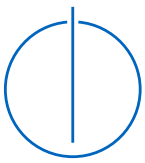
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Autonomous Locomotion Control for a Snake-like Robot with a Dynamic Vision Sensor and a Spiking Neural Network

Christoph Clement



DEPARTMENT OF INFORMATICS

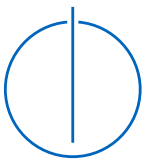
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Autonomous Locomotion Control for a Snake-like Robot with a Dynamic Vision Sensor and a Spiking Neural Network

Autonome Bewegungssteuerung für einen schlangenähnlichen Roboter mit einem Dynamic Vision Sensor und einem Spiking Neural Network

Author:	Christoph Clement
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Zhenshan Bing, M.Sc.
Submission Date:	September 26, 2018



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, September 26, 2018 Christoph Clement

Acknowledgments

To my parents who always support me during my studies.
To my girlfriend Eva who explained to me the inner workings of the brain on a beautiful beach in Thailand.
To my dear friend and colleague Anselm who is a great discussion partner and comes up with great ideas.
And to my supervisor Mr. Bing who stretched the limits of the work further than I ever expected.

Abstract

How does the brain process vast amounts of information with such great energy efficiency? Compared to modern Artificial Neural Networks (ANNs), the energy consumption of the brain is ten times lower - $20W$ in comparison to $200W$. On top of that, we are still far away from imitating the brain's capabilities with ANNs.
A solution to this question might come from the third generation of ANNs - Spiking Neural Networks (SNNs) that mimic the underlying mechanism of the brain better than current ANNs. They incorporate spatial and temporal information into their calculations leading them to be more computationally and energy efficient. Especially in combination with Dynamic Vision Sensors (DVSs), they are an excellent fit for autonomous robots where energy efficiency and fast real-time computations are critical success factors.
In this work, a SNN controller using the Reward-modulated Spike Timing Dependent Plasticity (R-STDP) learning rule for the autonomous locomotion control of a snake-like robot is implemented. The controller is trained in a maze environment and demonstrates the ability to cope with new situations in the form of different wall heights and maze angles during testing. In a real-world application, such a robot could be deployed in areas with uneven terrain like in a collapsed factory building.

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
CC	Classical Conditioning
CS	Conditioned Stimulus
DVS	Dynamic Vision Sensor
FPGA	Field-Programmable Gate Array
LIF	Leaky-Integrate-and-Fire
LTD	Long-Term Depression
LTP	Long-Term Potentiation
NEST	Neural Simulation Technology
NN	Neural Network
OC	Operant Conditioning
R-STDP	Reward-modulated Spike Timing Dependent Plasticity
ROS	Robot Operating System
SKAN	Synaptodendritic Kernel Adapting Neuron
SNN	Spiking Neural Network
SRM	Spike Response Model
STDP	Spike Timing Dependent Plasticity
UR	Unconditioned Response
US	Unconditioned Stimulus
V-REP	Virtual Robot Experimentation Platform

1. Introduction

How does the brain process vast amounts of information with such great energy efficiency? Although the brain makes up only two percent of the human body weight, it burns up 20% of the body's energy - around 20*W* [5]. But compared to modern Artificial Neural Networks (ANNs) this number is quite low. AlexNet v2 [18] or VGG [37], two of the most successful networks from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [19] - a well-known annual competition of ANNs in several visual recognition tasks, consume around 200*W* of CPU and GPU power combined [22]. It is also important to note that the brain is capable of doing many more tasks than just recognizing objects in images.
Especially when it comes to real-time applications like in autonomous cars or robots where energy efficiency and fast computations are critical factors for a successful deployment, current ANNs are not suitable exactly for this reason. At this point the third generation of ANNs comes into play - Spiking Neural Networks (SNNs) [32]. Besides the average firing rate of spikes, which is comparable to the activation value of a neuron in a conventional ANN, they also take into account the precise timing between spikes mimicking the functionality of real brains. This results in a more efficient network in terms of computation as the same task needs fewer neurons [26]. An ongoing research problem is that the backpropagation algorithm, which almost every ANN utilizes to calculate the weight updates, is not applicable to SNNs as the spikes are non-differentiable at spike times [20]. A promising solution in particular for reinforcement learning like task seems to be the Reward-modulated Spike Timing Dependent Plasticity (R-STDP) learning rule [8].
In this work, a snake-like robot equipped with a Dynamic Vision Sensor (DVS) learns to navigate through a maze using a SNN controller that uses the Reward-modulated Spike Timing Dependent Plasticity (R-STDP) learning rule. Snake-like robots with autonomous locomotion capabilities would be of great use for traversing unpassable terrain in dangerous areas like collapsed factory buildings. A DVS has several advantages compared to a regular camera as a pixel of a DVS only sends data if it perceives a change in light intensity. Thus, the amount of produced data is significantly reduced, and an asynchronous stream of events are created, which suits the character of SNNs perfectly.

The snake and its environment are simulated with the robot simulator Virtual Robot Experimentation Platform (V-REP) [11], and the SNN is implemented in Python using the simulation kernel of Neural Simulation Technology (NEST) [31]. The SNN follows a two-layered architecture: the input layer consists of Poisson neurons that are excited by the pixel values of the preprocessed DVS frame, and the output layer is made up of two Leaky-Integrate-and-Fire (LIF) neurons whose numbers of output spikes are used to calculate the snake's turning radius. The weight updates of the SNN are calculated with the R-STDP learning rule proposed by Legenstein et al. [21] that incorporates a reward modulated signal to control local synaptic plasticity. With this setup, the SNN controller's adaptability to new situations is shown: it is able to cope with different wall heights and angles between segments of the maze.
The thesis is structured as follows: Chapter 2 provides a brief background that is needed to understand Spiking Neural Networks. In Section 2.1 the biological background and the history of Artificial Neural Networks, as well as a mathematical description of SNNs and a brief description of Dynamic Vision Sensors are given. In Section 2.2 related work is reviewed.
In Chapter 3, the methodology used in this work is explained. Section 3.1 describes the V-REP simulation environment and its communication with the SNN. How the different ingredients of the SNN controller work together is provided in Section 3.2.
The SNN is evaluated in terms of its capabilities to cope with new scenarios with different wall heights and different angles in Chapter 4. The training performance is discussed in Section 4.1 before assessing the controller's ability to cope with new scenarios in Section 4.2.
Chapter 5 concludes the work and provides an outlook on open tasks and questions.

2. Background

This chapter provides a brief background that is needed to understand Spiking Neural Networks. In Section 2.1 the biological background and the history of Artificial Neural Networks, as well as a mathematical description of SNNs and a brief description of Dynamic Vision Sensors are given. In Section 2.2 related work is reviewed.

2.1. Theoretical Background

2.1.1. Artificial Neural Networks

Biological Background

Seemingly easy tasks for humans like speech recognition or movement control turn out to be incredibly complicated to imitate with computers. The brain can process vast amounts of information in a short time as it consists of about 86 billion neurons and each neuron is connected to up to thousands of other neurons via synaptic contacts [14]. According to Micheva et al., the number of synapses in the cerebral cortex is more than 125 trillion equaling to about 1,500 times the number of stars in the Milky Way galaxy [29].
Neurons communicate with each other using spikes - electrical signals that alter the potential of connected neurons. If the potential of such a neuron reaches a certain threshold, it sends out a spike itself. A spike or action potential is created in the cell body or soma and is depicted in Figure 2.1b. After the spike is created, it travels down the axon where Bodies of Ranvier amplify the signal to minimize losses. Axons divide themselves multiple times along the way, and after some time, the spike reaches a synapse. Synapses connect two neurons with each other and consist of the axon terminal, the synaptic cleft and the dendrite of the next neuron as depicted in Figure 2.2.
The arriving spike triggers the release of neurotransmitters into the synaptic cleft as synaptic vesicles fuse with the cell membrane. The neurotransmitters then traverse the synaptic cleft, which is filled with extracellular fluid and connect to matching receptors on the dendrite. There, ion gates are opened, triggering a cascade of events leading to the release of a postsynaptic potential which can be either excitatory (EPSP) or

inhibitory (IPSP) (see Figure 2.1c). This potential travels down the dendrite into the soma where it alters the overall membrane potential, and if it reaches a threshold value, a spike is sent out to connected neurons. After a refractory period of around 10*ms*, the next spike can be sent out.
Whereas the form of a spike is typically the same, the form of postsynaptic potentials changes depending on different amounts and types of neurotransmitters and activated ion channels. This dependency is called *synaptic efficacy*. A spike can have effects with various strength on a postsynaptic neuron depending on, for example, the density of neuro-hormones in the extra-cellular fluid, which is one example of *synaptic plasticity*. Just understanding how a single neuron works is extremely complicated. The interplay of several thousand neurons to recognize a face, for example, exceeds current human comprehension by far. Nevertheless, experiments show that neurons use spatial and temporal information to encode information [43].

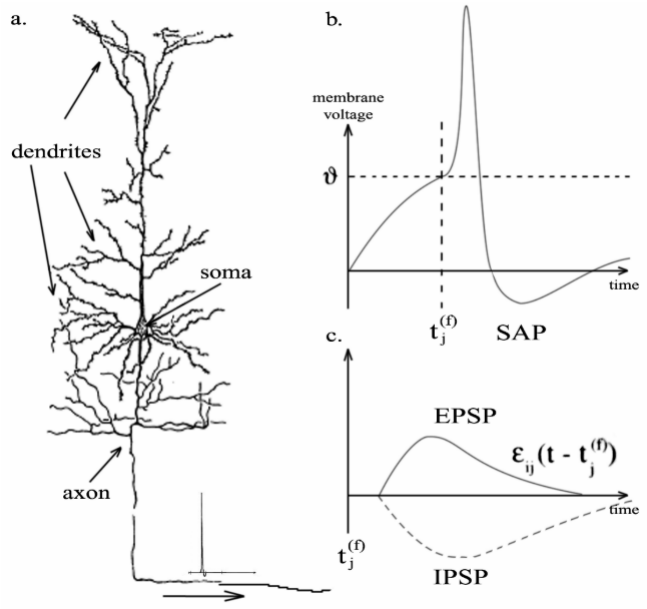


Figure 2.1.: (a) is a drawing of a neuron showing the axon, soma, and dendrites. (b) shows a spike or action potential over time triggered by the reached threshold of the membrane potential in the soma. (c) shows a positive (excitatory) and negative (inhibitory) postsynaptic potential over time. [43].

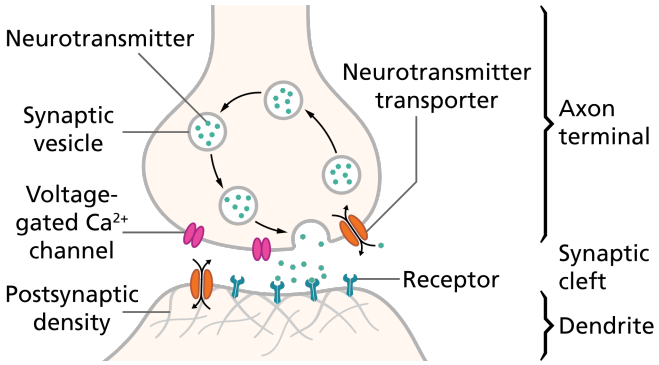


Figure 2.2.: A schematic of a synapse with the axon terminal of a neuron at the top, the dendrite of connected neuron at the bottom, and the synaptic cleft between them. A spike triggers the release of neurotransmitters that traverse the synaptic cleft and dock to receptors of the dendrite. There, a postsynaptic potential is triggered that travels down the dendrite into the soma [38].

A Short History of Artificial Neural Networks

Neurophysiologist Warren McCulloch and mathematician Walter Pitts modeled a simple Neural Network in 1943 [27]. Their model worked based on an *"all-or-none"* process. If a certain number of synapses of a neuron are excited during a fixed period, the neuron sends out a high signal. This mechanism can compute any Boolean output function.
Donald Hebb published his famous book *The Organization of Behavior* in 1949 [13], in which he proposed one of the earliest learning theories in neuroscience: *"neurons wire together if they fire together"* [24]. Hebbian processes can be used to change the weights between neurons in ANNs and lay the foundation for the understanding of how learning works in the brain. Based on these ideas, researchers at Stanford University developed a system called "MADALINE", short for Multiple ADAptive LINear Elements in 1959, that is still in use today to filter echoes on phone lines. [34]
The examples described above belong to the first-generation of NNs that were limited to digital output. The second generation uses continuous activation functions like the sigmoid function, which makes them more computationally efficient than first-generation NN. Additionally, they can perform computations with analog input and output, which enables the use of learning algorithms based on gradient descent, for example, backpropagation [26].
The output of a neuron in an ANN can be interpreted as the current firing rate of a biological neuron, the so-called rate coding. However, if we take into account research by Thorpe and Imbert [41], we need to rethink this interpretation. They demonstrated

that humans analyze and classify visual patterns in just $100ms$, which is too fast to be explained by rate coding. There are ten synaptic steps between the retina and the frontal lobe, which means that a neuron processes the signal in about $10ms$. However, the measured firing rate of these neurons is $20-30ms$ to process a signal. Thus, a new explanation for these fast cortical computations is needed. The functionality of Spiking Neural Networks can give a possible explanation for these observations.

2.1.2. Spiking Neural Networks

The third generation of NNs applies results of neuroscientific experiments to its mechanisms. Evidence showed that biological neural systems take into account the precise timing of spikes rather than just the firing rate to encode information. Spiking Neural Networks (SNNs) incorporate spatial-temporal information in their computations and can be applied to all problems that are solvable by normal ANNs [32]. Simultaneously, they are in theory even more computationally efficient meaning that they need fewer neurons than ANNs to solve the same task [26].

Mathematical description

The most commonly used model to mathematically represent spiking neurons are Leaky-Integrate-and-Fire (LIF) neurons. This implementation underlies the assumption that not the specific shape of spikes, but their precise timing encodes information [32]. A sequence of firing spikes - a spike train, can be described as

$$S(t) = \sum_t \delta\left(t - t^f\right), \tag{2.1}$$

where $f = 1, 2, ...$ is the label of a spike, and $\delta(.)$ is the Dirac delta function with $\delta(t) \neq 0$ for $t = 0$ and $\int \delta(t)\, dt = 1$.
The input signal $i(t)$ of the postsynaptic neuron that is induced by an incoming spike train can be described by

$$i(t) = \int_0^\infty S_j(s-t) exp(-s/\tau_s) ds, \tag{2.2}$$

where τ_s is the synaptic time constant and $S_j(t)$ is a presynaptic spike train [32].
A LIF neuron is modeled as

$$C\frac{du}{dt}(t) = -\frac{1}{R}u(t) + \left(i_0(t) + \sum w_j i_j(t)\right), \tag{2.3}$$

where $u(t)$ corresponds to the neural membrane potential, C is the membrane capacitance, R is the input resistance, $i_0(t)$ is the external current driving the neural

state, $i_j\,(t)$ is the input current from the j-th synaptic input, and w_j corresponds to the strength of the j^{th} synapse [32]. In this model, a neuron fires when the membrane potential u reaches a firing threshold value v. After a spike fires, its potential u is reset to a value $u_{reset} < v$ for a certain time representing the neural absolute refractory period (see Figure 2.3) [10].

