negative relative luminance change of a scene. Due to its advantages, such as speed, dynamic range, and energy efficiency (Lichtsteiner et al., 2008), DVS is used in this study to detect the lane marks and generate spikes. First, in order to decrease the computational complexity of the task, images are reduced to a lower resolution as well. Second, due to the event-based nature of the DVS data, image frames coming from the simulation do not always contain sufficient information for the network to make meaningful decisions. Therefore, the state input is computed by condensing information from several consecutive DVS frames into a single image. To be clear, the DVS frame means the events accumulated in the 50 ms interval instead of being the whole image frame as traditional vision sensor. As shown in Fig. 6, this is done by dividing the original 128 × 128 DVS frames into small 4 × 4 regions and counting every event over consecutive frames regardless of the polarity. Furthermore, the image is cropped at the top and bottom, resulting in a 32 × 16 image. To further increase the performance, the final DQN state input $s_{M\times N}$ is a binary version of the state input $i_{M\times N}$, only containing ones and zeros:

$$s_{M\times N} = \begin{cases} 0 & \text{if } i_{M\times N} = 0 \\ 1 & \text{if } i_{M\times N} > 0 \end{cases} \tag{1}$$

In the simulation, DVS frames are calculated and published every 50 ms (with every simulation time step). Actions are executed every 500 ms. Therefore, during one action step, DVS frames are stored in a first-in first-out (FIFO) queue of length 10, and the last ten DVS frames are then converted into the final state input.

3.3. Reward generation for DQN

Rewards play a crucial role in reinforcement learning and define the goal of an agent. In this research study, the robot is supposed to learn to follow a lane staying as close to the center as possible. Fig. 7 shows the definition of the reward that is given at every time step of the MDP. It is defined as a Gaussian distributed function of the lane center distance. As the model-free DQN algorithm learns from experience samples with a one-step lookahead, it is beneficial for learning to use a reward that is well distributed over the state space and monotonically increasing towards the goal. This ensures that the robot will learn to navigate in the direction of the goal, even if it has not been there yet. Besides DVS data, the simulator publishes position data of the robot every 50 ms as well. With a mathematical model of the lane center in both directions, this data is used to calculate the exact distance of the robot to the lane center and the resulting reward.

If the robot reaches a position where its distance to the lane center is greater than 0.5 m, training episodes are terminated and a reset message is generated that causes the simulator to place the robot at its starting position on the opposite lane. Letting the robot alternate between both lane directions increases its experienced states and results in a more generalized policy after learning. In reinforcement learning, the extent to which an agent takes expected future rewards into account is usually controlled by a discount factor. Although the lane-keeping task does not necessarily require looking ahead many steps, the discount factor is set to 0.99, therefore potentially being able to solve tasks that involve some foresight as well.

3.4. DQN-based controller

A fully connected feedforward network architecture using rectified linear units (ReLU) as activation function was chosen, inspired by similar work (Diehl, Neil, Binas, Cook, Liu, & Pfeiffer, 2015a; Lee, Delbruck, & Pfeiffer, 2016b). The network takes the binary state image as input, resulting in $32 \times 16 = 512$ input neurons. It consists of two hidden layers with 200 neurons each and three output neurons representing the discrete actions. Training is performed using the stochastic optimization algorithm Adam.

Fig. 8A shows a detailed flow chart of the DQN algorithm. There are two networks being used in the DQN algorithm, which share the same architecture but with different weight parameters. The action-value network $\mathcal{Q}$ is used to determine the action with the highest Q value, of which the weights are updated every step. The target action-value network $\hat{\mathcal{Q}}$ is used to predict the *q_target* value, of which the weights will be updated every several steps by assigning the weights as $\hat{\mathcal{Q}} = \mathcal{Q}$. More details about the DQN algorithm itself can be found in Van Hasselt, Guez, and Silver (2016). At the beginning, the action-value network $\mathcal{Q}$ is initialized with random weights and copied to the target action-value network $\hat{\mathcal{Q}}$. Each episode of the training procedure begins with a reset of the robot to its starting position, switching lanes after each episode. Hereby, the initial state input is a vector of zeros. At each time step, actions are chosen from $\mathcal{Q}$ following an ϵ-greedy policy. This means that with a probability of $\epsilon \in [0, 1]$

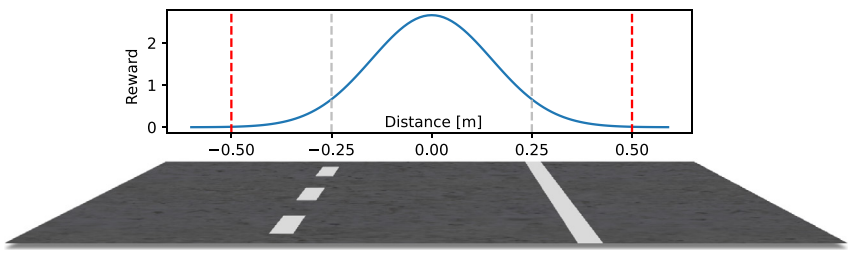


Fig. 7. Reward given in the lane-keeping task. It is defined as a Gaussian distribution over the distance to the center of the lane with a standard deviation of $\sigma = 0.15$ and mean at 0. The lane markings are 0.25 m away from the lane center. If the robot goes further than 0.5 m from the lane center, episodes are terminated and the robot is positioned at its starting position.

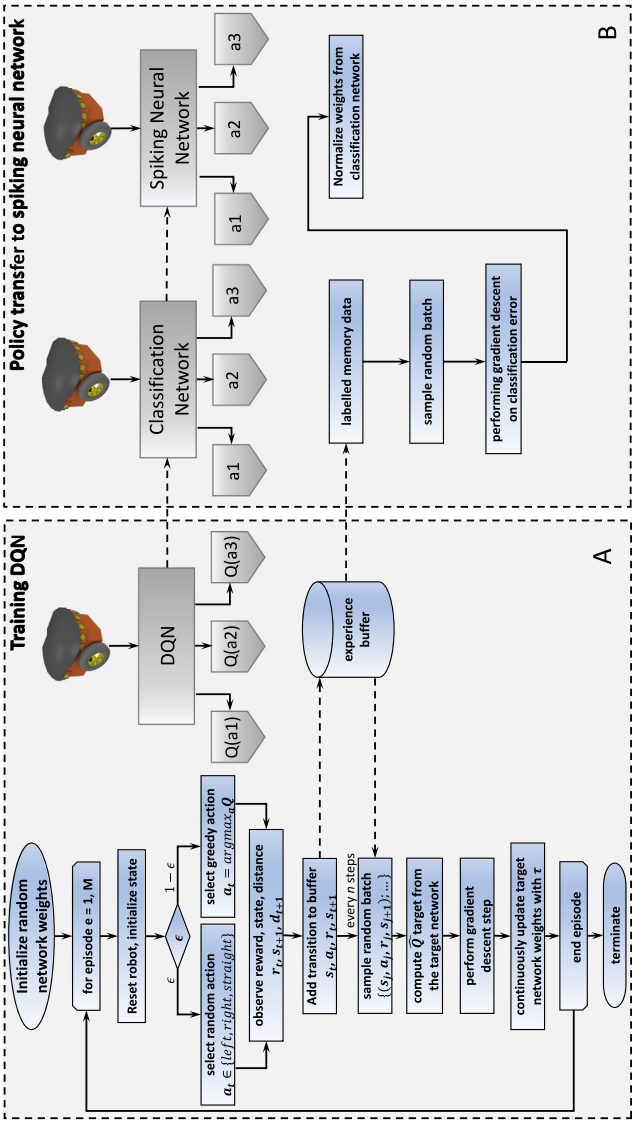


Fig. 8. A. Flowchart of the DQN algorithm. B. Flowchart of the policy transfer from DQN to SNN.

the agent will randomly select an action. Otherwise, it will select the action with the highest action value. At the start, ϵ is set to 1, ensuring pure exploratory behavior. After predefined 1000 time steps, ϵ then linearly decreased to its end value close to zero.

Chosen actions a_t are sent to the environment handler, which will communicate with the simulator to acquire the reward r_t, next state image s_{t+1}, and the distance to the lane center d_t. Moreover, each transition (s_t, a_t, r_t, s_{t+1}) is stored in the experience buffer. Every n steps, the actual training step is performed by randomly sampling transitions from the experience buffer. Using the target action-value network $\hat{\mathcal{Q}}$ for calculating the updating targets, the loss function is then constructed in order to perform a stochastic gradient descent step on the action network. At the end of each training step, the target action-value network $\hat{\mathcal{Q}}$ weights are gradually updated towards the action-value network $\mathcal{Q}$ weights with $\tau \in [0, 1]$ and $\tau \leqslant 1$ (Lillicrap et al., 2015).

Training episodes will be terminated if the robot goes beyond the maximum distance to the lane center or if the maximum number of steps in an episode is reached. The latter mechanism guarantees that the robot will experience both directions of the road, even if it has already learned a good policy for keeping the lane. The overall training procedure is ended either after reaching a predefined number of episodes or total training steps.