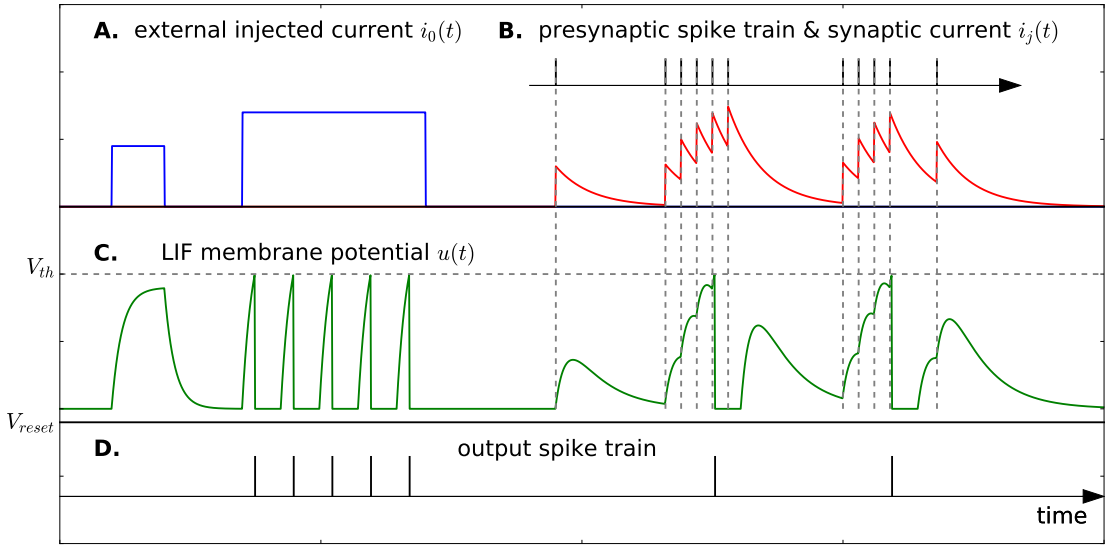


Figure 2.3.: (A) shows an external input current $i_0\,(t)$. (B) depicts a sample presynaptic spike train and the associated synaptic current $i_j(t)t$. (C) shows the corresponding membrane potential $u(t)$ of a LIF neuron over time. (D) depicts its output spikes, firing when the threshold V_{th} is reached [32].

Learning Rule

Simply said, learning means that multiple synapses in the brain change their strength in a coordinated way. The first proposal on how synapses change their weights came from Donald Hebb in 1949 and included only synaptic potentiation - two neurons being active at the same time strengthens their connection.
Other researchers expanded this theory over time to Long-Term Potentiation (LTP) and Long-Term Depression (LTD) of synapses dependent on the firing activity of pre- and postsynaptic neurons. This process is called *Hebbian plasticity* and constitutes the basis for most learning and memory models. As Hebbian plasticity only acts locally, it can lead to destabilization through extraordinarily high or low firing rates. Thus, a biologically plausible mechanism is needed that regulates the total amount of excitation. One such rule derived from experimental data is called Spike Timing Dependent

Plasticity (STDP). Synaptic changes across many neurons occur due to differences in pre- and postsynaptic firing rate. If a presynaptic spike fires within a window of tens of milliseconds before the postsynaptic spike, LTP is induced in the synapse connecting the two neurons. If it is the other way round, LTD is induced. The smaller the difference between pre- and postsynaptic spikes, the higher the effect of the potentiation or depression [1].
An ongoing research problem is to bridge the gap between the microscopic level where experiments proved the STDP rule and the macroscopic level where behavioral changes of organisms take place. Researchers think that rewards through neuromodulatory systems play a crucial role in bridging that gap. The presence of dopamine, for example, influences the synaptic weight changes.
Legenstein et al. propose the following rule for Reward-modulated Spike Timing Dependent Plasticity (R-STDP):

$$\frac{d}{dt}w_{ji}(t) = c_{ji}(t)d(t), \tag{2.4}$$

where w_{ji} is the weight of a synapse from neuron i to neuron j, $c_{ji}(t)$ is an eligibility trace of this synapse that collects weight changes proposed by STDP, and $d(t) = h(t) - \overline{h}$ results from a neuromodulatory signal $h(t)$ with mean value $\overline{h}$, which can be interpreted as a reward signal [21].
The weight update function used in this work is defined as

$$\Delta t = t_{post} - t_{pre}, \tag{2.5}$$

$$W(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+}, \text{ if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, \text{ if } \Delta t < 0, \end{cases} \tag{2.6}$$

where the constants A_+ and A_- scale the strength of potentiation and depression, and τ_+ and τ_- are time constants defining the width of the positive and negative learning window. Figure 2.4 depicts the weight update function.
Izhikevich [16] defined the eligibility trace of a synapse as

$$\frac{d}{dt}c(t) = \frac{-c}{\tau_c} + W(\Delta t)\delta\left(t - t_{pre/post}\right), \tag{2.7}$$

where τ_c is the time constant of the eligibility trace, δ is the Dirac delta function and $t_{pre/post}$ are the spike times of the associated neurons. The eligibility trace is plotted in Figure 2.5A and its influence on the synaptic weight are shown in Figure 2.5B.

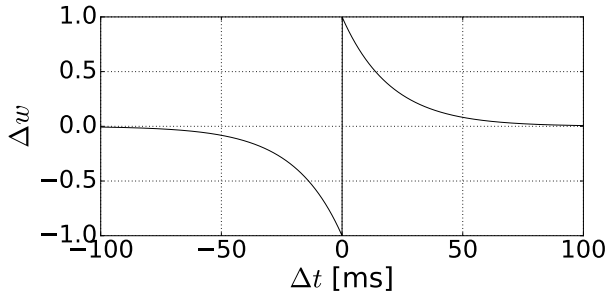


Figure 2.4.: Weight update function with $A_+ = A_- = 1.0$ and $\tau_+ = \tau_- = 20ms$, which are the same parameters used in the simulation.

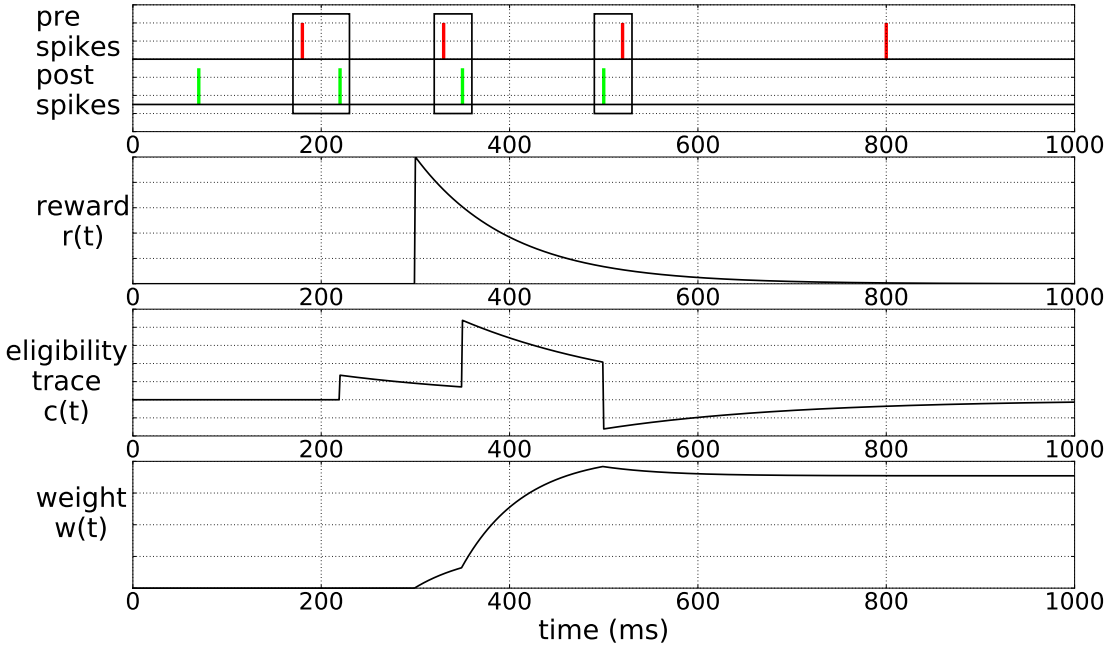


Figure 2.5.: The contributions of pre-before-post spike pairs and a post-before-pre spike pair to the weight function $w(t)$ are shown in this plot. Although the first pre-before-post spike pair changes the eligibility trace $c(t)$, it does not influence w(t) because the reward r(t) is zero during that time. The second pre-before-post spike pair increases the weight function as the time difference between post and pre spike is positive and thus excitation is induced. Vice versa for the post-before-pre spike pair that induces inhibition. [21].

2.1.3. Dynamic Vision Sensor

A Dynamic Vision Sensor (DVS) solves some of the problems a regular video sensor encounters like redundant information, redundant computation, and high latency with inspiration from the workings of the human retina. The basic idea is that the sensor transmits only local pixel-level changes events resulting in a lower data volume, energy consumption and latency, and in a higher speed and dynamic range.
Lichtsteiner et al. developed a $128px \times 128px$ DVS that has a dynamic range of $120dB$ and a latency of $15\mu s$ [23]. They designed a pixel that detects local relative intensity changes temporally independent from other pixels, which creates an asynchronous stream of spike events with sub-millisecond timing precision. Figure 2.6 depicts images taken with a DVS.

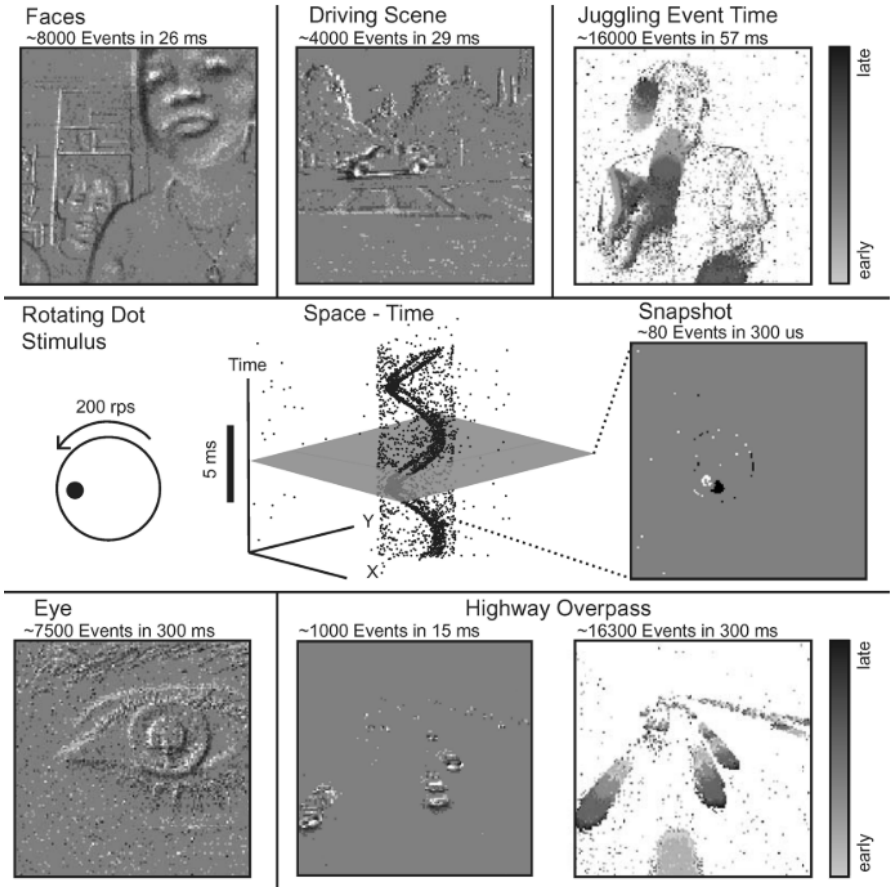


Figure 2.6.: Images taken with a DVS with object or camera motions rendered as contrast, gray scale-time or 3D space time [23].

2.2. Related Work

In this section, various approaches to using SNNs for robot control are reviewed and sorted by the used weight update mechanism.

2.2.1. Manual Set Weights

If the task environment in that the robot operates or the number of sensors used as input for the SNN is relatively small, the weights of the SNN controller can be manually set instead of being learned with a weight update rule, which is complicated to implement.
Wang et al. [44] build an autonomous mobile robot that can follow a wall using sonar sensors and a SNN controller. They utilize the mobile robot CASIA-I with a ring of 16 ultrasonic sensors and implement a wall following controller with manually set connections and weights based on a SNN with an input, a hidden, and an output layer. They simulate the robot in a square area with walls where it successfully follows the wall even when there is an obstacle in the way.
A robot that can avoid obstacles and reach targets is built by Blum et al. [3] using a spike-based neuromorphic controller for which a DVS provides the sensory input. They use a setup consisting of the neuromorphic chip ROLLS on the miniature computing board Paralella that wirelessly interfaces with the PushBot, a mobile platform with a differential drive and an embedded DVS. Their robot is able to navigate cluttered environments, avoid moving obstacles, and follow a target at the same time using only 256 spiking neurons.
One of the downsides of manually setting the network weights is that one has to renew them for every task by hand, and if the task gets too complicated, the manual setup is not possible anymore.

2.2.2. Genetic Algorithms

Genetic Algorithms (GAs) are an exciting approach to optimize a set of parameters, for example, the weights of a SNN. Their functionality is based on mutation, crossover, and selection inspired by the process of natural selection. Initially, there is a large population of candidate solutions, each with a set of fixed properties. For each candidate, the fitness, a metric that needs to be optimized, is determined and the properties of the best candidates are recombined or mutated for the next generation until the candidate achieves the desired fitness level [15].
Hagras et al. [12] implement a unique form of GAs: an adaptive GA using adaptive crossover and mutation probabilities to evolve a SNN controller for a real robot that

exhibits edge-following behavior. The robot used in this work consists of two wheels, nine ultrasound sensors, and four bump detection sensors. The SNN is implemented with the Spike Response Model (SRM) of which LIF neurons are a special case. The network is made of an input layer to which the analog ultrasound sensors of the robot are connected and an output layer that links to the robot's actuators. The GA based system used to evolve the SNN controller converges in a relatively short time interval and produces a controller with good edge following behavior.

2.2.3. Conditioning and Spike Timing Dependent Plasticity

Ivan Pavlov's famous dog experiments lay the foundation for the study of animal learning and behavior analysis through Classical Conditioning (CC). In his experiments, he presented the Unconditioned Stimulus (US) - food - to a hungry dog in combination with the Conditioned Stimulus (CS) - a ringing bell sound. First, the dog showed the Unconditioned Response (UR) - salivation - only if the US was presented. However, after Pavlov repeated the combination of US and CS for several times, he only needed to ring the bell to trigger the dog's response - salivation, which in this case is called the Conditioned Stimulus (CS). The dog learned to associate the ringing bell sound with food [35].
Operant Conditioning (OC) differentiates itself from CC through the behavior of the animal. Whereas in CC, an event like the presentation of food takes place independently of the animal's behavior, in OC the animal's action influence the events in its environment. The animal can influence its surroundings to its benefits either by maximizing a reward or minimizing punishment. In OC there are two types of consequences: behavior-enhancing that reinforce actions and behavior-suppressing that punish actions. Examples are a dog getting a treat for a trick he does or a rat that learns not to touch a button which leads to an electric shock [39].
Dumesnil et al. [6] implement these two types of learning from the animal world with a STDP learning rule for a SNN that controls a robot in a maze environment. Their robot consists of an Arduino, a Field-Programmable Gate Array (FPGA), RGB and proximity sensors, and motors. The STDP learning rule is implemented in the FPGA with the Synaptodendritic Kernel Adapting Neuron (SKAN) model. The environment is a maze that can change its features. They demonstrate that their relatively simple system with few neurons can navigate through the maze both with CC and OC.
Evans [7] uses a dopamine modulated STDP learning rule controlling a SNN for a robot that performs food foraging tasks. The two-wheeled robot used in this work has two range and two touch sensors attached. The environment consists of either food items, poison items, or food containers holding a food item that the robot cannot sense from outside the container. The SNN has a two-layer architecture with sensory

neurons in the first layer and motor neurons in the second. Their weights are updated according to the dopamine modulated STDP learning rule, which is similar to the one used in this work. With this setup, the robot manages to learn food attraction in changing environments and demonstrates that a SNN with a dopamine modulated STDP learning rule can solve reinforcement learning like tasks in continuous action space.
In Meschede's work [28], two different approaches to training SNNs for robot control are implemented. First, he transfers a policy learned with Deep Q-Learning to a SNN, and second, he trains the SNN directly with a R-STDP learning rule. He uses a simulated version of the Pioneer P3-DX robot equipped with a DVS, and its environment consists of three different lane following scenarios as proposed by Kaiser et al. [17]. The SNN is implemented with a two-layer architecture with the DVS pixels as inputs for the first layer and two motor neurons as the output layer, which is similar to the architecture used in this work. Both controllers successfully learn the lane following scenarios, but the R-STDP controller works better than the Deep Q-Learning one, which is why in this work the capabilities of the R-STDP learning rule are further evaluated.
Shim et al. [36] implement a collision avoidance task using a SNN with a multiplicative R-STDP learning rule. The simulation environment consists of a robot car model with five ultrasonic sensors, obstacles, and a target. The network was made of three layers with the input and hidden layer consisting of excitatory and inhibitory neurons, and the output layer of a left and a right neuron controlling the turning angle. The authors compare their proposed network architecture to a reference SNN and Q-learning in scenarios with ascending complexity and find that their proposed network and learning rule worked best.
Tang et al. [40] employ a robot controlled by a SNN that is able to spatially map and navigate a real-world environment through STDP. Their SNN structure mimics the brain's navigational system with different kinds of neurons connected either by hardwired or STDP synapses. The network outputs the linear and angular speed of the robot. The environment is simulated with Gazebo consisting of a double-T-shaped maze, and in it, the robot is able to map its environment with minimum learning in an end-to-end fashion.
Rast et al. [33] built a learning system able to attend to specific objects in real-time. The system is made of the iCub humanoid robot and a SpiNNaker neuromorphic chip. Their SNN model includes a bottom-up visual pathway with a DVS as input and a top-down pathway from goals to action biasing. With this complex network architecture, their system learns object-specific attention using a STDP learning rule.

3. Methodology

In this chapter, the methodology of this work is outlined. In Section 3.1 the V-REP environment in that the snake is simulated and its communication with the SNN are described. How the different subsystems of the SNN controller work together is explained in Section 3.2, before the training workflow is illustrated in Section 3.3.

3.1. Simulation Environment

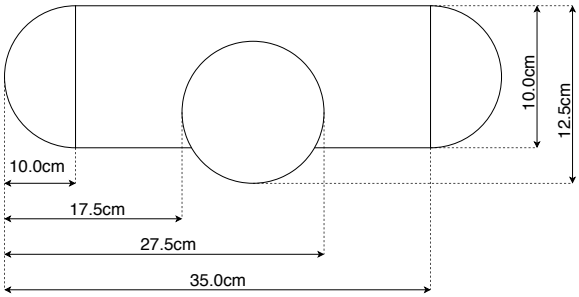
The virtual model of the snake and its surroundings are built with the robot simulation software Virtual Robot Experimentation Platform (V-REP) [11]. The program has several features that make it a great tool to simulate robots. First, it is based on a distributed control architecture that enables each object in the simulation to be controlled by its own script. Second, these scripts can be programmed with mutually compatible approaches like plugins, ROS nodes and six different programming languages, which enable the integration of external software into the simulation. And third, V-REP can simulate vision and proximity sensors with several physics engines built in to simulate object interaction. In this case, the Bullet physics engine (V2.78) [4] is used, and the simulation time step is set to $50ms$.
When researchers build a robot application, they need to deal with lots of different software tools and hardware. Thus, a flexible framework that defines standardized interfaces and functionalities like message passing between programs is of great use. The Robot Operating System (ROS) [9] plays this role in the simulation environment, which can easily be implemented in V-REP with the ROS interface plugin. It is utilized to create ROS nodes to enable the communication between the simulation environment and the SNN.

3.1.1. Snake Robot

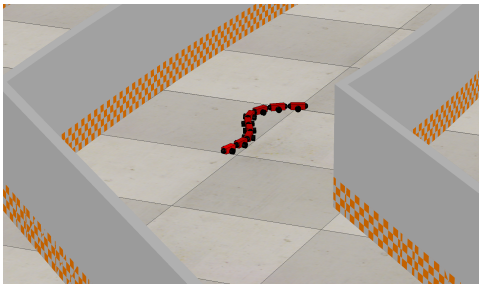
Model

The model of the snake is built of nine connected segments that turn with respect to each other. Each segment is made of a cuboid that has a size of $25cm \times 10cm \times 10cm$ and two hemispheres attached to each of the two $10cm \times 10cm$ sides of the cuboid.

Additionally, there are two cylindrical wheels attached to two opposite $25cm \times 10cm$ sides as can be seen in Figure 3.1a. A revolute joint that turns around the z-axis connects two hemispheres of different segments. The angle between the joints are controlled by an embedded Lua script, which is a lightweight embedded scripting language supported by V-REP. Figure 3.1b depicts the snake in the maze environment, which is explained in more detail in Section 3.1.3.



(a) Dimensions of one segment of the snake.



(b) The simulated snake in V-REP.

Figure 3.1.: Two Figures of the Snake Model.

Slithering Gait Equations

Bing et al. [2] developed the slithering gait equations that are implemented in the Lua script of the snake. The extended gait equations in 2D are described as

$$\phi(n,t) = C + P \cdot A \cdot sin(\Omega \cdot n + \omega \cdot t), \tag{3.1}$$

$$P = \left(\frac{n}{N} \cdot z + y\right) \in [0, 1], \forall n \in [0, N], \tag{3.2}$$

where $\phi(n,t)$ is the joint angle of the n^{th} joint at time t, C is a body shape offset that will be explained shortly, P is the linear reduction parameter described in equation 3.2, A is the amplitude of the wave in the plane, Ω is the spatial frequency that defines the cycle number of the wave, ω is the temporal frequency that defines the traveling speed of the wave, N is the module number of the snake, and y and z are linear coefficients for the linear reduction equation 3.2.
Through the slithering, sine-like motion of the snake, the head module strongly moves from side to side hindering a stable DVS image, for which reason the linear reduction equation is introduced. The linear reduction parameter P in equation 3.2 modifies the value of the amplitude such that it has its smallest value at the head segment and

linearly increases to the tail. With the linear reduction, the body of the snake moves in a frustum shape, which reduces the sideways motion of the head by 70% [2].
The body shape offset C is added as a bias value to equation 3.1 and is used to control the turning radius of the snake. It is calculated as

$$C = \frac{l_0 \cdot \sum_{k=1}^{N} cos\left(A \cdot \sum_{p=1}^{k} sin\left(\Omega \cdot p\right)\right)}{r}, \tag{3.3}$$

where l_0 is the length of each module and r is the turning radius of the snake [2].

Implementation

These equations are implemented in a non-threaded embedded Lua child script attached to the snake, which the main script in V-REP calls. A child script in V-REP consists of four main parts: *initialization*, *actuation*, *sensing* and *cleanup*.
The *initialization* part is executed once at the beginning of the simulation. In the case of this simulation it is used to save the initial position and orientation of the snake, to check whether the ROS plugin correctly loaded, to set up the ROS publishers and subscribers, to set the initial parameter values, and to calculate the initial values for the gait equations.
During the simulation, the *actuation* part of the script is executed every step, and in it, the values of the gait equations are calculated to update the angle between the joints of the segments of the snake. To be able to do that, the snake receives the turning radius via the *turningRadius* subscriber from the SNN. In case the simulation needs to be reset it receives a Boolean value indicating whether it should start in the positive or negative travel direction via the *resetRobot* subscriber. Every time the subscribers receive a new value, a callback function is executed, which updates the body shape offset C in case of the turning radius subscriber or resets the snake in case of the reset subscriber. Besides that, the current position of the snake in the maze is published via the *transformData* ROS node.
The *sensing* part is also executed every step and it handles the data from the DVS by formatting and publishing it via the *dvsData* node.
Before the simulation stops, the *cleanup* part is executed once to shut down the ROS publishers and subscribers.

3.1.2. Dynamic Vision Sensor

Due to their event-based nature, DVSs are an excellent fit for working in combination with SNNs (see section 2.1.3 for further details on how they work).

However, these properties impede realistic simulations of DVSs at the same time, which is why in other research a real DVS was used to observe the screen instead of simulating it. For example, Orchard et al. [30] created a neuromorphic version of the well known MNIST dataset by moving the DVS in saccade movements in front of a screen showing the MNIST digits.
Even so, in this work, the approach proposed by Kaiser et al. to simulate a DVS is used [17]. Hereby, two consecutive frames of a conventional vision sensor are subtracted, and a threshold is applied to filter pixels that show a substantial deviation in light intensity. Thus, ON-events are created for pixels that have a higher value than the threshold θ and OFF-events for these that have a lower value. The threshold value θ can be compared to the slow light adaption parameter in biology [42] and the DVS bias.
The comparison between a real DVS and a simulated one in Figure 3.3 clearly shows the drawback of a simulated DVS with a much lower temporal resolution and multiple events having the same time-stamp. However, Kaiser et al. argue that a robust SNN can handle such discrete inputs as humans are also able to see fluid motion when looking at discrete frames with a high enough frame rate.
IniLabs, a spin-off of the Institute of Neuroinformatics at the University of Zurich and the ETH Zurich and a manufacturer of DVSs, offers a V-REP model of their DVS128 camera, which has a resolution of $128px \times 128px$ and is used in this simulation [25]. Figure 3.2 shows a DVS frame from the simulation with the walls of the maze on the left and right side.

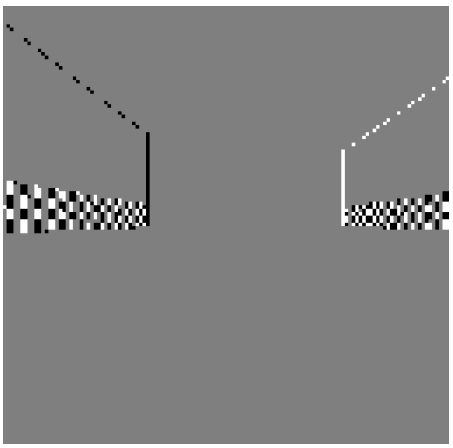


Figure 3.2.: Output of the simulated DVS in V-REP. Black and white pixels show ON- and OFF-events, respectively.

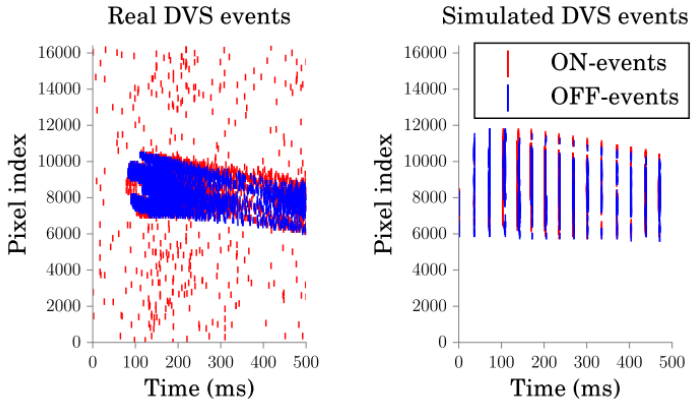


Figure 3.3.: Kaiser et al. compare events generated from a real DVS and a simulated one while observing a flying ball. ON-events are red, OFF-events are blue. (a) shows the events created by a real DVS, and (b) the events created by a simulated DVS [17].

3.1.3. Maze Environment

The V-REP scenario that the snake navigates through consists of an eight-shaped maze, which is depicted in Figure 3.4. Each straight segment is $10m$ long, the distance between the walls is set to $5m$, and the angle between two segments is $135°$ as depicted in Figure 3.5. The idea behind the design is that the snake encounters as many right as left turns if it alternates its starting direction every reset. For a whole round in the positive direction, the snake first needs to turn left, then six times right, two times left, six times right and finally left again. If it reaches the starting position again, the simulation resets, and the snake starts in the other direction. The other reset condition is met if the snake's distance to the middle position between the two walls is greater than $2.3m$. If the snake stayed entirely in the middle between the two walls for a whole round, the length of the covered track would amount to $160m$.
The SNN controller is trained on the eight-shaped maze with a $2m$ high wall. For testing, the wall heights of the maze are varied to evaluate the trained controller's robustness to new situations, which will be explained in more details in section 4.2. Additionally, the controller is tested on zig-zag and cross-shaped mazes with varying angles between two straight segments. What all scenarios have in common is that a $0.5m$ high orange checkered strip is attached to the bottom part of the wall as a marker for the snake.