3.5. DQN-SNN based controller

The aforementioned DQN algorithm handles event-based data by storing consecutive DVS frames and batch processing them at every step in the MDP. Clearly, this approach cannot be the ideal mechanism for handling DVS data, as it annihilates some of the advantages that make the sensor powerful in the first place, e.g. its temporal resolution. However, handling data streams is precisely what the SNNs are good at, e.g. from a DVS device, without the need for batch processing. Due to their event-based nature, spikes have to be decoded somehow in order to obtain real values, which makes it very difficult to get network output with similar precision as well as DQN. Moreover, the non-differentiability of spike events makes it very difficult to use a training mechanism such as back-propagation. Therefore, in this sub-section we show how the previously learned DQN policy can be used to create a state–action dataset created by collecting from the RL scenario for training an SNN using supervised learning (Fig. 8B).

To address this problem, ReLU is considered as an activation function of the input stimulus and the firing frequency, rather than the function of the input stimulus and the action potential, since there is a linear relationship between the action potential and the firing frequency. Based on the fact that simple integrate-and-fire (IF) neurons in SNNs behave very similarly to rectified linear units (ReLU) in conventional ANNs, an indirect training method can be used for training:

1. Create the state–action dataset by labeling stored states with corresponding actions.
2. Train conventional ANN with no hidden layer biases and ReLU activation functions.
3. Normalize weights.
4. Transfer weights to SNN with IF neurons and perform control task.

For training ANNs using stochastic gradient descent on the prediction error, the DQN algorithm stores every single transition in the previously discussed experience buffer. This makes it very convenient to use the same data for training the SNN as well. Therefore, first, all stored state images as shown in Fig. 6 are labeled using the pre-trained action network from DQN. It is important to note that the input images $s_{M\times N}$ for previously training the DQN are still used here to train the SNN. Furthermore, the pixel values $i_{m,n}$, describing the number of spike events in the same 4×4 window over consecutive DVS frames, are scaled to $\hat{i}_{m,n} \in [0, 1]$ by dividing every value by the maximum pixel value $i_{max} = max_{j,m,n}(i^j_{m,n})$ in the whole dataset. As a result, the input values can be interpreted as spike firing rates making the network transferable to an SNN.

In the next step, the labeled dataset is used to train a conventional ANN. The fully connected feedforward network consists of an input layer with bias and $32 \times 16 = 512$ input neurons, a hidden layer with 200 neurons, and an output layer with three output neurons. For converting an ANN to an SNN, we would like all the neurons in the hidden layer and the output layer are only effected by the activities of the neurons in the previous layer. Then, we can scale the weights only according to the threshold value, which is used for all the neurons in the same layer and do not have to worry about the bias value for each individual neuron. Therefore, all the neurons in the hidden layer and the output layer are set without bias. After training, the weights can be transferred to an SNN with IF neurons that matches the previous network architecture. There is a possibility that the

Algorithm 1: Model-Based Normalization (Diehl et al., 2015b).

```
for layer in layers do
    max_input = 0
    for neuron in layer.neurons do
        input_sum = 0
        for input_wt in neuron.input_wts do
            input_sum += max(0, input_wt)
        end for
        max_input = max(max_input, input_sum)
    end for
    for neuron in layer.neurons do
        for input_wt in neuron.input_wts do
            input_wt = input_wt / max_input
        end for
    end for
end for
```

inputs will stimulate the hidden neurons, firing immediately in a single simulation time step. In this case, the information from inputs cannot be precisely transmitted and indicated by the firing rate of the hidden neurons, which may cause information loss for the output layer. Therefore, we have to make sure that all the neurons can only fire once in each time step and then ensure minimal accuracy reductions in the SNN with transferred weights. This can be done by scaling weights so that the maximal weighted input to a neuron in each layer is equal to the firing threshold. In other words, the weights are normalized for each layer separately beforehand.

The normalization algorithm is explained in Diehl et al. (2015a) and shown in Algorithm 1. For the first layer, the bias is necessary to handle input vectors consisting of zeros only. Without any bias, the network output would be always zero as well, regardless of its weights. In multi-layered SNNs, external input currents introducing biases to deeper layers are difficult to handle, because they have to be scaled to match the firing rates coming from previous layer neurons. For the first layer though, the bias can be interpreted as an additional input current with a constant firing rate of 1.

The SNN with simple IF neurons is implemented, inspired by Diehl et al. (2015a), which uses a time-step-based approach in order to propagate spikes through the network and update membrane potentials. As mentioned earlier, the simulator publishes DVS data every 50 ms. The data is then scaled to [0, 1], causing Poisson input neurons (Stevens & Zador, 1996) to fire for 50 ms as well. The scaling factor can be roughly estimated by dividing the maximum pixel value v_{max} by the number n of consecutive DVS frames used in the data set. Therefore, if a Poisson neuron fires with its maximum frequency over n simulation steps, it can be interpreted as the maximum firing rate of 1 in the data set.

The information from consecutive DVS frames is propagated through the SNN over time. When a neuron fires and sends a spike to the next layer, it increases the membrane potential of the next layer's neurons. Therefore, information is stored in the membrane potentials and it takes some time to generate output spikes. As a consequence, this means that output spikes are generated sparsely in time, leaving simulation steps with no spike output at all. In order to generate a control signal, even if there are no output spikes during a simulation step, a trace is implemented for each action. The action trace z_t^a accumulates output spikes s_t for each action respectively and decays over time with a factor $c \in [0,\ 1]$. The action a_t with the highest trace value is eventually chosen at every simulation step:

$$z_{t+1}^a = c \cdot z_t^a + s_t \tag{2}$$

$$a_t = \text{argmax}_a(z_t^a) \tag{3}$$

4. Direct learning of SNN with R-STDP

Training a neural network with DQN to learn a policy and transferring the policy to a SNN by creating an labeled state–action dataset is cumbersome, and it introduces some loss in the training process. Furthermore, this approach ignores one of the main strengths that SNNs bring compared to conventional ANNs, which is their ability to take the precise timing of spikes into account and not just the averaged rate. To tackle this problem, an SNN is constructed and trained using R-STDP for steering the robot in the aforementioned lane-keeping tasks.

4.1. R-STDP learning rule

As the most important theory in neuroscience explaining the adaption of synaptic efficacies in the brain during the learning process, the spike-timing-dependent plasticity (STDP) learning rule (Caporale & Dan, 2008) has been successfully proven by neuroscience experiments (Bi & Poo, 1998; Markram, Lübke, Frotscher, & Sakmann, 1997).

For this study, the weight update rule under STDP as a function of the time difference between pre- and postsynaptic spikes is defined as

$$\Delta t = t_{post} - t_{pre} \tag{4}$$

$$W(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+}, & \text{if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, & \text{if } \Delta t < 0 \end{cases} \tag{5}$$

$$\Delta w = \sum_{t_{pre}} \sum_{t_{post}} W(\Delta t), \tag{6}$$

where w is the synaptic weight. Δw is the change of the synaptic weight. t_{pre} and t_{post} stand for the timing of the firing spike from pre-neuron and post-neuron. A_+ and A_- represent positive constants scaling the strength of potentiation and depression, respectively. τ_+ and τ_- are positive time constants defining the width of the positive and negative learning window.

A simple learning rule combining models of STDP and a global reward signal was proposed by Izhikevich (Izhikevich, 2007) and Florian (Florian, 2007). In the R-STDP, the synaptic weight w changes with the reward signal R. The eligibility trace of a synapse can be defined as,

$$\dot{c}(t) = -\frac{c}{\tau_c} + W(\Delta t)\delta(t - s_{pre/post})C_1 \tag{7}$$

where c is an eligibility trace. $s_{pre/post}$ means the time of a pre- or postsynaptic spikes. C_1 is a constant coefficient. τ_c is a time constant of the eligibility trace. δ is the Dirac delta function

$$\dot{w}(t) = R(t) \times c(t) \tag{8}$$

where $R(t)$ is the reward signal. More details on the R-STDP mechanism can be found in Frémaux and Gerstner (2016), Potjans, Morrison, and Diesmann (2010).

4.2. Reward generation for R-STDP

Instead of dividing the input data and feeding it into two separate networks with static weights as was done in Kaiser et al. (2016), a single SNN based on R-STDP is designed as shown in Fig. 9. The input data is scaled and used for excitation of Poisson neurons, in a single network with $8 \times 4 = 32$ input neurons. Then, the input layer is connected to two LIF output neurons in an "all to all" fashion using R-STDP synapses. The reward signal given at each simulation time step is shown in Fig. 10. It is defined for each motor with opposite signs linearly dependent on the robot's

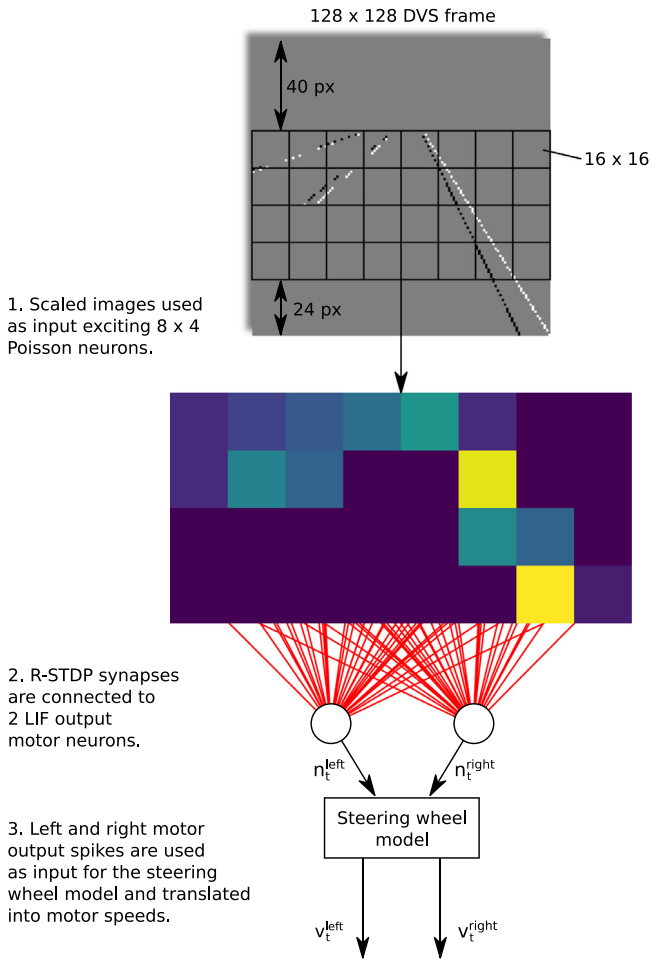


Fig. 9. Network architecture of the R-STDP implementation using DVS frames as input.

distance to the lane center. When the robot is on the right side of the lane center and should turn left to get back, connections that lead the right motor neuron to fire are strengthened, connections that lead the left motor neuron to fire are weakened. Conversely, if the car is on the opposite side of the lane-center, this process is reversed. Over time, the robot should learn to associate certain input stimuli with left or right turns and act accordingly. These considerations lead to the following rewards for left and right motors neuron connections, with d being the distance to the lane center and c_r a constant scaling the reward:

$$r_{left/right} = -/+(d \cdot c_r) \tag{9}$$

4.3. Encoding and decoding

For communicating with robot sensors and motors in SNNs, the sensory information should be encoded into input spikes and the output spikes should be decoded into motor commands. A similar processing procedure for the encoding and decoding can be found in Kaiser et al. (2016). The same model is implemented in this paper with only one change. Instead of steering angles, turning speeds are computed and added or subtracted for the left and right motors. First, the output spike count $n_t^{left(right)}$ is scaled by the maximum possible output n_{max}:

$$m_t^{left(right)} = \frac{n_t^{left(right)}}{n_{max}} \in [0; 1], \text{ with } n_{max} = \frac{T_{sim}}{T_{refrac}}, \tag{10}$$

where T_{sim} denotes the simulation time step length and T_{refrac} describes the refractory period length of the LIF neuron. Based on the difference of the normalized activities m_t^{left} and m_t^{right} and a turning constant c_{turn}, the turning speed is defined as

$$S_t = c_{turn} \cdot a_t, \text{ with } a_t = m_t^{left} - m_t^{right} \in [-1; 1]. \tag{11}$$

Furthermore, in order to ensure a minimum running for the robot, the overall speed is controlled according to

$$V_t = -|a_t| \cdot (v_{max} - v_{min}) + v_{max}, \tag{12}$$

where v_{min} and v_{max} are predefined speed limits. Since controlling a car is generally a continuous process, overall speed and turn speed were smoothened based on the activities:

$$v_t = c \cdot V_t + (1 - c) \cdot v_{t-1}, \tag{13}$$

$$s_t = c \cdot S_t + (1 - c) \cdot s_{t-1}, \tag{14}$$

$$\text{with } c = \sqrt{\frac{(m_t^{left})^2 + (m_t^{right})^2}{2}} \tag{15}$$

Finally, the control signals for the left and right motors are computed by

$$v_t^{left} = v_t + s_t \text{ and } v_t^{right} = v_t - s_t. \tag{16}$$

4.4. Training

In order to train the network successfully, the parameters of the R-STDP controller have to be carefully chosen (see Table 3 in the appendix). First, the training result is closely related to the reward in (9). If the value is too low, the learning will take too much time and it might be difficult to see any progress at all. By contrast, if it is too high, the learning will become increasingly instable and the robot will not learn anything. Second, the initial network weights are critical for learning as well. In this study, weights are initialized uniformly at a relatively low value of 200. The weights have to be larger than zero, because both motor neurons must be excited from the beginning in order to induce

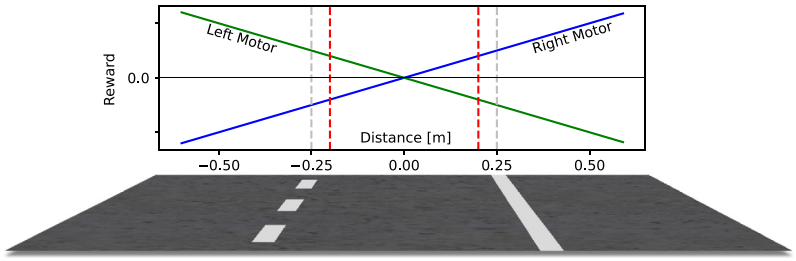


Fig. 10. Reward given by the R-STDP controller: It is defined for each motor individually as a linear function of the lane center distance scaled by a constant c_r. The lane markings are 0.25 m away from the lane center. If the robot goes further than 0.2 m from the lane center, episodes are terminated and the robot is positioned at its starting position.

Table 1
DQN parameters.

DQN	network architecture	512 - 200 - 200 - 3
	connections	fully connected
	batch size	32
	update frequency	4
	soft update	$\tau = 0.001$
	learning rate	$\alpha = 0.0001$
	buffer size	5000
ϵ-greedy policy	pre-training steps	1000
	annealing steps	49000
	random probability start	1.0
	random probability end	0.1
MDP	discount factor	$\gamma = 0.99$
	reset distance	0.5 m
	maximum episode steps	1000
	time step length	0.5 *s*
Robot	motor speed straight	$v_s = 1.0$ m/s
	motor speed turn	$v_t = 0.25$ m/s

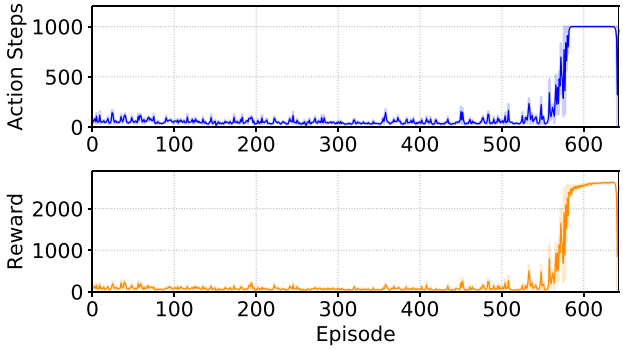


Fig. 11. Scenario 1: After 500 episodes, the robot learns to follow the lane without triggering a reset. Episodes are limited to 1000 action steps. The accumulated rewards collected in 1000 action steps still improve after the robot has reached the step limit.

weight changes following the R-STDP learning rule. Furthermore, the weights are clipped to [0 : 3000], only allowing excitatory synaptic connections.

5. Results

With the lane-keeping tasks in mind, DQN, DQN-SNN, and R-STDP controllers for the Pioneer robot were presented earlier in this paper with regard to the basic principles and implementation details. In this section, the training results of each controller are discussed and their performances are also compared with each other and with the Braitenberg controller from Kaiser et al. (2016).

5.1. DQN training results

Fig. 11 shows the training progress of the DQN algorithm in the first scenario. Training parameters are presented in Table 1. At the beginning, the robot will randomly choose actions regardless of the rewards. Episodes are terminated once these random actions lead the robot beyond the 0.5 m lane-center distance threshold. Therefore, action steps and rewards are randomly distributed at a low level at the beginning. Even though the ϵ-greedy policy constantly increases the chance of choosing the action with the highest action value, the robot does not show any learning effect until episode 400. At around episode 300, action steps and rewards actually decrease, because the robot is following a policy that is not optimal yet. After approximately 580 episodes, the robot has learned to follow the lane without trigging a reset.

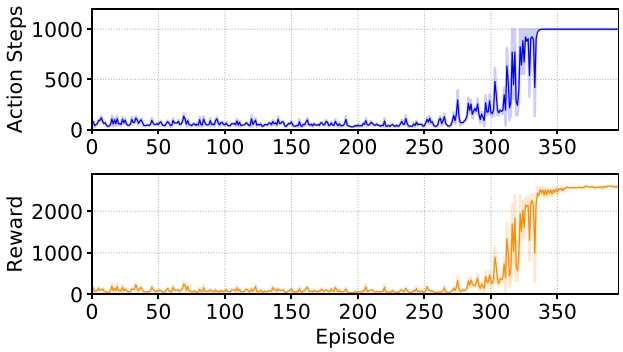


Fig. 12. Scenario 2: Action steps and rewards for each training episode of the DQN controller.