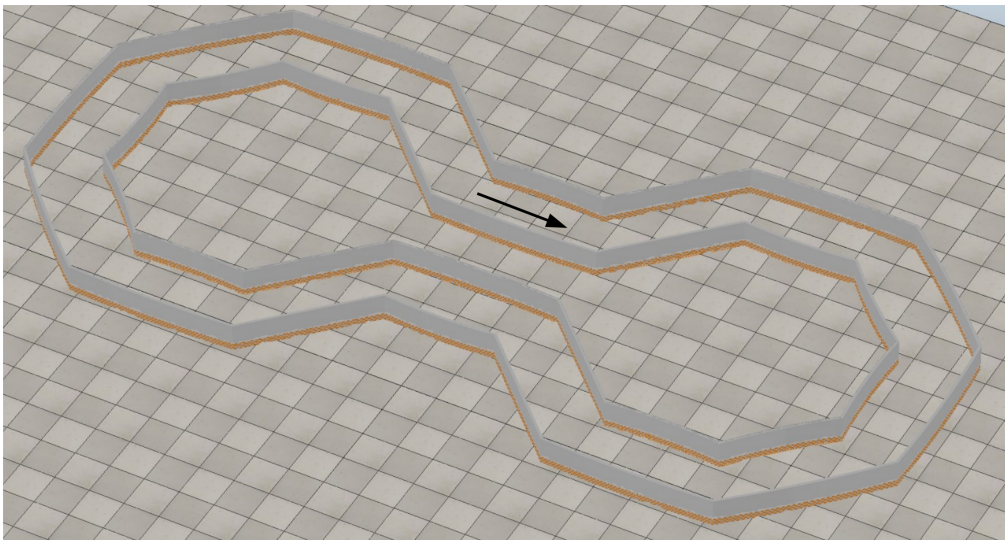


Figure 3.4.: The eight-shaped maze in V-REP shown from above. The arrow points in the positive travel direction. The snake starts alternately in the positive and negative direction for training so that it encounters as many right as left turns.

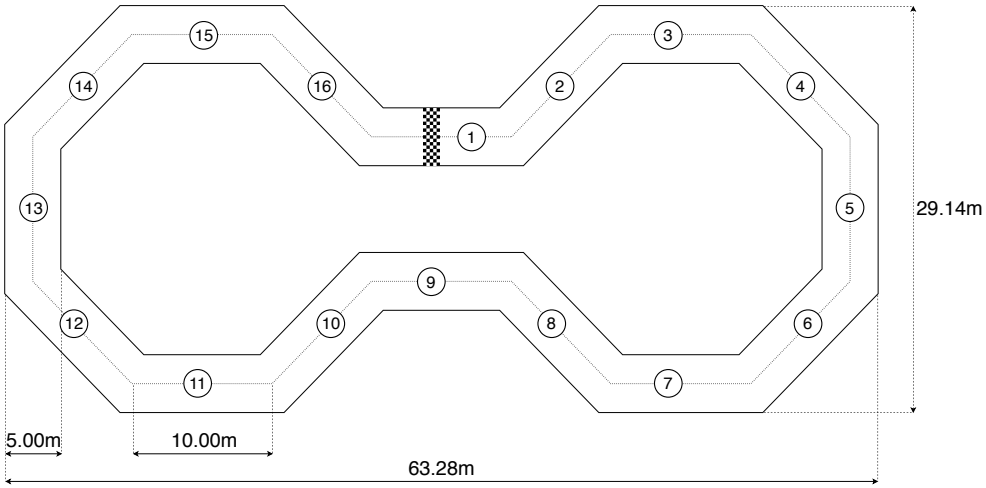


Figure 3.5.: Dimensions of the eight-shaped maze. The segments are numbered increasingly for the positive travel direction. The snake finishes the maze successfully if it crosses the finish line. The path length of the middle line amounts to $160m$.

3.2. Spiking Neural Network Controller

The SNN controller is part of a framework that consists of three Python 2.7 modules - *environment.py*, *controller.py* and *network.py*, which is shown in Figure 3.6. The environment module communicates with V-REP via ROS nodes as explained in section 3.1. The SNN is implemented in the network module and uses the simulation kernel of Neural Simulation Technology (NEST) [31].

The user runs the controller module that calls the other modules to execute the necessary steps to simulate the SNN. First, the network weights of a trained SNN are read and set before the state, which is a preprocessed version of the DVS frame, and the reward value are initialized. Then as the first step of a loop, the network is run for 50ms taking the current state as input and returning the number of output spikes and weights (see section 3.2.1 for further details on how the SNN simulation works). These are then fed into the step function of the environment module that communicates with the V-REP simulation and returns a new state, distance, position, reward, termination, and step number (see section 3.2.2 for further details). As a final step of the loop, the current position and distance of the snake to the center are stored. If the total amount of steps is reached, the loop breaks, the performance data are saved, and the controller terminates. The functionality of the module is depicted in Figure 3.7 as a flowchart.

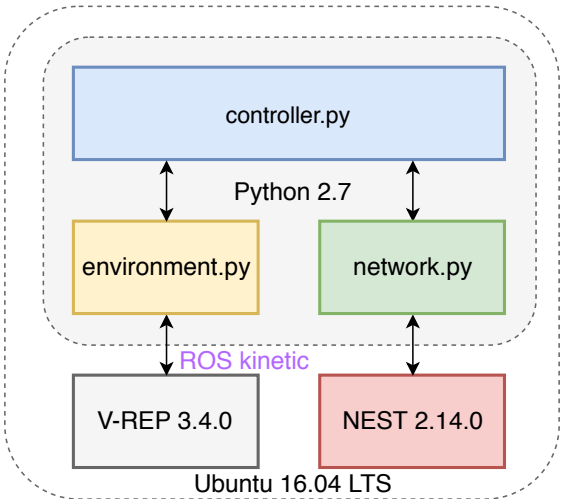


Figure 3.6.: The simulation framework consisting of three Python modules that interact with NEST for the SNN simulation and V-REP for the snake and DVS simulation.

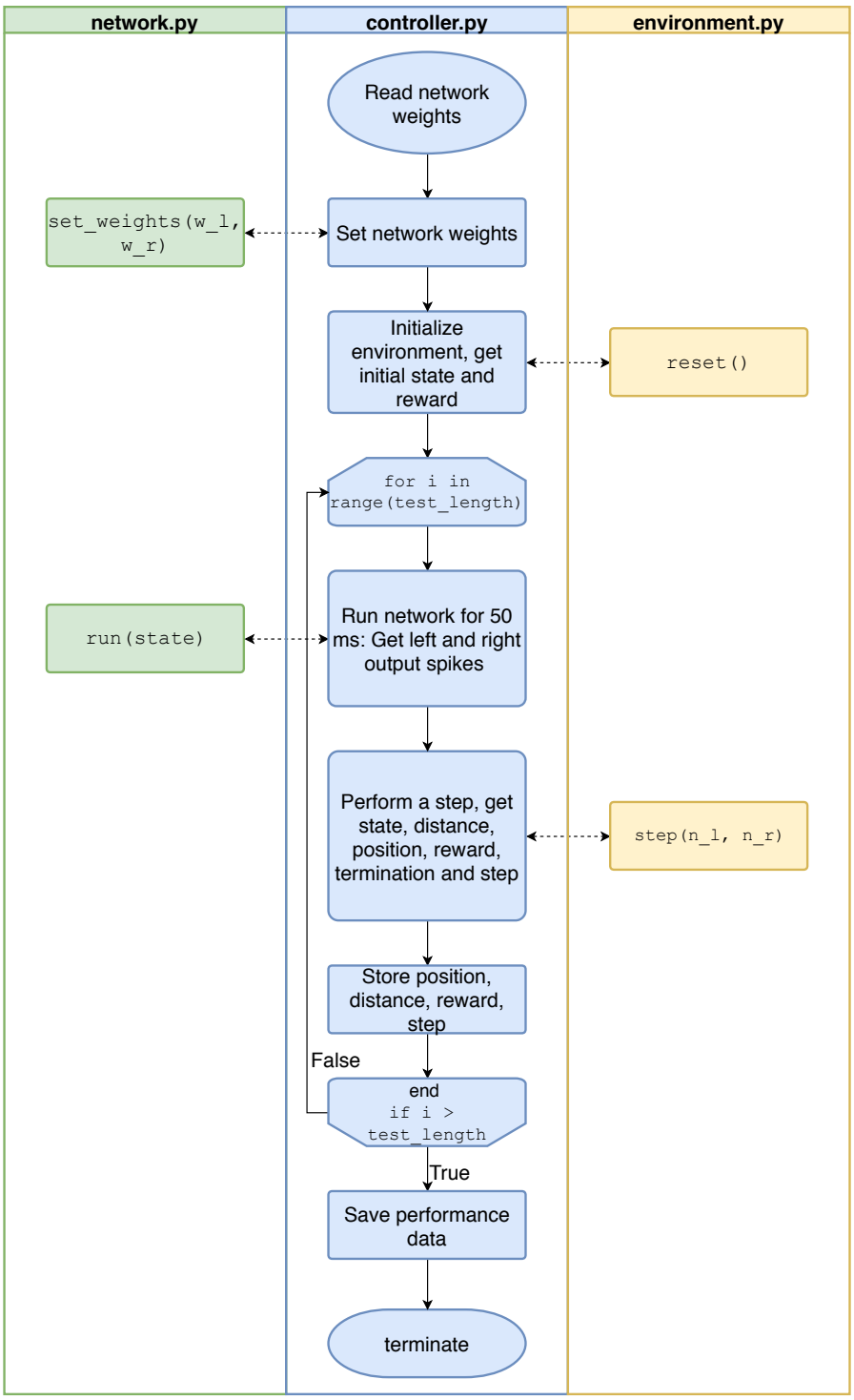


Figure 3.7.: The controller module depicted as a flowchart.

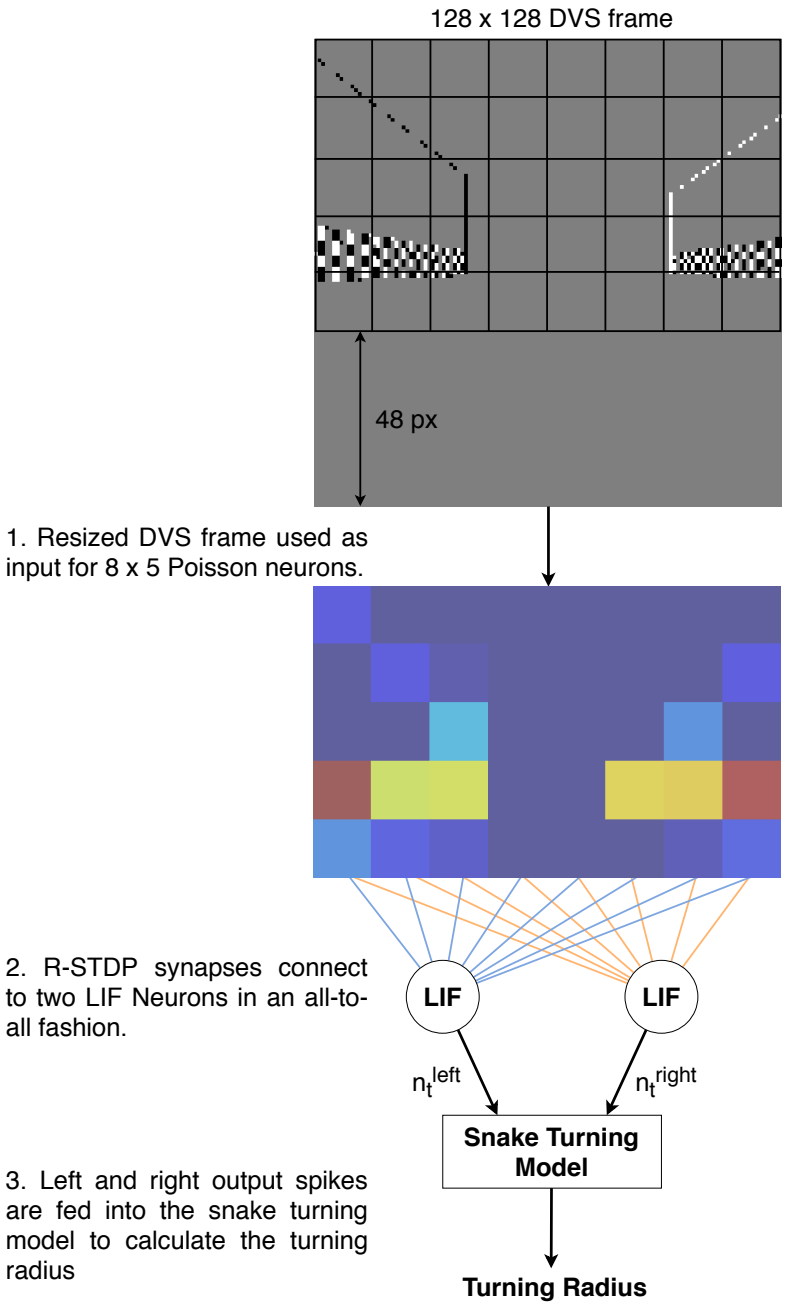


Figure 3.8.: Network architecture of the R-STDP implementation using DVS frames as input.

3.2.1. Network

Architecture

The SNN receives the state and reward as inputs, whereby the state is a preprocessed DVS frame. The frame is first cropped at the bottom by $48px$ because there is no information in this area as the floor is set to be invisible for the DVS to reduce noise. Then, a tile array of $16px \times 16px$ is united to a new pixel that has the value of the total count of events in this region regardless of polarity. Thus, a new $8px \times 5px$ frame is created, in which each pixel is used as an excitation signal for a Poisson neuron. The Poisson neurons connect to two LIF neurons that are the output neurons of the network in an all-to-all fashion via R-STDP synapses. The numbers of spikes of the two output neurons are fed into the snake turning model of the environment module to calculate the turning radius for the next simulation step (see section 3.2.2 for a detailed explanation of the turning model). The network architecture is illustrated in Figure 3.8.

Simulation Step

During one step of the SNN run-function the Poisson neurons fire first. A Poisson neuron converts the input pixel values into spikes by generating a spike train with exponentially distributed intervals between spikes. The pixel value is hereby the average firing rate in Hz. NEST then runs the network for $50ms$ and returns the number of spikes of the two output neurons. The run function and its interactions with NEST are depicted in Figure 3.11a as a flowchart.