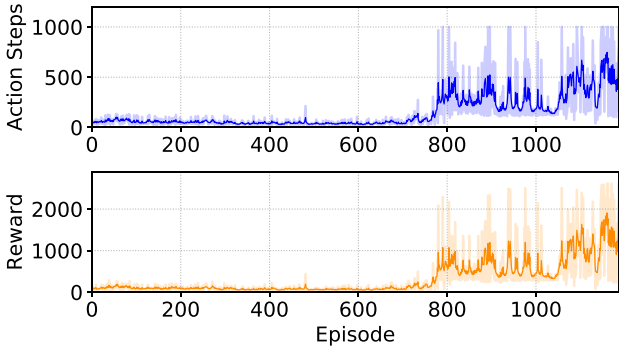


Fig. 13. Scenario 3: The algorithm failed to learn a stable policy after 1,186 episodes and 170,000 steps.

To ensure experiences from both inner and outer lanes, even if the robot has successfully learned to follow them, episodes are also terminated after 1000 action steps (10,000 time steps), since competing a full lap takes around 5000 time steps at the predefined motor speed. After each episode, the robot is placed at the start point of the other side of the road. The accumulative rewards have exceeded 1000 action steps after 580 episodes, and still slowly increase, approaching a reward maximum afterwards.

Similarly, the algorithm learns a control strategy in the second scenario as well. Due to the reduced complexity in the state images, effective learning already begins after 300 episodes (Fig. 12). Interestingly, we can observe that the DQN learns faster compared with the process in the first scenario. The reason is that the right side of the input image does not generate any information due to the missing boundary lane, which leads to less complexity in the network. In the third scenario, by contrast, the DQN algorithm failed to learn a stable policy. Fig. 13 shows the episode lengths and rewards in 170,000 time steps total. The starting positions in the scenario are switched so that the robot experiences both road patterns from the beginning. At around episode 800, the robot learns to follow the lane, even completing laps and reaching the time step limit at times. Unfortunately, it does not learn a generalized policy that works for both lanes. Once the algorithm figures out how to take a turn or a transition section from one pattern to another, it seems to have detrimental effects on its behavior in other situations. Even though the average reward over several episodes increases towards the end, the algorithm never reaches the time step limit in consecutive episodes. Taken together, the algorithm in the third scenario optimizes its behavior but fails to reach a global reward maximum.

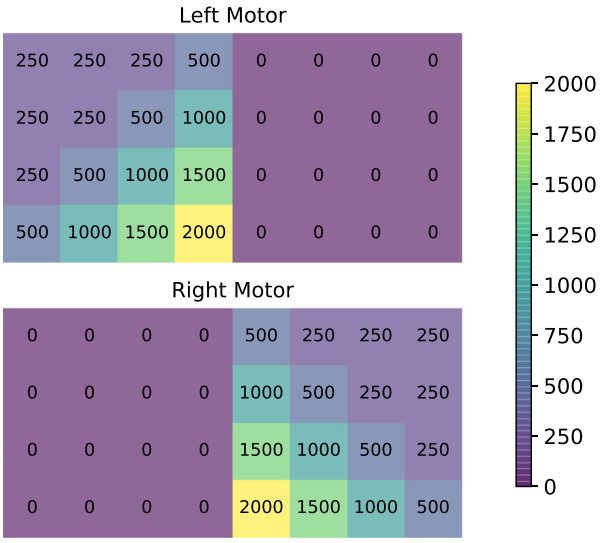


Fig. 14. Scenario 1: Static connection weights to the left and right motor neurons of the Braitenberg vehicle controller.

Table 2
DQN-SNN parameters.

ANN training	network architecture	512 - 200 - 3
	connections	fully connected
	batch size	50
	training steps	10000
	optimizer	ADAM
	learning rate	0.0001
SNN simulation	simulation time	10 ms
	max. firing rate	1000 Hz
	simulation step length	1 ms
	membrane potential threshold	1 mV

Table 3
Simulation parameters specification.

Steering model	max. speed	$v_{max} = 1.5$ m/s
	min. speed	$v_{min} = 1.0$ m/s
	turn constant	$c_{turn} = 0.5$
	max spikes during a simulation step	$n_{max} = 15$
Poisson neurons	max. firing rate	300 Hz
	number of DVS events for max. firing rate	$n = 15$
SNN simulation	simulation time	50 ms
	time resolution	0.1 ms
LIF neurons	NEST model	iaf_psc_alpha
	Resting membrane potential	$E_L = -70.0$ mV
	Capacity of the membrane	$C_m = 250.0$ pF
	Membrane time constant	$\tau_m = 10.0$ ms
	Time constant of postsynaptic excitatory currents	$\tau_{syn,ex} = 2.0$ ms
	Time constant of postsynaptic inhibitory currents	$\tau_{syn,in} = 2.0$ ms
	Duration of refractory period	$t_{ref} = 2.0$ ms
	Reset membrane potential	$V_{reset} = -70.0$ mV
	Spike threshold	$V_{th} = -55.0$ mV
	Constant input current	$I_e = 0.0$ pA
R-STDP synapse	NEST model	dopamine_synapse
	Amplitude of weight change for facilitation	$A_+ = 1.0$
	Amplitude of weight change for depression	$A_- = 1.0$
	STDP time constant for facilitation	$\tau_+ = 20.0$ ms
	Time constant of eligibility trace	$\tau_c = 1000.0$ ms
	Time constant of dopaminergic trace	$\tau_n = 200.0$ ms
	Minimal synaptic weight	0.0
	Maximal synaptic weight	3000.0
	Initial synaptic weight	200.0
	Reward constant	$c_r = 0.01$

5.2. DQN-SNN training results

For the policy transfer from the DQN action network to an SNN, an state–action dataset is created using the state samples stored in the experience buffer. During training of the DQN controller in the first scenario, 98,990 state samples are visited and stored. At the beginning of the training procedure, the robot experiences many states that are far from the optimal lane center position. Once it has learned to follow the lane, however, it will experience only states close to the lane center. These states are much more important for the policy transfer, because the robot controlled by the SNN will likely never see those "poor" states far from the lane center. Therefore, it is important to train the SNN on a dataset with mostly "good" states. This is achieved by letting the robot run and collect states for a while after successfully learning a good policy to ensure a favorable distribution of states in the dataset. Using the previously trained DQN action network, all states are labeled with actions and an ANN is trained reaching a classification accuracy of 93.05%. Further training and network parameters are shown in Table 2. Following that, the network weights are normalized and transferred to an SNN based on Algorithm 1 with the same architecture performing the robot control task. In the second scenario, 100,236 states could be classified with an accuracy of 91.71% following the same procedure. Due to the fact that the DQN algorithm could not successfully learn a stable policy in the third scenario, the DQN-SNN controller is only implemented for the first two scenarios.

5.3. Braitenberg vehicle controller

To serve as a basis for further investigations, Kaiser proposed a simple Braitenberg vehicle controller for the lane following task (Kaiser et al., 2016). Depending on simple static connection schemes between sensors and motors, the vehicle exhibits simple animal-like behavior, such as turning towards or away from a sensory stimulus, e.g. in the form of light.

In a classic Braitenberg vehicle, the activity of sensory inputs steers the agent towards stimuli or away from stimuli depending on the connection scheme. In the first scenario, the robot is supposed to follow the lane without crossing the solid line on the right or the dashed line in the middle of the road. Therefore, if the robot deviates from the lane center, the motor neuron activities should increase or decrease so that the robot adjusts its direction accordingly. Fig. 14 shows the weights of the synaptic connections to the left and right motor neurons. If a line in the robot's vision gets closer to the bottom center of the image, the related motor neuron activity will be increased while the opposite motor neuron's activity will be decreased. If the robot gets close to the solid line on its right side, for example, left and right motor neurons will decrease and increase their firing rate, respectively, causing the robot to turn to the left. The same principle applies for the opposite side as well. The network weights are chosen manually by trial and error. This controller is only applied in the first scenario for further performance comparison.

5.4. R-STDP training results

Fig. 15 shows the training progress of the R-STDP controller in the first scenario. Specifically, the changes in the synaptic weights

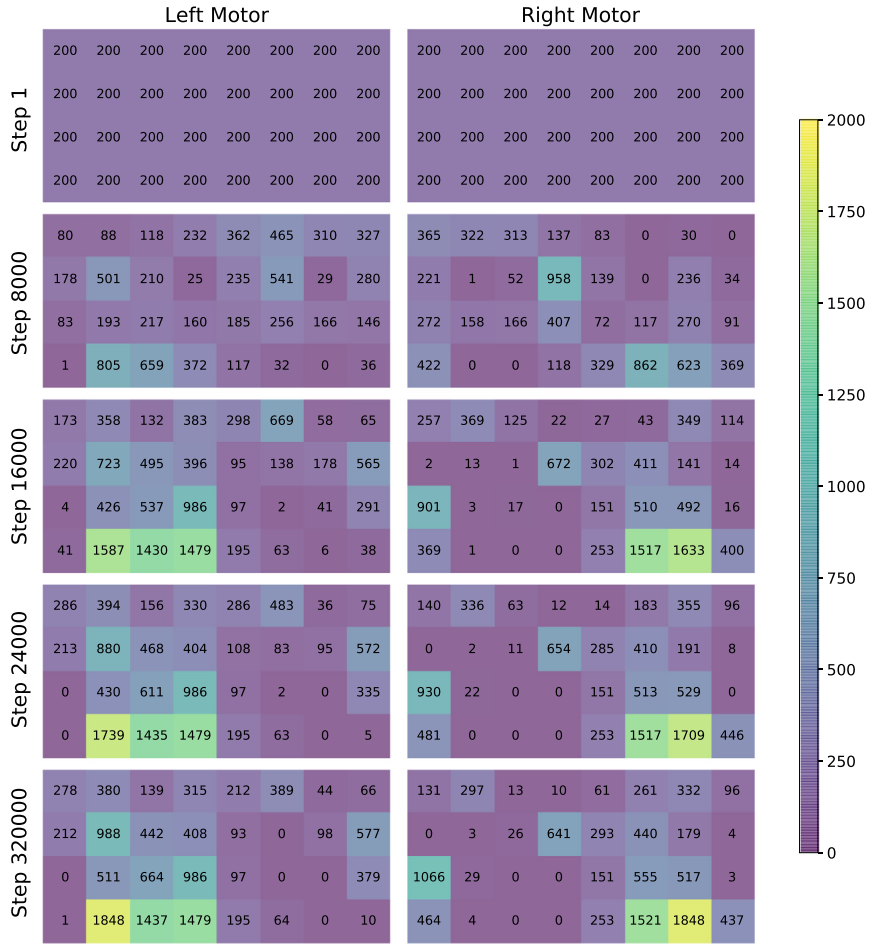


Fig. 15. Scenario 1. Learning progress of the R-STDP controller over every 8000 steps (1 *step* = 50 ms). Learned connection weights to the left and right motor neurons of the R-STDP controller are shown in last row after 30,000 simulation steps.

are shown every 8000 steps over the course of the simulation. Fig. 16 shows the termination position of the robot at each trail when it exceeds the lane center distance of 0.2 m, triggering a reset. A simulation step is equivalent to 50 ms both for the simulation of the SNN as well as the robot simulator itself. At the beginning of the training procedure, the robot will go straight forward, because all connection weights for both motor neurons have been set to the same value. Therefore, during the first 10,000 simulation steps, trials are mostly terminated at the first turn in both directions, when the robot misses the turn and the lane center distance exceeds 0.2 m. Each time the robot misses a turn, it will periodically induce high reward values at the beginning, changing the synaptic weights. Shortly before step 10,000, the robot has learned to take the turn, but it still deviates from the optimal lane center position. Consequently, the high reward over a longer period of time causes a significant change in the connection values. The learned weights after 30,000 simulation steps are shown in the last row of Fig. 15. Interestingly, the connection weights resemble the theoretically derived weights of the Braitenberg controller (see Fig. 14), with very low values in one half of the image and increasing values from the top corner to the bottom center in the other half of the image. Furthermore, it can be seen that left and right motor neurons mostly seem to be triggered through the middle and right road line enclosing the lane.

The training results of the controller in the second scenario are shown in Figs. 17 and 18. The results are similar to the first scenario, completing the first full lap in less than 5000 simulation steps. The weights of the controller network after 30,000 simulation steps are shown in the last row of Fig. 17. While the networks weights on the left side from both motor neurons resemble the connection weights learned in the first scenario, it can easily be seen that the weights on the right side have been left unchanged, due to the missing lines in this scenario and the consequential lack of activity during training.

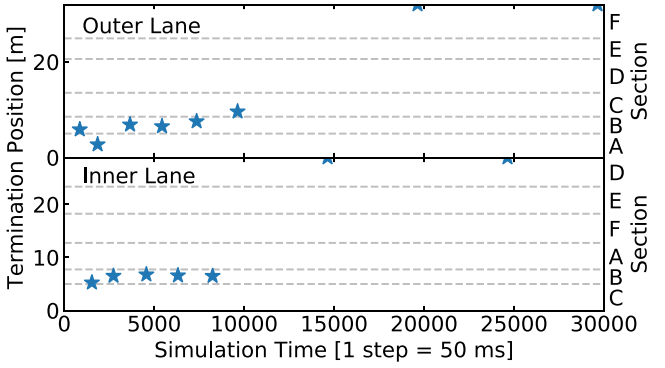


Fig. 16. Scenario 1. Termination position of the robot at each trail is marked by a star. During the first 10,000 simulation steps, the robot triggers resets at each trial in the first turn in both directions. Afterwards, it has successfully learned how to follow the lane, only triggering a reset when a complete lap is finished.

Fig. 19 shows the learning progress during the training in the third scenario. First, learning a successful control strategy

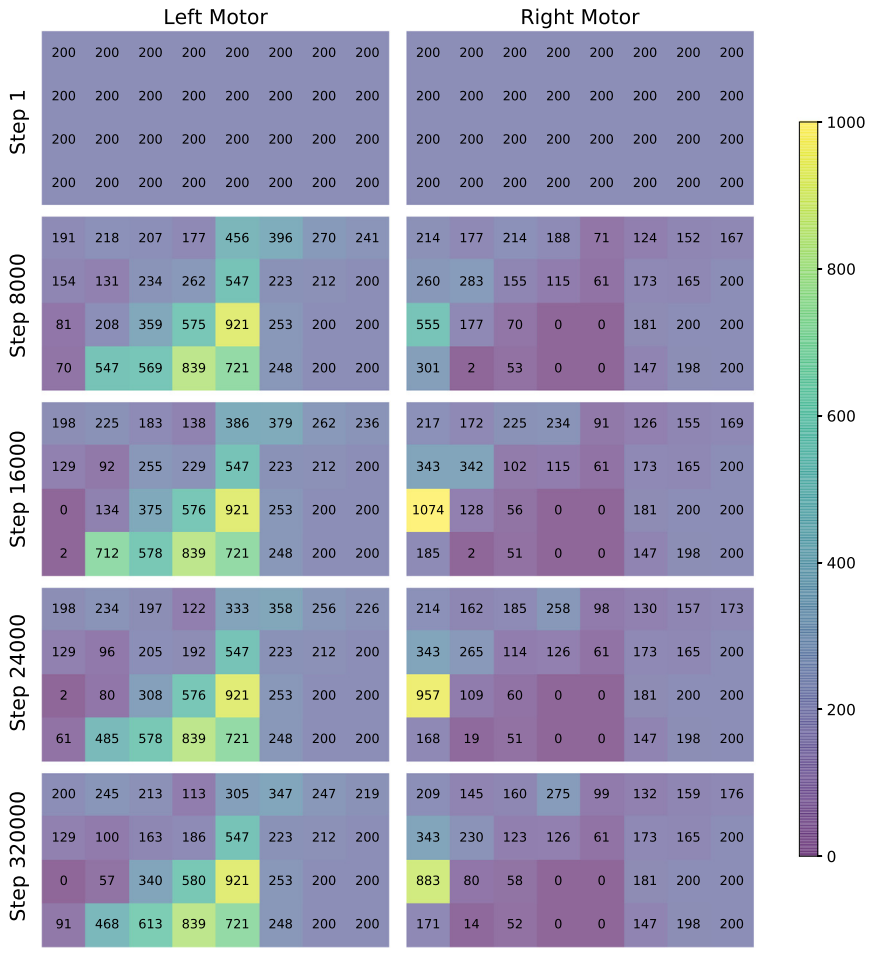


Fig. 17. Scenario 2. Learning progress of the R-STDP controller over every 8000 steps (1 *step* = 50 ms). Learned connection weights to the left and right motor neurons of the R-STDP controller are shown in last row after 30,000 simulation steps.

takes considerably more time than in the first two scenarios. The obvious explanation for this is that the third scenario incorporates two different road patterns, making the environment more complicated. Therefore, the controller has to distinguish between a higher number of different situations as well as slowing down the learning procedure. Moreover, due to the simple fact that the robot does not encounter certain situations until it has learned how to get there, it will only start learning a generalized control strategy that works for both lanes towards the end. The termination positions are shown in Fig. 20. As we can see, after an initial learning phase until approximately step 20,000, the controller is mostly reset in Section B (outer lane) and D (inner lane). When the weights have adapted sufficiently after approximately 75,000 steps, the robot finished the laps on both lanes. The last row of Fig. 19 shows the learned weights after 100,000 steps. In comparison to the first scenario, the weight patterns seem very similar, which makes sense considering the fact that the road pattern in the first scenario is the combination of both road patterns in the third scenario.