3.2.2. Environment Module

The environment module is the link between the SNN and the V-REP simulation. It implements the snake turning model, the reward calculation, and the communication via ROS nodes.

Snake Turning Model

The turning model used in this work is inspired by the implementation by Kaiser et al., which is based on an agonist-antagonist muscle system [17].
The turning radius of the snake is defined as

$$radius(t) = \frac{r_{min}}{turn_{pre}(t)}, \tag{3.4}$$

where r_{min} is the minimum turning radius and $turn\_pre$ is a scaling parameter whose calculation is explained next.

As a first step, the numbers of left and right output spikes are normalized by n_{max}.

$$m_{l/r}(t) = \frac{n_{l/r}(t)}{n_{max}} \in [0,1], \text{ with } n_{max} = \frac{T_{sim}}{T_{refrac}}, \tag{3.5}$$

where $n_{l/r}$ are the number of left and right output spikes, n_{max} is the maximum number of output spikes, T_{sim} is the simulation time, and T_{refrac} is the refractory period of the neuron.
Kaiser then defined the steering angle of his turning model as

$$a(t) = m_l(t) - m_r(t) \in [-1;1]. \tag{3.6}$$

As the environment of the snake is continuous, the turning radius of the snake should be continuous too, which is why equation 3.6 is smoothed as shown in 3.7:

$$turn_{pre}(t) = c(t) \cdot a(t) + (1 - c(t)) \cdot turn_{pre}(t-1), \text{ with } c(t) = \sqrt{\frac{(m_l(t))^2 + (m_r(t))^2}{2}}. \tag{3.7}$$

After $turn_{pre}(t)$ is calculated, the radius is published via the *turningRadius* ROS node to the Lua script of the snake.
Figure 3.9 illustrates the radius calculation. A high difference between the left and right output spikes results in low turning radius, which makes sense as the snake should strongly turn if there are a lot of DVS events on one side.

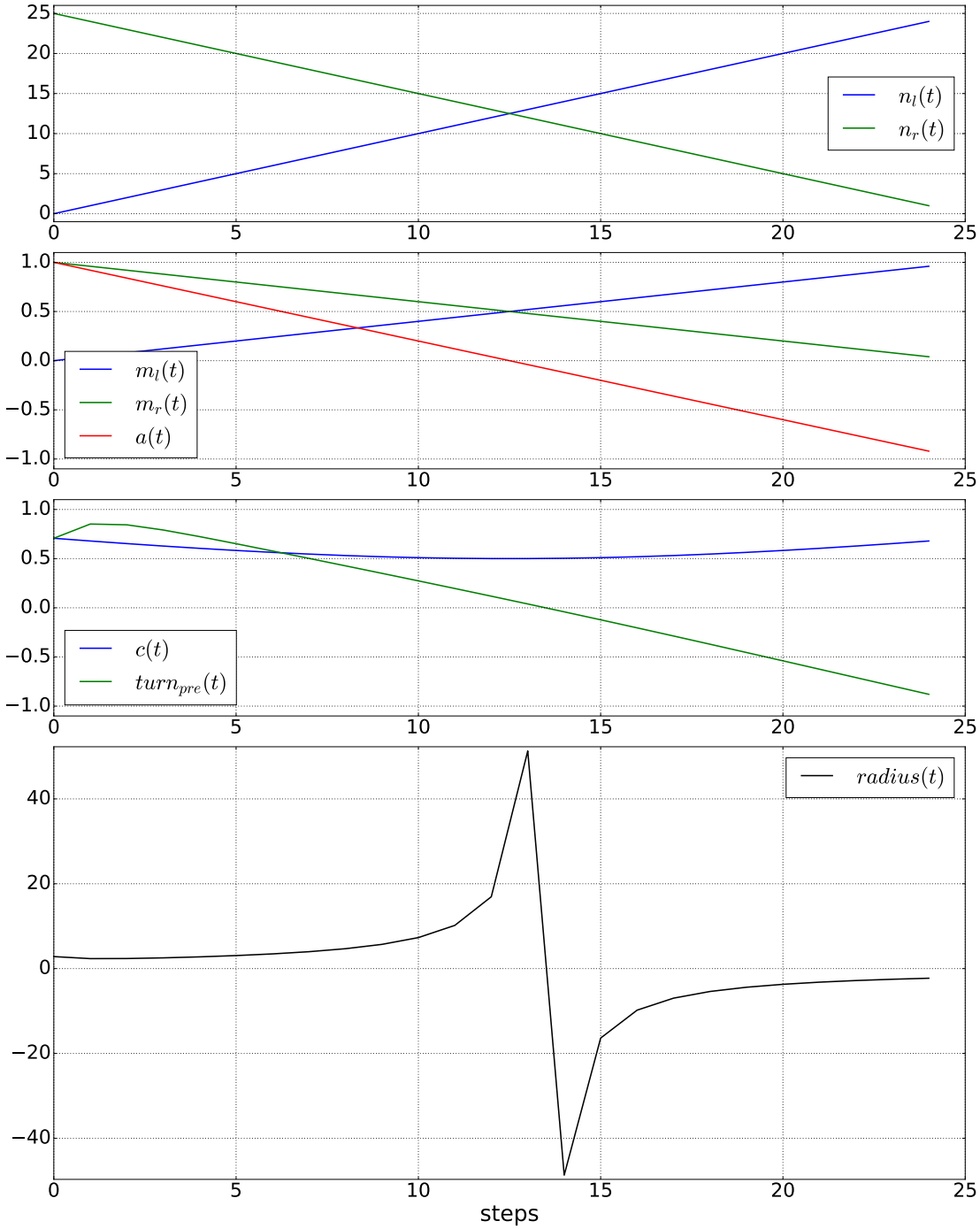


Figure 3.9.: The radius calculation in the snake turning model. n_l and n_r come from the network module, and are assumed to linearly increase and decrease functions for demonstration purposes.

Reward Calculation

After the turning radius is published, the reward signal for the SNN is set. The reward modulates the neurotransmitter concentration of the R-STDP synapses, and is defined with the following considerations: If the snake is close to the left part of the wall, it needs to turn right to stay in the middle position. Therefore, the left output neuron needs to fire more spikes than the right neuron. Thus, the neurotransmitter concentration of the left neuron should be high, and the concentration for the right one low. Additionally, the side-to-side movement of the head of the snake through the slithering motion should not cause the snake to change its turning radius if it is in the middle. These considerations lead to the reward signal being defined as a cubic function dependent on the distance to the center position between the two walls such that it has low absolute values in the middle and high absolute values at the edges:

$$reward_{left/right} = -/+ \left(d^3 \cdot c_r\right), \tag{3.8}$$

where d is the distance to the center position and c_r is a scaling constant. The reward function is depicted in Figure 3.10.

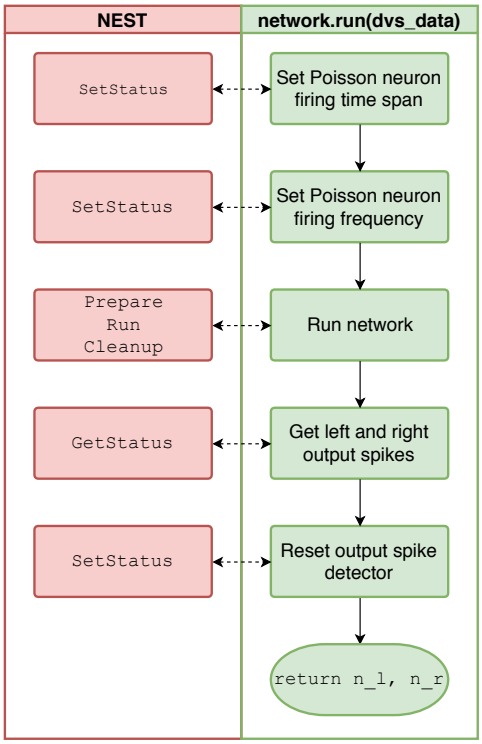
Figure 3.10.: Reward given by the R-STDP controller with $c_r = 0.00025$, which is the value used in the simulation.

Step Function

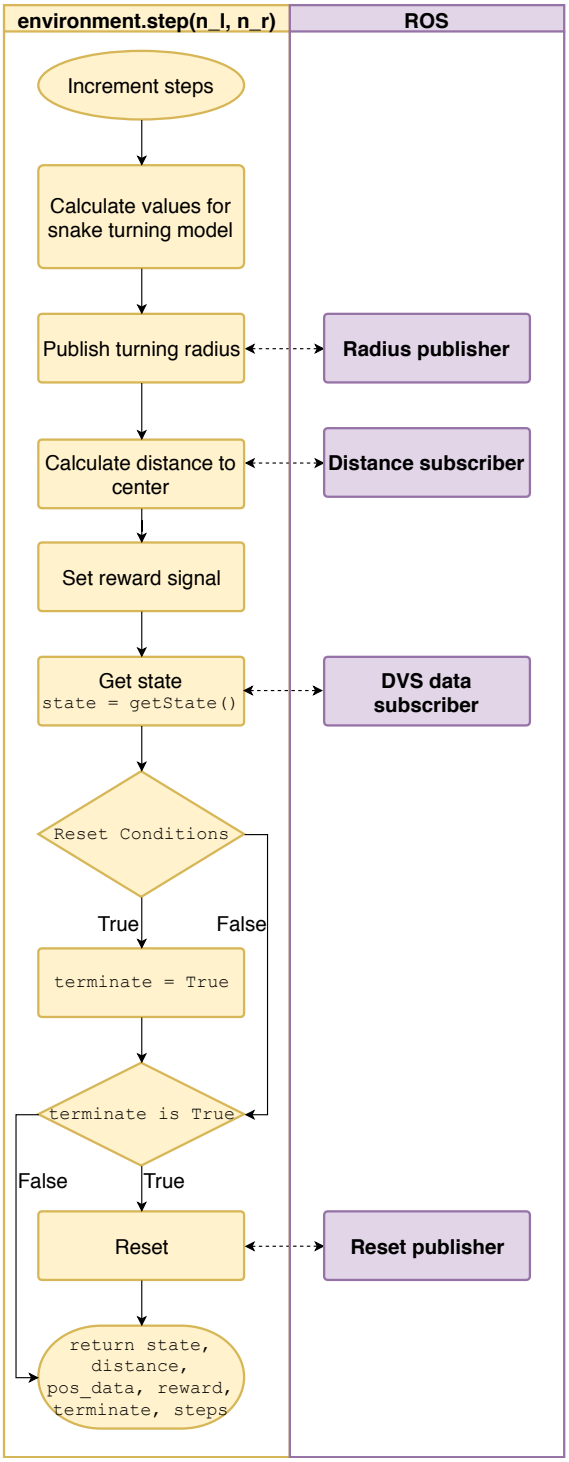
The step function of the environment module implements all necessary steps to communicate with the V-REP simulation. It is executed every simulation step, and first, it calculates the values for the snake turning model as explained in section 3.2.2. After publishing the turning radius, the distance of the snake to the center position is computed. As the next step, the reward is calculated as explained in section 3.2.2 and published. Then, the state is updated by preprocessing the DVS camera image received via the *dvsData* node as explained in section 3.2.1. As a final step, the module checks whether the distance of the snake to the center is too large, or whether it has reached the starting position again after making a full round. If one of the conditions is met, the simulation is reset via the *resetRobot* ROS node. The step function terminates after returning the updated state, distance, position data, reward, termination, and step number. Figure 3.11b depicts the process as a flowchart.

3.3. Training Workflow

The training workflow differs in some details from the one for the controller described in Section 3.2. Whereas the controller module is used to test a set of learned synaptic weights, the training module's purpose is to generate this set beforehand. So for training, the training module takes the place of the controller module in the framework (see Figure 3.6). Instead of the run() function of the network module that simulates a SNN with NEST without changing the set of synaptic weights, the simulate() function of the network module is used that adjusts the weights according to the R-STDP learning rule. Besides the preprocessed DVS frame it also takes the reward signal as an input that changes the neuromodulator concentration of the R-STDP synapses. The higher the concentration, the stronger the change in synaptic weight. The flowcharts for the training module and the simulate function are depicted in the Appendix in Figure A.1 and Figure A.2.



(a) Run function in network.py.



(b) Step function in environment.py.

Figure 3.11.: Flowcharts showing the run and step function.

4. Discussion

The SNN is evaluated in terms of its capabilities to cope with new scenarios in this chapter. The training performance is discussed in Section 4.1 before testing the controller in Section 4.2 on scenarios that have different wall heights and different angles between segments.

4.1. Training

The SNN controller is trained on the eight-shaped maze with $2m$ high walls, and it successfully finishes the maze in both directions after two episodes terminated early. Figure 4.1 shows the traveled steps per episode for the training. The snake is reset in the first two episodes after 287 and 154 steps because it moves too close to the wall, but from the third episode the snake makes the full round and reaches the starting area again taking around 5,350 steps per episode.
In Figure 4.2 the development of the synaptic weights connecting to the left and right motor neurons and the reward that is fed into the SNN are plotted over the simulation time. The most substantial changes happen at the beginning of the training when the reward and thus the neuromodulator concentration have the highest absolute values. First, the right motor neuron weights are changed excitatory because the snake encounters a left turn in the first episode that leads to a lot of DVS events on the right half of the input frame. Additionally, the snake moves close to the inner wall resulting in a high reward value. The combination of a lot of DVS events and a large positive reward value lead to substantial excitatory weight changes for the right motor neuron synapses. For the synapses connecting to the left motor neuron, it is the other way round: large negative reward values lead to excitatory synaptic changes for the left motor neuron and inhibitory changes for the right one as can be seen in the Figure right before the second episode resets. Another change happens at the beginning of the third episode, which is similar to the second change with falling weight values for the right motor neuron and rising values for the left one. After these initial changes, the weights only change slightly as the snake gets less reward by staying close to the middle line, and it finishes each episode reaching the starting position again.
Figure 4.3 shows the final distribution of the synaptic weights after the training. The middle right and middle left areas have the most significant influence on the output

neurons, which makes sense from the controller's perspective. If, for example, the snake is too close to the right wall, it needs to turn left in order to return to the middle position. Turning left is achieved if the right motor neuron fires more spikes than the left motor neuron, explaining why the right half of the synaptic weights connecting to the right motor neuron are high - excitatory, and their counterparts connecting to the left motor neuron are low - inhibitory. Vice versa for the left weights.
The final simulation parameters for the training can be found in the Appendix in Table A.1.