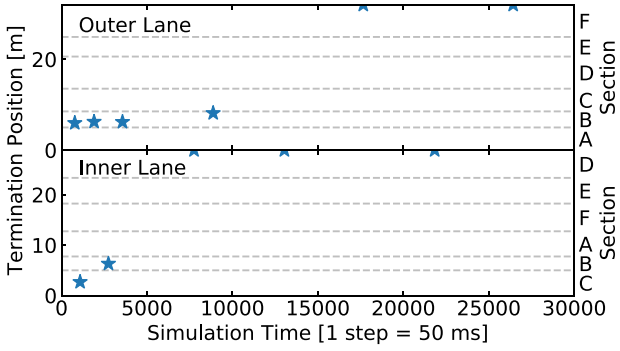


Fig. 18. Scenario 2. Termination position of the robot at each trail is marked by a star. The robot is mostly reset in sections B until laps are completed on both lanes after approximately 14,000 steps.

6. Performance & comparison

At the beginning, all the controllers are successfully trained and tested in the first scenario. While the Braitenberg controller is only implemented for the first scenario for comparison purposes, the remaining controllers learned a control strategy for the second scenario as well. Only the R-STDP controller, however, learned a stable control strategy for the third scenario. In order to obtain comparable performance metrics for each controller, they are evaluated after completing one lap on the outer lane in each scenario. Figs. 21–23 show the deviation of the robot from the lane center over the projected course position during one lap for each successful controller in all three scenarios, respectively. Moreover, the course is divided into the six sections as shown in Fig. 3. The robot path representation as a projection to the lane center line allows for a numerical analysis of the performance of

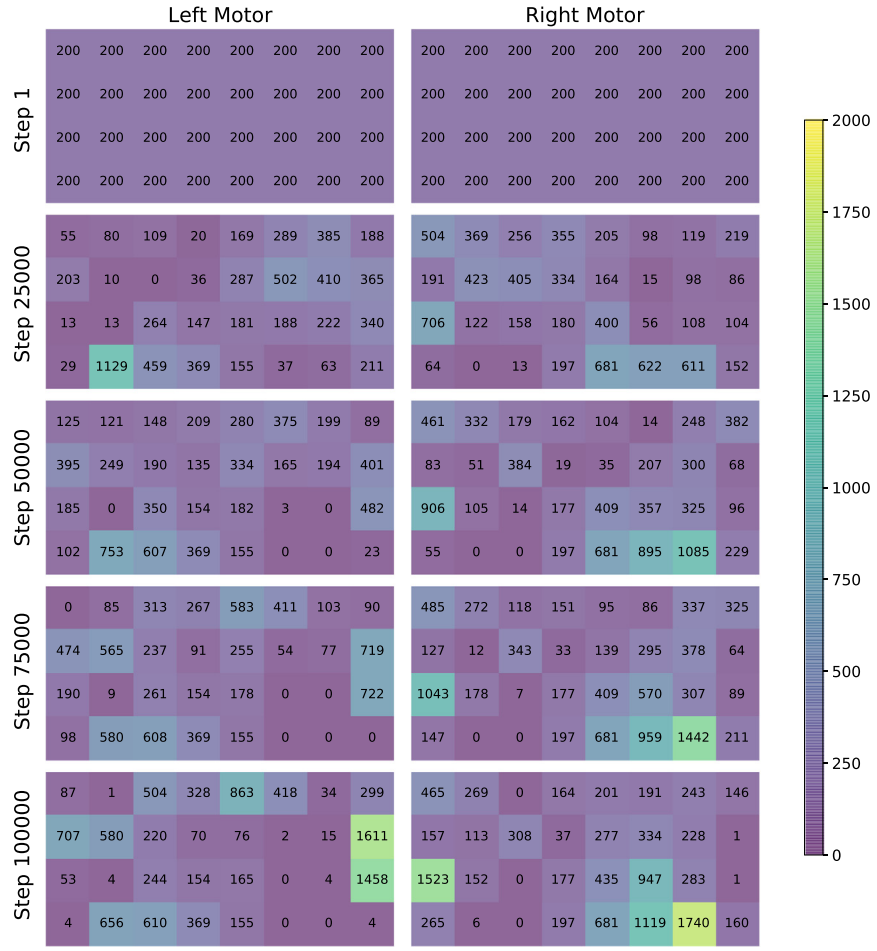


Fig. 19. Scenario 3. Learning progress of the R-STDP controller over every 25,000 steps (1 *step* = 50 ms). Learned connection weights to the left and right motor neurons of the R-STDP controller are shown in last row after 100,000 simulation steps.

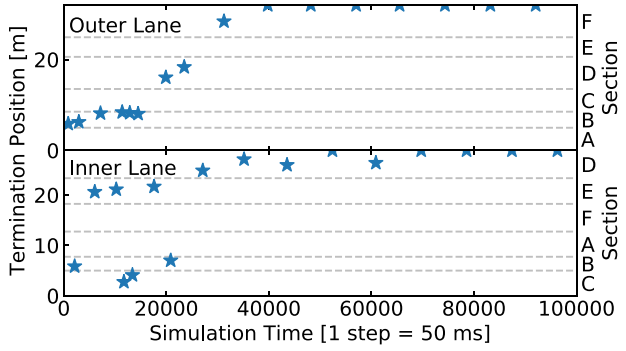


Fig. 20. Scenario 3. Termination position of the robot at each trail is marked by a star. After an initial learning phase, the robot is mostly reset in sections B and D until laps are completed on both lanes after approximately 75,000 steps.

all controllers . Specifically, the error distribution (distance to the lane center) can be shown in the form of a histogram as well as the mean error for each controller.

Initially, the DQN controller is trained and tested in the first scenario (see Fig. 21). The behavior of the robot very clearly depends on the section that it is in. In the straight sections A and C of the first scenario, the robot exhibits a tendency to the left (−0.1, defined in Fig. 7). In the left turn sections B, D, and F as well as the right turn section E, the robot tends to the right side of the lane (+0.1). While the controller does not optimally minimize the lane center distance over the whole course, it seems to be very stable with a constant deviation during each section. This behavior can also be seen in the error histogram with two peaks at both sides of the lane center. In contrast to the other controllers, the DQN algorithm leads to the highest numerical error with a mean deviation of $e = 0.041$ m. In the second scenario, the DQN algorithm shows a higher mean error (see Fig. 22), which can be explained by the reduced information in the state images. Especially in turns on the outer lane, when the robot sees only a very small part of the dashed line, there are only a few pixels containing any information. During the straight sections A and C, on the other hand, the robot follows the lane very close to its center, having enough information for a near-optimal control strategy. As discussed in the previous section, the DQN algorithm does not learn a stable policy in the third scenario that combines two different road patterns. During training, however, it manages to complete full laps and reaches the time step limit several times, proving that it can in fact learn a good policy with different road patterns. The problem here seems to be a general policy that works for both lanes, handling the full complexity of the task. Considering that other reinforcement learning tasks have successfully been solved using DQN (e.g. playing Atari games in Mnih et al. (2015)), it seems likely that this is mainly due to the simple network architecture that is implemented in this study which fails to evaluate states accurately enough in order to learn a stable policy.

Following the DQN controller, an SNN is trained in order to approximate the policy learned by the DQN algorithm. Therefore,

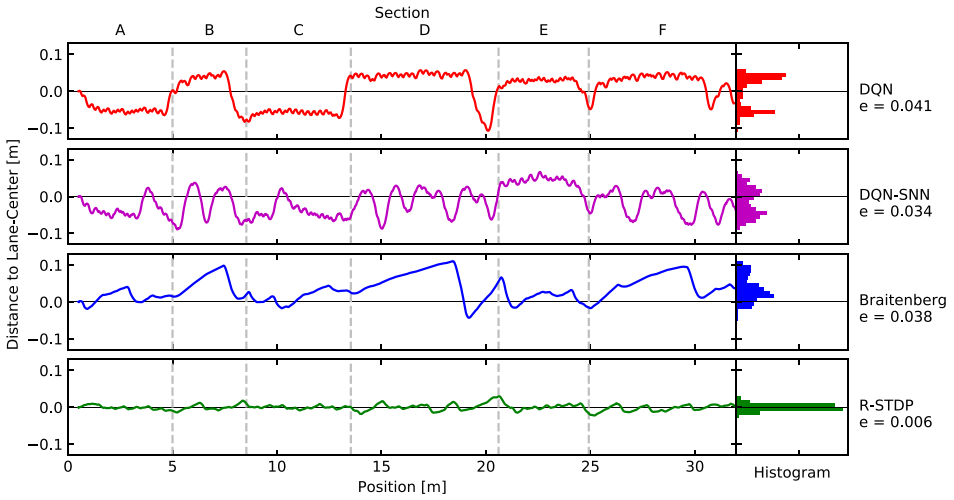


Fig. 21. Scenario 1: Comparison of different controllers on the outer lane. The deviation from the lane center is shown over the robot position projected to the lane center. Positive lane center distances correspond to deviations to the right side, negative distances to the left side. Course sections are marked by vertical dashed lines (A=straight, B=left, C=straight, D=left, E=right, F=left). On the right side, error distributions for all controllers as well as mean errors e (mean distance to the lane center) are shown.

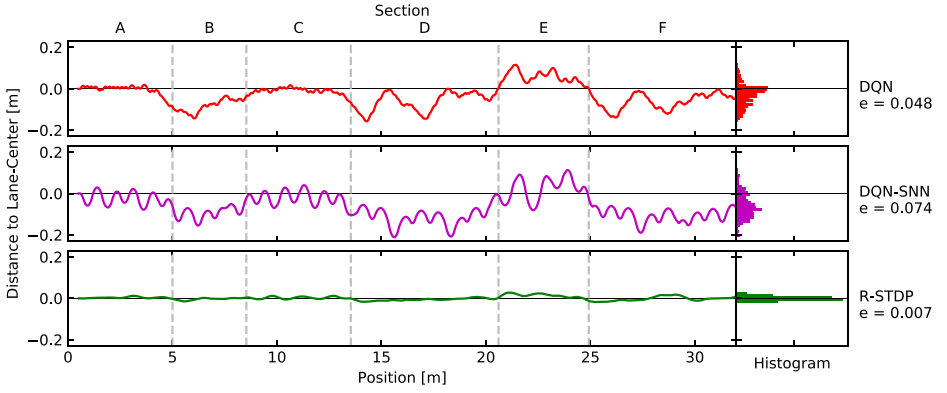


Fig. 22. Scenario 2: Comparison of different controllers on the outer lane. The deviation from the lane center is shown over the robot position projected to the lane center. Positive lane center distances correspond to deviations to the right side, negative distances to the left side. Course sections are marked by vertical dashed lines (A=straight, B=left, C=straight, D=left, E=right, F=left). On the right side, error distributions for all controllers as well as mean errors e (mean distance to the lane center) are shown.

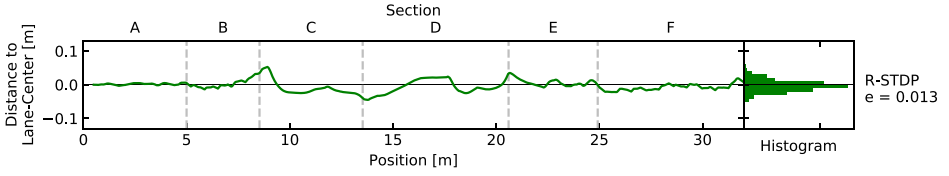


Fig. 23. Scenario 3: R-STDP controller on the outer lane. The deviation from the lane center is shown over the robot position projected to the lane-center. Positive lane center distances correspond to deviations to the right side, negative distances to the left side. Course sections are marked by vertical dashed lines (A=straight, B=left, C=straight, D=left, E=right, F=left). On the right side, the error distribution as well as the mean error e (mean distance to the lane center) are shown.

when looking at the performed lap, the DQN-SNN controller exhibits some similarities to the DQN controller, e.g. its left tendency in straight sections (A and C) or its right tendency in right turns (E). Overall, the controller seems more unstable, exhibiting a lot more oscillatory behavior, especially in left turns (sections B, D and F). When looking at the histogram, the error distribution of the DQN-SNN controller looks like a smoothened version of the DQN controller. Moreover, the mean error of the transferred SNN controller is surprisingly lower than the one of the original DQN controller. One explanation for this interesting behavior could be the decision frequency that is much higher than before. For every decision, the DQN controller collects consecutive frames in order to have enough data and combines them into one single state image. In this study, states images are composed of 10 DVS frames and decisions are made every 10×50 ms $= 500$ ms. The SNN, on the other hand, does not have to accumulate DVS frames beforehand. The network architecture will combine the data in the membrane potentials over time. Therefore, the network output can be read every 50 ms without having to wait for 10 simulation steps, although it takes some time until enough data has been

propagated through the network to produce meaningful output spikes. In fact, in many time steps during the simulation the SNN will not produce any output spikes at all, which is why action traces (defined in (3)) are used to ensure a control signal even if there are no output spikes. Considering the loss that was introduced when training the SNN on the state–action dataset, the performance of the controller still seems pretty good. In the second scenario (see Fig. 22), the SNN controller exhibits strong oscillatory behavior. Again, this can probably be explained by the reduced amount of information in the image data due to the missing lines. If fewer events are created and fed into the network, it takes longer until the information gets propagated through the network and generates an output spike. Therefore, the frequency in which the network can make decisions is much lower, resulting in this unstable behavior.

Next, the Braitenberg controller is evaluated while performing the same lap in the first scenario (see Fig. 21). While the controller successfully finishes the course, it can be seen quite clearly that it strongly tends to the right side of the lane, which can be explained by the robot's field of view. In the right half of the DVS images, the robot usually only sees the right solid line. In the left half, however, the robot sees the left solid line as well as the dashed middle line of the road, leading to a higher number of detected events and eventually greater activity of the left motor neuron. This will shift the robot to the right until it has reached a balance in the activity of the motor neurons. Even in the right turn (section E), the robot is mostly to the right of the lane center. In left turns, the distance to the lane center grows until a point is reached where previously unstimulated neurons with high weights are now excited. These will push the right motor neuron activity, leading to a movement correction back to the center. This can be seen in all three left turns (sections B, D, and F). The value of the controller's mean error during the performed lap is comparable to the first two controllers.

The purpose of the Braitenberg controller is to show the basic underlying control principle here. Instead of improving the controller performance by iteratively adjusting the network weights, the network is re-implemented using R-STDP synapses so that the weights could be automatically learned by the robot. In the previous section, we have already shown that those learned weights resemble the theoretically derived weights of the Braitenberg vehicle controller. Of all four controllers, the R-STDP controller shows the best performance in this task with comparatively very small deviations from the lane center. This gets even clear when looking at the performance histogram and the mean error that is almost an order of magnitude lower in comparison to the other controllers. First, one explanation for this behavior can be found in the very nature of SNNs that allow for high frequency decision-making without the need to split time into discrete steps. Second, the R-STDP training algorithm and the related reward are to a great extent tailored to this specific problem. The great success of deep reinforcement learning methods such as DQN lies in their capability to learn value functions in high-dimensional state spaces. This property allows for a general algorithm that is capable of solving sequential decision-making tasks formulated as MDP, even if rewards are sparse and delayed in time. The R-STDP controller, on the other hand, does not have this property. Basically, the R-STDP reward can be interpreted as a pre-defined value function with a global maximum that the algorithm will seek out. Furthermore, the reward signal incorporates prior knowledge, e.g. that increasing or decreasing motor neuron activities will lead the robot back to the center. Therefore, the R-STDP training algorithm solves a mathematically much less complex problem, leaving out the state evaluation step estimating future rewards that is crucial for every classic reinforcement learning algorithm solving MDPs with sparse, delayed rewards.

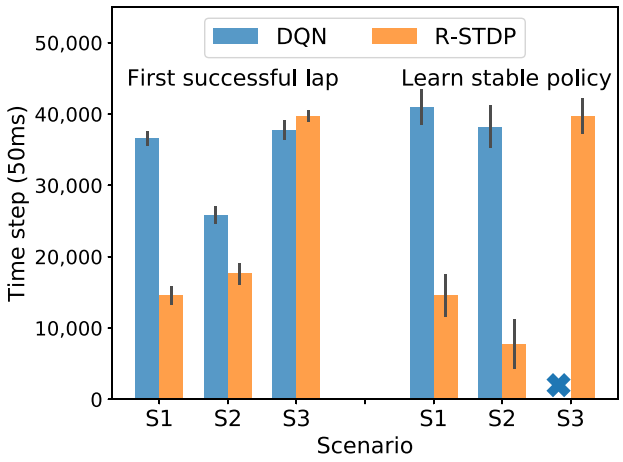


Fig. 24. Training time steps comparison for DQN and R-STDP controllers. The left group of bars shows the training time for achieving the first successful lap for both controllers. The right group shows the total training time steps for a stable policy. The standard deviation is marked with a black solid line for three training trails. For the DQN controller in the third scenario, it is marked with a *X*, since it fails to learn a stable policy.

Moreover, the training time steps of the DQN and R-STDP controllers for the three scenarios are shown in Fig. 24. For the first two scenarios, the R-STDP controller takes notably less time to learn a stable policy as compared to the DQN controller, even the DQN only takes random actions for the first 1000 time steps. For the third scenario, the R-STDP takes about 40,000 time steps to complete the successful learning process, while the DQN fails to learn a stable policy to accomplish consecutive full inner and outer laps. On the other hand, even for completing the first successful lap, the R-STDP still takes less time. For the DQN controller, there are still many episodes to be conducted to achieve a stable policy after the first successful trial. However, for the R-STDP controller, it will almost learn a stable policy once it completes the first successful lap. The possible explanation for the DQN controller is obvious, since the DQN achieves the first full lap by chance during the process of maximizing its rewards, rather than seeking its global maximum as R-STDP. In overall time steps, the R-STDP also beats the DQN, due to its inherent high frequency making for processing event-based data.