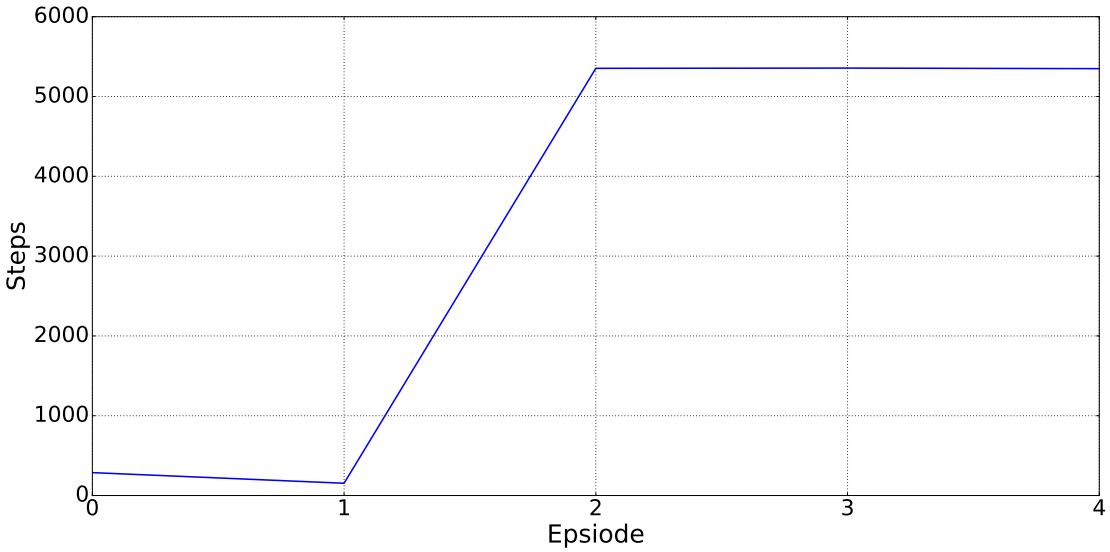


Figure 4.1.: Steps per episode plotted for the training. The snake successfully finishes a full round of the maze after two early resets when it hit the wall.

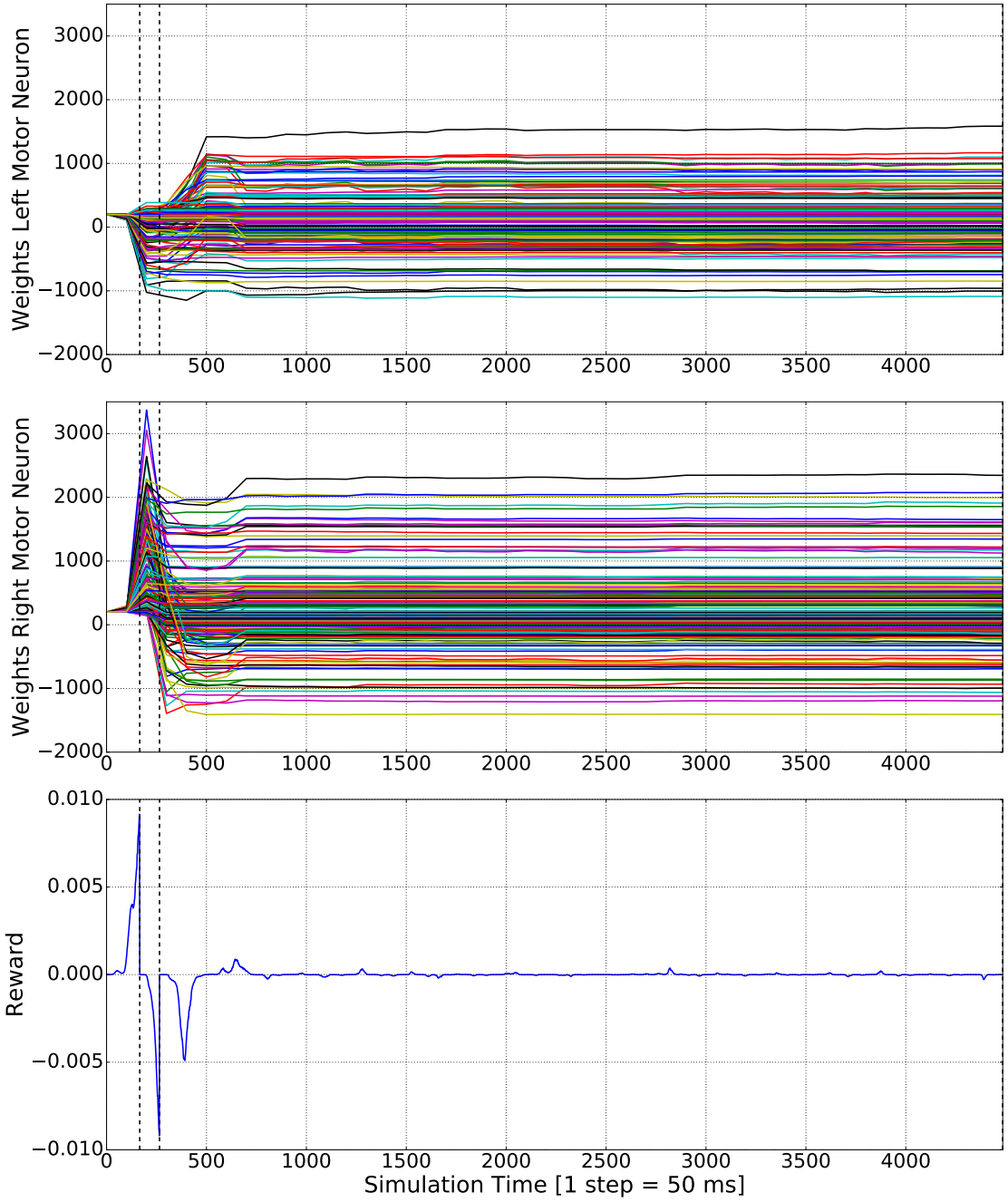


Figure 4.2.: Weights over simulation time plotted for the controller. The vertical lines mark new episodes, and only the first three epsiodes are shown as the weights do not change significantly after step 1,000.

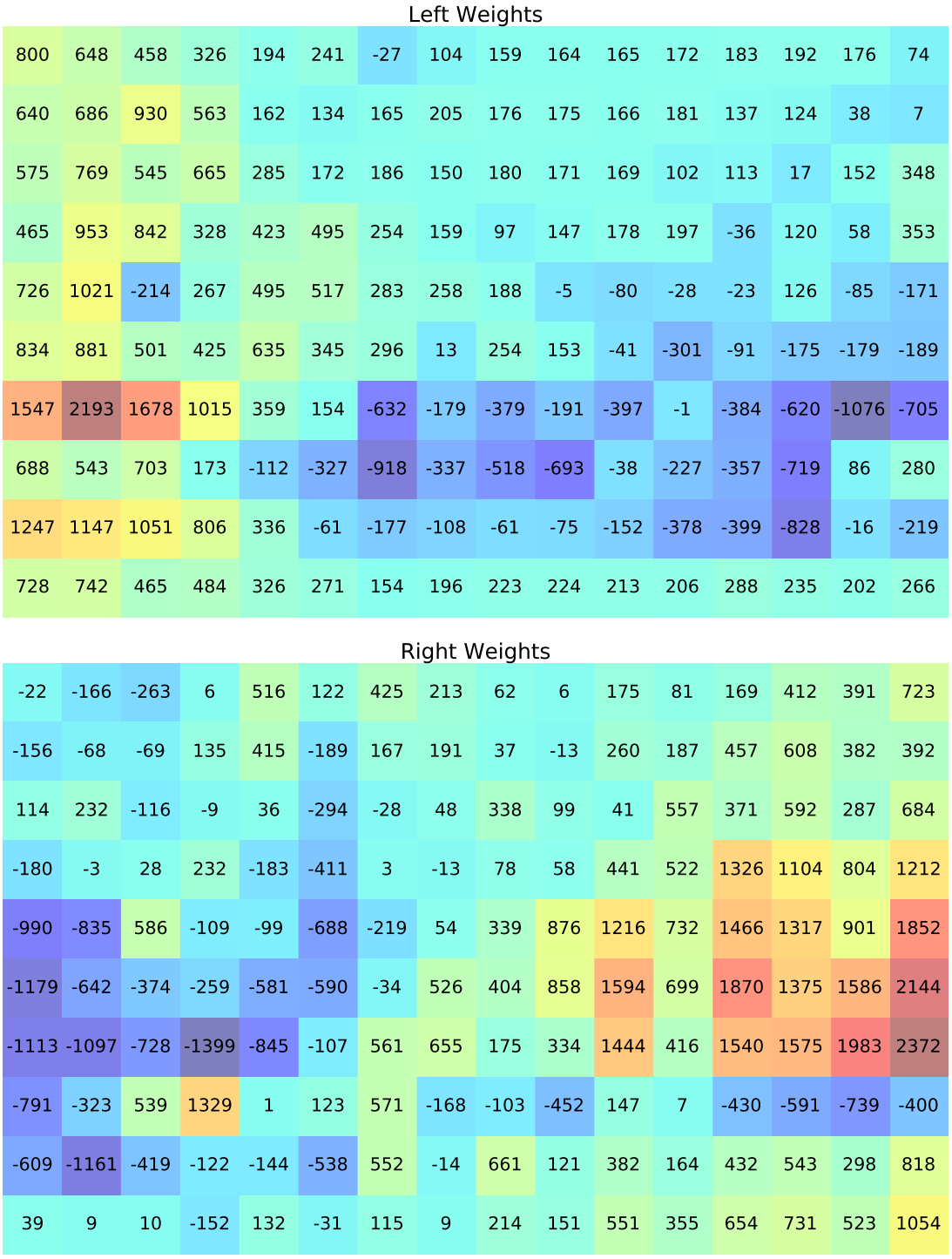


Figure 4.3.: Final weights of the controller.

4.2. Testing

The controller that uses the synaptic weights described in Section 4.1 is tested on six V-REP scenarios with varying heights in Section 4.2.1 and on four scenarios with steeper angles in Section 4.2.2.
The controller's performance is evaluated based on the following values: First, the mean of the distance to the middle position between the two walls over one episode is determined. This value reveals whether the snake traversed the maze moving closer to the inner or outer wall as the distance is defined such that it takes on positive values if it is close to the inner wall and negative values if it is close to the outer wall. Second, the mean of the absolute values of the distance to the middle position is defined as an error value to asses how well the controller manages to stay in the middle position. The higher the error, the more the snake deviated from the middle over the course of one round. And third, the length of the traveled distance of the snake over one round is calculated.

4.2.1. Different Wall Heights

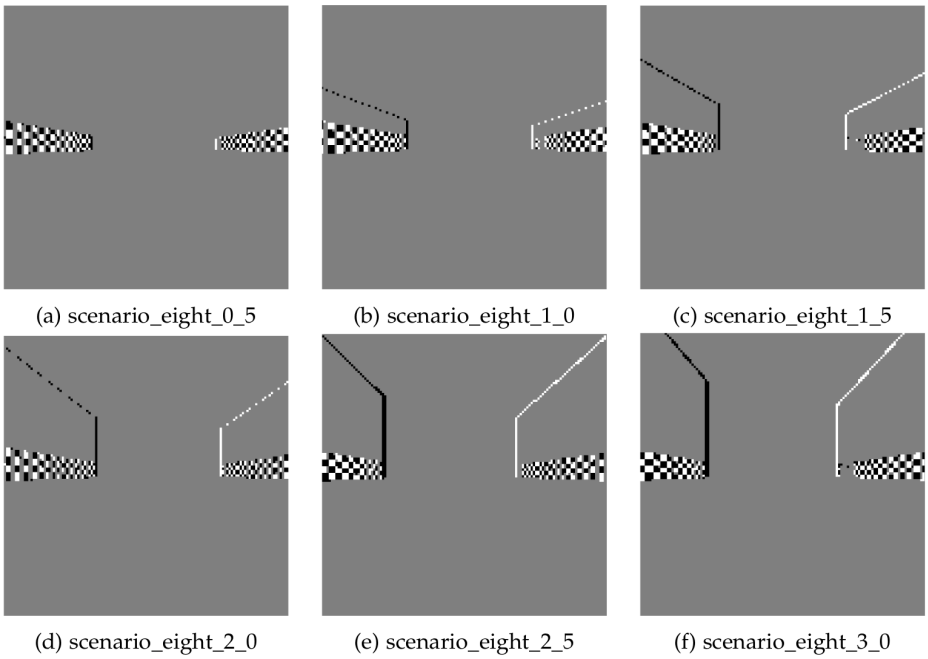
As a first evaluation of the controller's ability to cope with new situations, the snake is tested on six eight-shaped maze scenarios with wall heights that vary from $0.5m$ to $3m$. These heights are chosen for evaluation as they could be encountered in a real-world deployment of the robot, for example, in a collapsed factory building. The six scenarios are referred to as *scenario_eight_0_5*, *scenario_eight_1_0*, etc. whereby the number subfix depicts the wall height. Figure 4.4 shows DVS frames from the six different scenarios.

As Figure 4.5 shows, the controller is able to finish a full round of the eight-shaped maze in each scenario. One can see the course over the maze the snake takes in the distance over steps plot of *scenario_eight_3_0*, for example. Whereas during left turns, which take place at the first, eighth, ninth, and last turn the snake moves closer to the outer wall resulting in a negative distance value, during right turns the snake moves closer to the inner wall resulting in a positive distance. The snake's positions in the maze are also depicted in Figure 4.6 where the paths of the snake from the different scenarios are laid on top of each other.
For a better comparison, the mean value, the error value, and the length of the traveled distance are displayed in Table 4.1. The test on *scenario_eight_2_0* can be seen as the comparative basis because the snake is trained on this scenario, and thus reaches the lowest value for the error with $0.329m$. Generally, the following trends are observable: First, the higher the wall, the lower the mean value. A high mean value means that the snake travels close to the inner wall and thus needs less distance to finish one round of

the maze. This explains the second trend: the higher the wall, the longer the traveled distance. This behaviour can also be seen in Figure 4.6 where the blue path results from the snake in *scenario_eight_0_5* moving close to the inner wall and thus needing less distance to finish the maze, and the yellow path from the snake in *scenario_eight_3_0* resulting in a higher traveled distance value.
And third, the more wall height gets removed (in case of *scenario_eight_1_5*, *scenario_eight_1_0* and *scenario_eight_0_5*) or added (in case of *scenario_eight_2_5* and *scenario_eight_3_0*), the greater the error becomes having its highest value for *scenario_eight_0_5* with 0.624m.

	Wall Height [m]					
	0.5	1.0	1.5	2.0	2.5	3.0
Mean [m]	0.617	0.469	0.350	0.299	0.270	0.183
Error [m]	0.624	0.483	0.367	0.329	0.331	0.356
Length [m]	162.81	165.90	167.99	169.49	170.73	172.12

Table 4.1.: Comparison of the mean value, error value and length of the traveled distance for the eight-shaped maze scenarios.