7. Conclusion

Spiking neural network, inspired by the mechanism of the brain, offers a promising solution to control robots with biological plausibility and exceptional performances. However, it lacks sophisticated training algorithms and practical robotic implementations, due to its complexities in constructing and optimizing an SNN. To bridge this gap, we trained an SNN controller with indirect and direct methods based on DQN policy transfer and R-STDP learning rule, respectively, and further implemented them in lane-keeping tasks for a Pioneer robot. For the indirect training, we first trained a DQN controller to accomplish lane-keeping tasks and then transferred its policy to an SNN controller by training it on an state–action dataset using supervised learning. Our indirect methods offer a quick and efficient way to build up an applicable spike-based controller that is able to be executed on neuromorphic hardwares. For the direct training, our method directly learns an SNN by utilizing the biological R-STDP learning rule and the event-based vision sensor, aiming to bring reinforcement learning capabilities to SNNs directly. Finally, we demonstrated the superiority of the controller trained by R-STDP by comparing the training results and their performance

of all controllers in terms of accuracy and speed, which were represented by the lateral localization accuracy and training time speed.

For future research, the R-STDP controller is intended to be a first step towards more sophisticated algorithms with real reinforcement learning capabilities. To date, research has not incorporated reward prediction errors yet, even though this phenomenon was observed in the brain. Therefore, such networks based on R-STDP should also be implemented using deep architectures in the future.

Acknowledgments

The research leading to these results has received funding from the Shenzhen Research JCYJ20180507182508857, the European Union Research and Innovation Programme Horizon 2020 (H2020/2014-2020) under the Specic Grant Agreement No. 720270 (Human Brain Project SGA1), and the Chinese Scholarship Council.

Appendix

Simulation parameters for each controller are listed in Tables 1–3.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2019.05.019.

References

Ambrosano, A., Vannucci, L., Albanese, U., Kirtay, M., Falotico, E., Martínez-Cañada, P., et al. (2016). Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform. In N. F. Lepora, A. Mura, M. Mangan, P. F. Verschure, M. Desmulliez, T. J. Prescott (Eds.), *Biomimetic and biohybrid systems* (pp. 16–27). Cham: Springer International Publishing.

Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., & Wu, Y. (2017). STDP-Compatible approximation of backpropagation in an energy-based model. *Neural Computation*.

Bi, G.-q., & Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, *18*(24), 10464–10472. http://dx.doi.org/10.1523/JNEUROSCI.18-24-10464.1998, http://www.jneurosci.org/content/18/24/10464.

Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M., et al. (2018). End to end learning of spiking neural network based on r-STDP for a lane keeping vehicle. In *2018 IEEE international conference on robotics and automation* (pp. 1–8). http://dx.doi.org/10.1109/ICRA.2018.8460482.

Bing, Z., Meschede, C., Röhrbein, F., Huang, K., & Knoll, A. C. (2018). A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, *12*, 35. http://dx.doi.org/10.3389/fnbot.2018.00035, https://www.frontiersin.org/article/10.3389/fnbot.2018.00035.

Bogacz, R. (2017). A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, *76*, 198–211. http://dx.doi.org/10.1016/j.jmp.2015.11.003, http://www.sciencedirect.com/science/article/pii/S0022249615000759, Model-based cognitive neuroscience.

Brette, R. (2015). Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, *9*, 151. http://dx.doi.org/10.3389/fnsys.2015.00151, http://journal.frontiersin.org/article/10.3389/fnsys.2015.00151.

Caporale, N., & Dan, Y. (2008). Spike timing–dependent plasticity: a Hebbian learning rule. *Annual Review of Neuroscience*, *31*(1), 25–46. http://dx.doi.org/10.1146/annurev.neuro.31.060407.125639, PMID: 18275283.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 international joint conference on neural networks* (pp. 1–8). http://dx.doi.org/10.1109/IJCNN.2015.7280696.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 international joint conference on neural networks* (pp. 1–8). http://dx.doi.org/10.1109/IJCNN.2015.7280696.

Drubach, D. (2000). *The brain explained*. Prentice Hall.

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468–1502.

Frémaux, N., & Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, *9*, 85. http://dx.doi.org/10.3389/fncir.2015.00085, http://journal.frontiersin.org/article/10.3389/fncir.2015.00085.

Indiveri, G. (1999). Neuromorphic analog VLSI sensor for visual tracking: circuits and application examples. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *46*(11), 1337–1347. http://dx.doi.org/10.1109/82.803473.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, *17*(10), 2443–2452. http://dx.doi.org/10.1093/cercor/bhl152.

Kaiser, J., Tieck, J. C. V., Hubschneider, C., Wolf, P., Weber, M., Hoff, M., et al. (2016). Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. In *2016 IEEE international conference on simulation, modeling, and programming for autonomous robots* (pp. 127–134). http://dx.doi.org/10.1109/SIMPAR.2016.7862386.

Kasabov, N. K. (2018). *Time-space, spiking neural networks and brain-inspired artificial intelligence: Vol. 7*. Springer.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016a). *Training Deep Spiking Neural Networks using Backpropagation*, *10*(November), 1–10. http://dx.doi.org/10.3389/fnins.2016.00508, arXiv:1608.08782.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016b). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508. http://dx.doi.org/10.3389/fnins.2016.00508, https://www.frontiersin.org/article/10.3389/fnins.2016.00508.

Legenstein, R., Pecevski, D., & Maass, W. (2008a). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), http://dx.doi.org/10.1371/journal.pcbi.1000180.

Legenstein, R., Pecevski, D., & Maass, W. (2008b). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), 1–27. http://dx.doi.org/10.1371/journal.pcbi.1000180.

Lewis, M. A., Etienne-Cummings, R., Cohen, A. H., & Hartmann, M. (2000). Toward biomorphic control using custom aVLSI CPG chips. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. symposia proceedings (Cat. No. 00CH37065): Vol. 1* (pp. 494–500). http://dx.doi.org/10.1109/ROBOT.2000.844103.

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128×128 120 dB $15\mu s$ latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, *43*(2), 566–576. http://dx.doi.org/10.1109/JSSC.2007.914337.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. ArXiv preprint. arXiv:1509.02971.

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*(5297), 213–215. http://dx.doi.org/10.1126/science.275.5297.213, http://science.sciencemag.org/content/275/5297/213.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Potjans, W., Morrison, A., & Diesmann, M. (2010). Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity. *Frontiers in Computational Neuroscience*, *4*, 141. http://dx.doi.org/10.3389/fncom.2010.00141, https://www.frontiersin.org/article/10.3389/fncom.2010.00141.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., et al. (2009). ROS: an open-source robot operating system. In *ICRA workshop on open source software: Vol. 3(3.2)* (p. 5). Kobe.

Rohmer, E., Singh, S. P. N., & Freese, M. (2013). V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ international conference on intelligent robots and systems* (pp. 1321–1326). http://dx.doi.org/10.1109/IROS.2013.6696520.

Song, S., Miller, K. D., & Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*(9), 919–926. http://dx.doi.org/10.1038/78829.

Stevens, C. F., & Zador, A. M. (1996). When is an integrate-and-fire neuron like a Poisson neuron? In *Advances in neural information processing systems* (pp. 103–109).

Thorpe, S., Delorme, A., & Rullen, R. V. (2001). Spike-based strategies for rapid processing. *Neural Networks*, *14*(6), 715–725. http://dx.doi.org/10.1016/S0893-6080(01)00083-1, http://www.sciencedirect.com/science/article/pii/S0893608001000831.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Wang, X., Hou, Z.-G., Lv, F., Tan, M., & Wang, Y. (2014). Mobile robot' modular navigation controller using spiking neural networks. *Neurocomputing*, *134*, 230–238. http://dx.doi.org/10.1016/j.neucom.2013.07.055, http://www.sciencedirect.com/science/article/pii/S0925231214000976, Special issue on the 2011 sino-foreign-interchange workshop on intelligence science and intelligent data engineering (IScIDE 2011) learning algorithms and applications.

Wang, X., Hou, Z., Tan, M., Wang, Y., & Hu, L. (2009). The wall-following controller for the mobile robot using spiking neurons. In *2009 international conference on artificial intelligence and computational intelligence: Vol. 1* (pp. 194–199). http://dx.doi.org/10.1109/AICI.2009.448.

Wang, X., Hou, Z.-G., Zou, A., Tan, M., & Cheng, L. (2008). A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing*, *71*(4), 655–666. http://dx.doi.org/10.1016/j.neucom.2007.08.025, http://www.sciencedirect.com/science/article/pii/S0925231207003025, Neural networks: algorithms and applications 50 years of artificial intelligence: a neuronal approach.

Whittington, J. C. R., & Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, *29*(5), 1229–1262. http://dx.doi.org/10.1162/NECO_a_00949, PMID: 28333583.