(a) scenario_eight_0_5 (b) scenario_eight_1_0 (c) scenario_eight_1_5

(d) scenario_eight_2_0 (e) scenario_eight_2_5 (f) scenario_eight_3_0

Figure 4.4.: DVS frames from the scenarios with different wall heights.

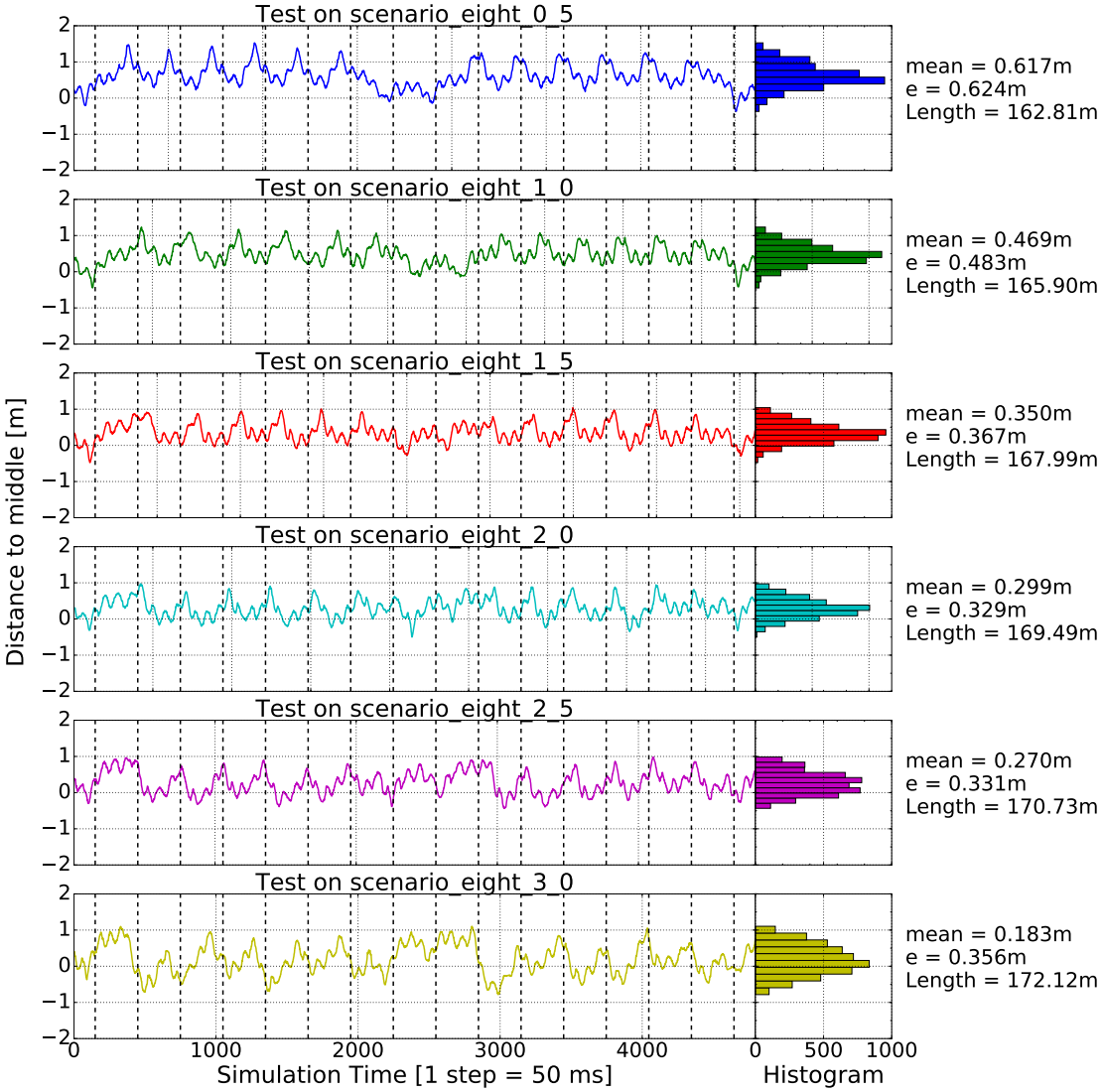


Figure 4.5.: Plot of the distance to the middle position over the traveled distance for the scenarios with different wall heights. The dashed vertical lines mark the 16 turns in the maze whereas the first one is a left turn, followed by six right, two left, six right, and a final left turn until the snake is at the starting position again.

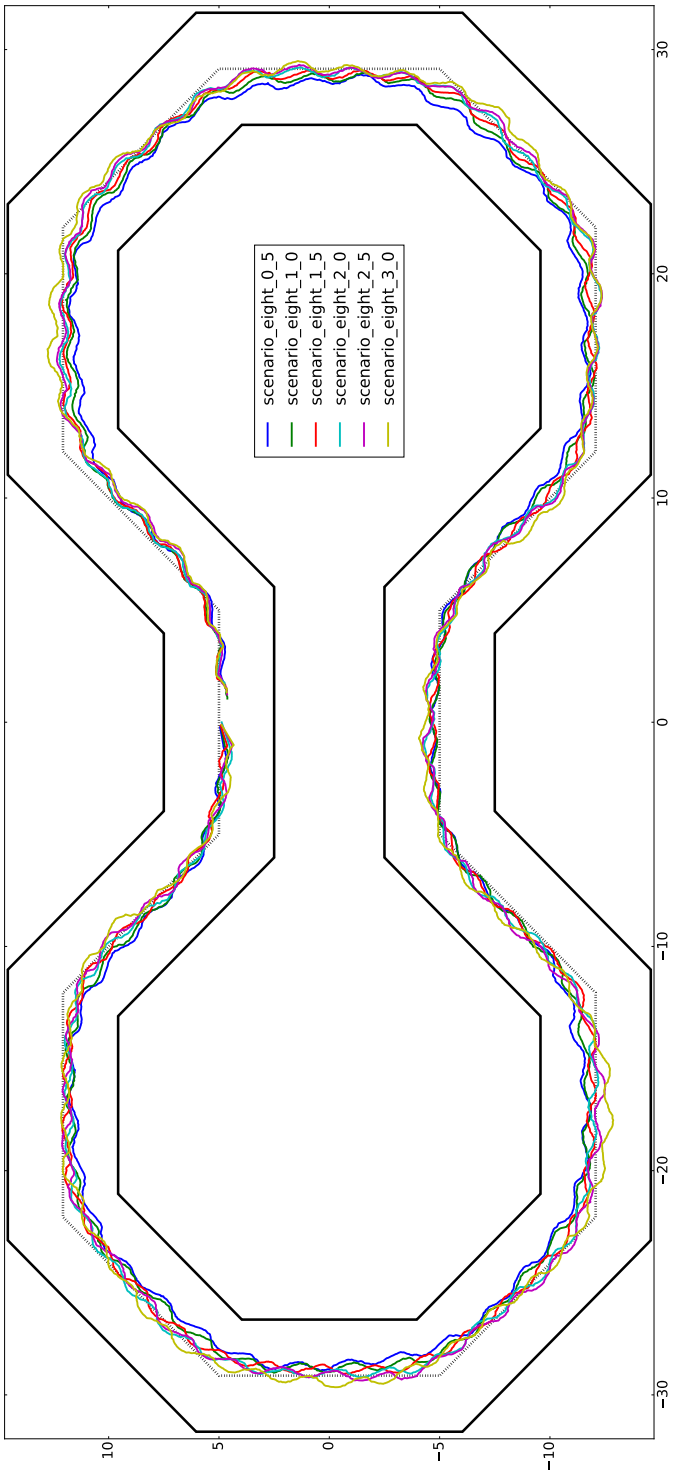


Figure 4.6.: The paths of the snake in the different wall heights scenarios laid on top of each other.

4.2.2. Different Angles

To further evaluate how the learned controller from Section 4.1 copes with new situations the angle between the straight segments of the maze is decreased.

Zig-Zag Shaped Maze

First, the snake is tested on a zig-zag shaped maze with $2m$ high walls that is designed such that the angle between two segments decreases every two turns by 5° from 135° to 90° as depicted in Figure 4.9. The snake is able to turn 15 times until an angle of 100°. During the first turns, the snake follows the middle line closely, but the steeper the angles get, the larger the deviation becomes until it is not able to take the turn, and the simulation resets. This behavior can also be observed in Figure 4.7, in which the distance to the middle gets larger the longer the snake travels.

Cross Shaped Mazes

Finally, the snake is tested on four mazes that are cross-shaped with top angles that vary from 110° to 95° as shown in Figure 4.10.
Figure 4.8 shows that the snake is able to finish one round of the maze for each of the scenarios. The mean, error and length values are depicted in Table 4.2.
The steeper the angle gets, the higher the error value becomes ranging from $0.394m$ for 110° to $0.564m$ for 95°. That the snake deviates more and more from the middle position can also be seen from the length of the traveled distance values that decrease from $122.35m$ to $118.89m$. They go down because the snake cuts corners more and more the steeper the angle becomes. Whereas for *scenario_cross*_110 the snake is able to follow to middle line closely, for *scenario_cross*_95 it deviates from the optimal position as can be seen in Figure 4.10.

	Top angle			
	110°	105°	100°	95°
Mean [m]	0.127	0.154	0.235	0.264
Error [m]	0.394	0.484	0.560	0.564
Length [m]	122.35	119.77	118.77	118.89

Table 4.2.: Comparison of the mean, error, and length of the traveled distance for the cross shaped maze scenarios with top angles ranging from 110° to 95°.

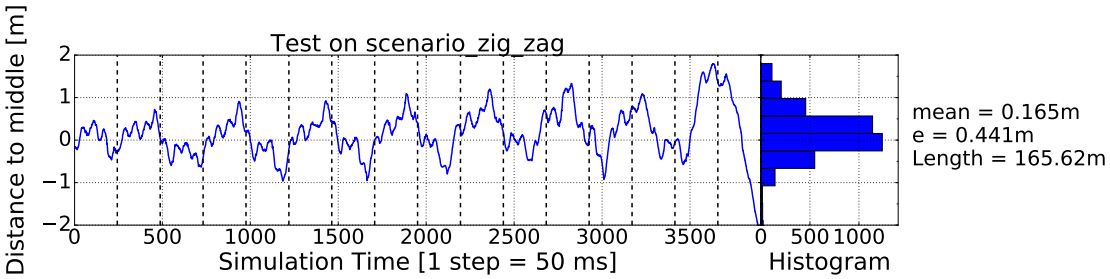


Figure 4.7.: Performance of the controller shown as a distance over steps plot and a histogram for the zig-zag shaped maze scenario. The dashed vertical lines mark the 15 turns the snake is able to take.

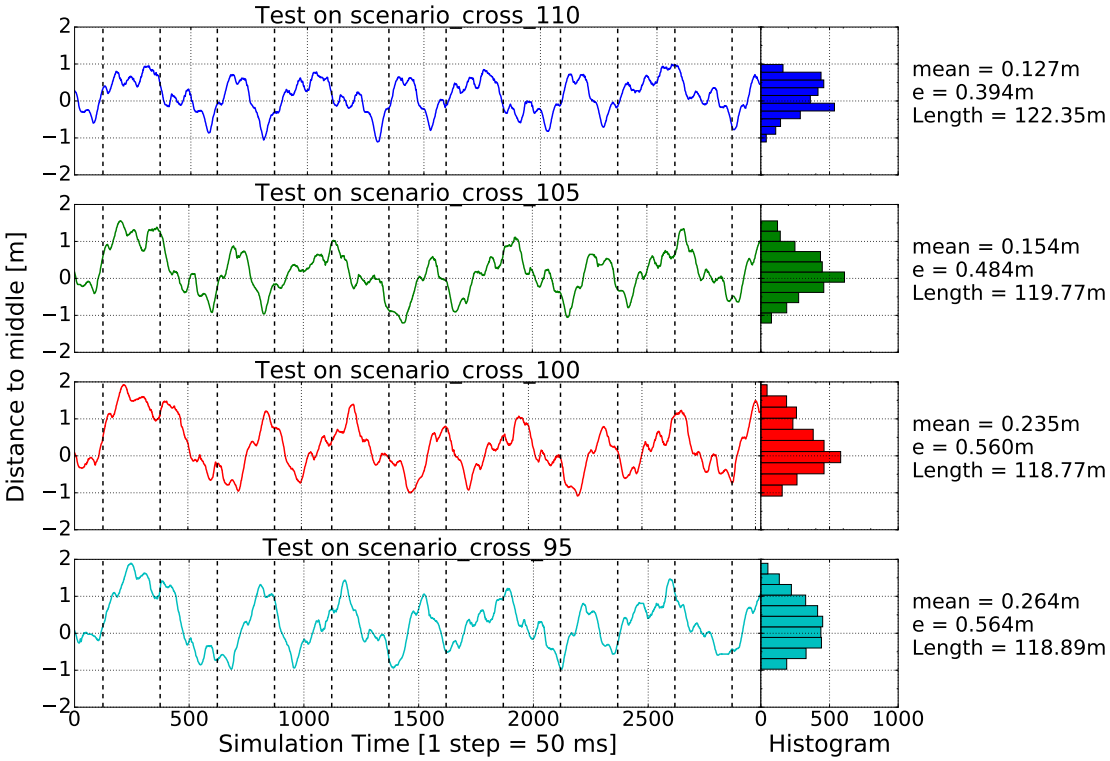


Figure 4.8.: Performance of the controller shown as a distance over steps plot and a histogram for the cross shaped maze scenario. The dashed vertical lines mark the 12 turns.

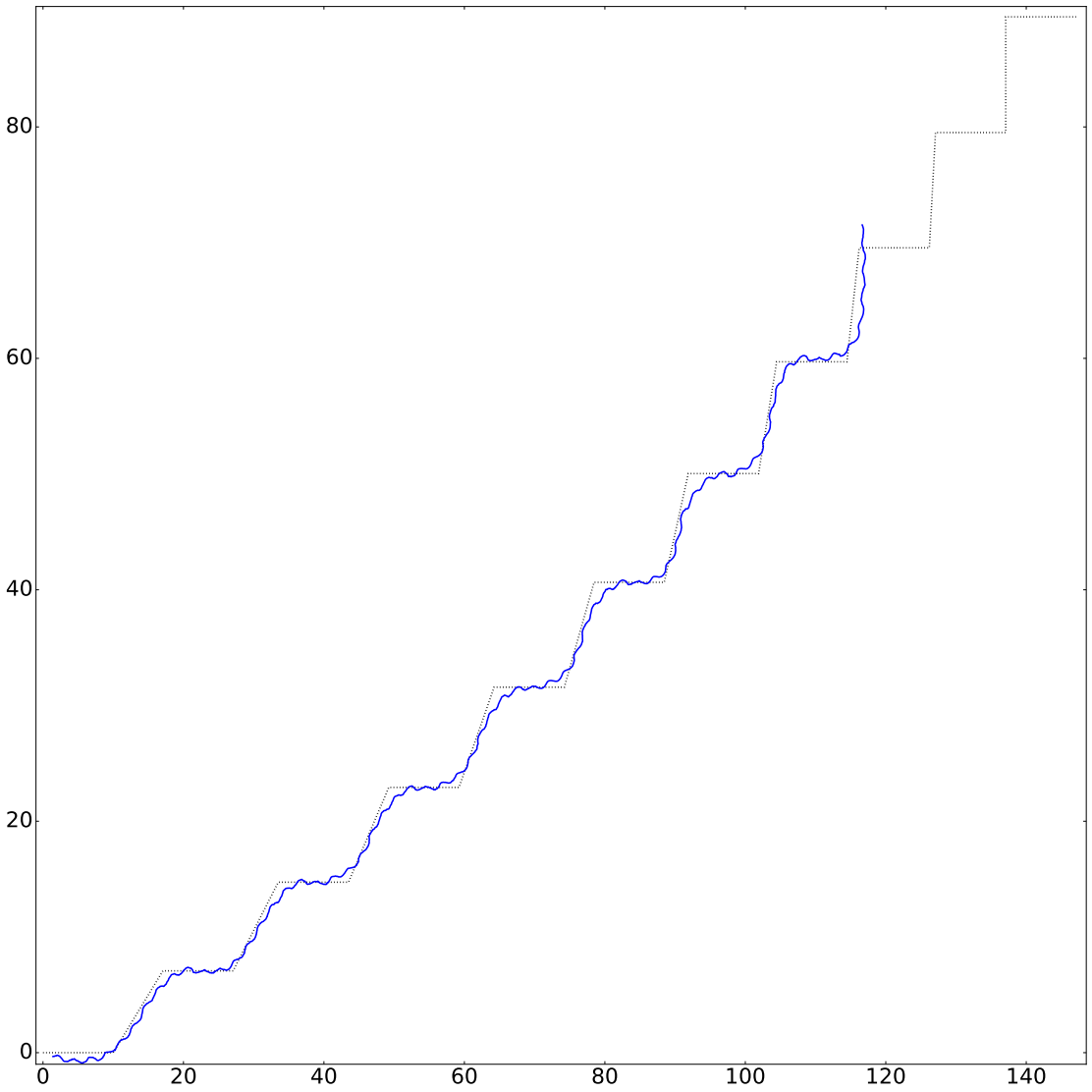
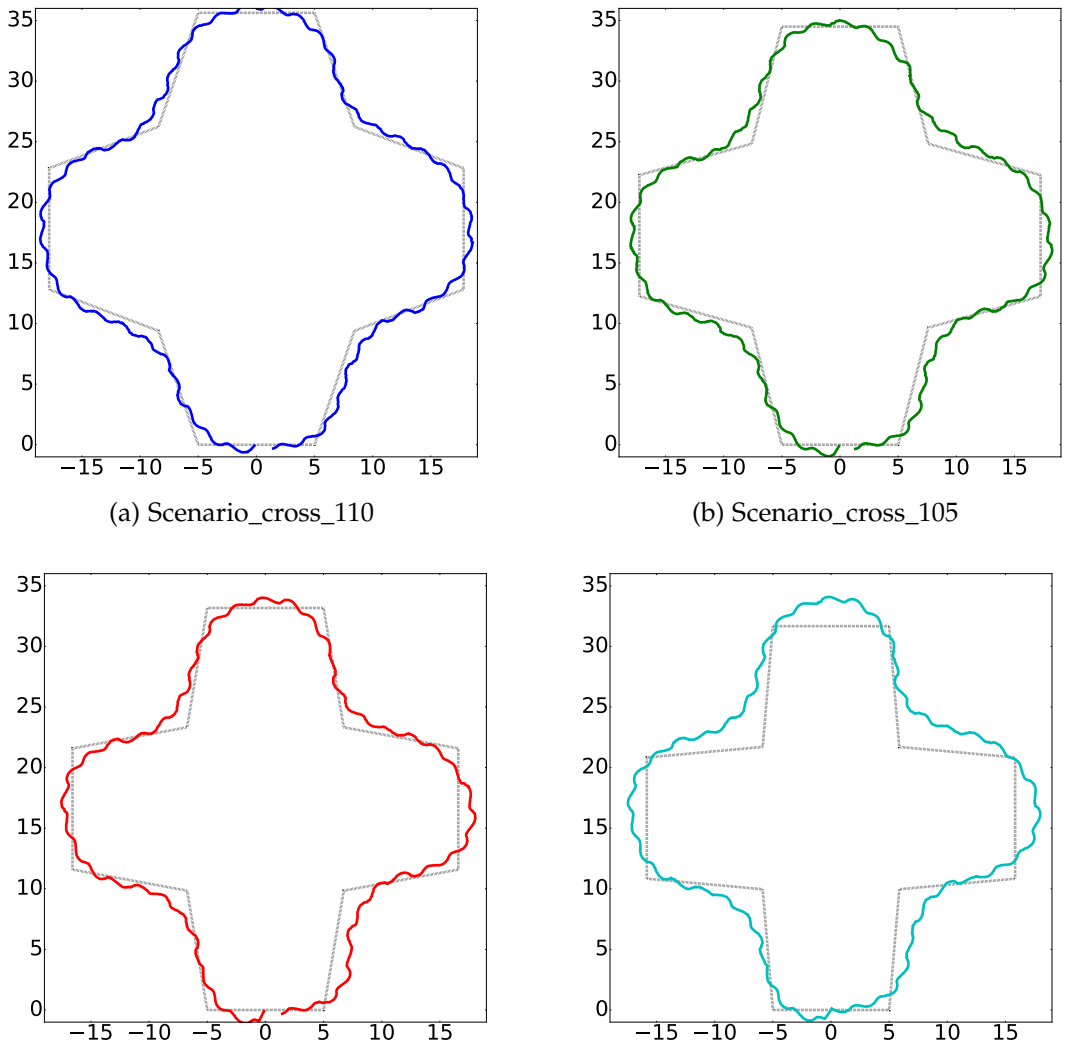


Figure 4.9.: The middle line of the zig-zag shaped maze is plotted as a dotted line. The angle between two straight segments decreases by 5° from 135° to 90° every second turn. A straight segment has a length of $10m$. The path that the snake takes is plotted as a blue line.



(a) Scenario_cross_110

(b) Scenario_cross_105

(c) Scenario_cross_100

(d) Scenario_cross_95

Figure 4.10.: The middle lines of the four cross shaped mazes with decreasing top angles from 110° to 95° are plotted as a dotted line. The snake's paths are plotted in blue, red, green and light blue.

5. Conclusion and Outlook

In this work, a SNN controller for a snake-like robot is implemented, and its ability to cope with new maze shapes demonstrated.
The architecture of the SNN is two-layered. The preprocessed output data of a DVS is used as the excitation signal for Poisson neurons in the first layer, and the number of output spikes from two LIF neurons in the second layer are used to control the snake's turning radius. The weights of the SNN are updated based on the R-STDP learning rule that controls the synaptic plasticity in the network.
The controller is trained on an eight-shaped maze with $2m$ high walls. It manages to successfully finish a whole round of the maze after two early terminated episodes with around 1200 simulation steps. After that, it is tested on different eight-shaped mazes with three lower and two higher wall heights, a maze with decreasing angles between the straight segments, and four cross-shaped mazes with varying angles. The controller proves to be adaptable to new situations as it is able to finish one full round of each of the testing scenarios. The steepest angle between two segments the snake makes are $95°$.
These tests show that the relatively simple two-layered network architecture in combination with the R-STDP learning rule and a DVS as an input is suitable for an autonomous locomotion control task for a snake-like robot. The next step would be to implement this control architecture in the real snake-like robot, and test whether the results from the simulation can be replicated in the real world.

A. Appendix

A.1. Training Flowcharts

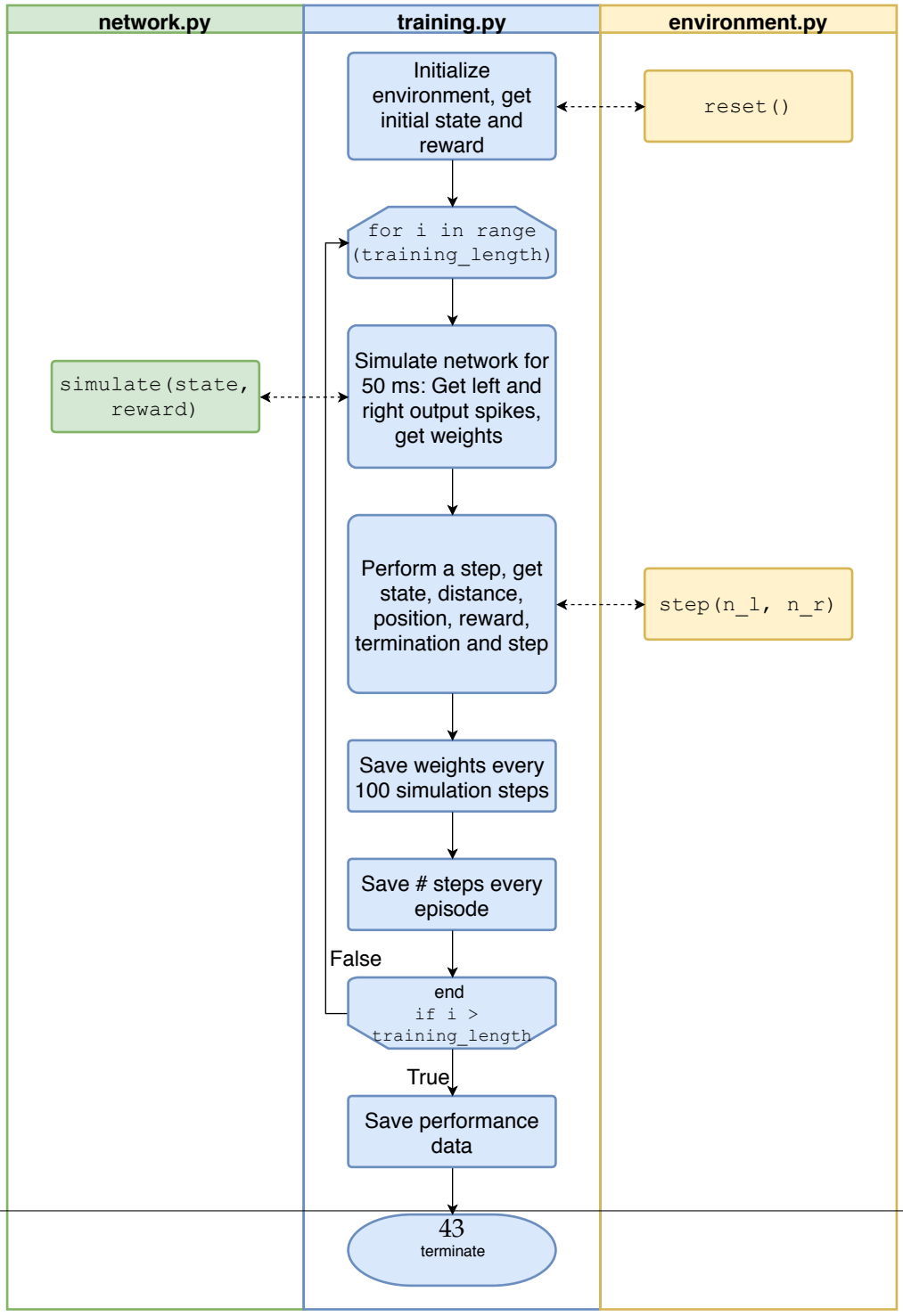


Figure A.1.: The training module depicted as a flowchart.

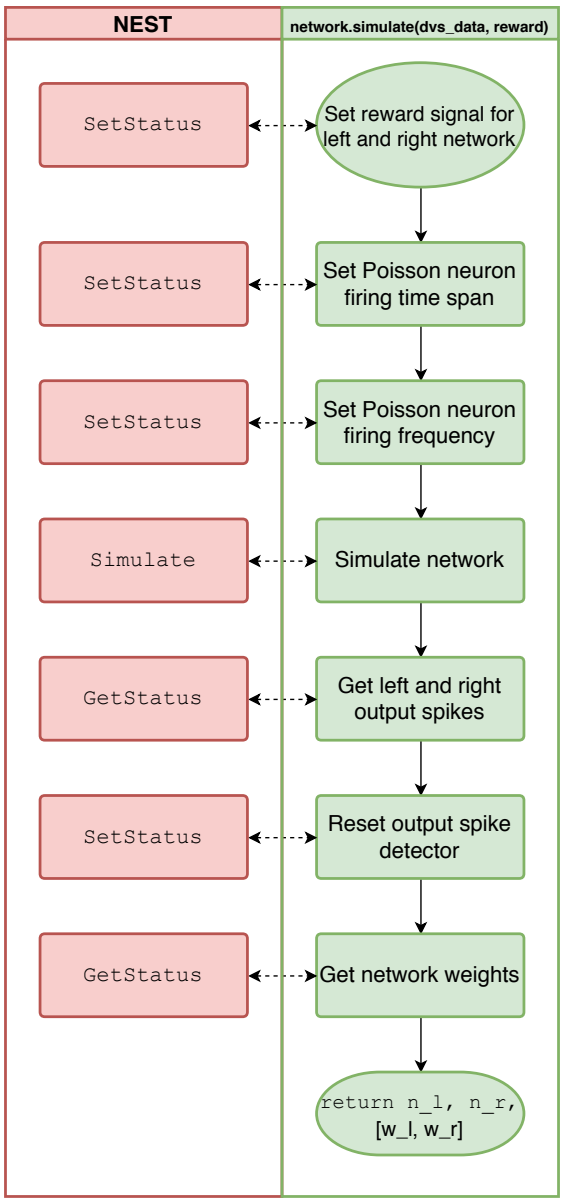


Figure A.2.: The simulate function of the network module as a flowchart.

A.2. Simulation Parameters

Table A.1.: Simulation Parameters

	Parameter	Value	Unit
SNN Input	DVS Resolution	$128x128$	px
	Crop Bottom	48	px
	SNN Input Resolution	$16x10$	px
Network Simulation	Simulation Time per Step	50.0	ms
	Refractory Period	2.0	ms
	Time Resolution	0.1	ms
	Max. Poisson Neuron Firing Frequency	300.0	Hz
	Max. DVS Events per Step	16.0	
R-STDP	Maximum Synapse Value	10000.0	
	Minimum Synapse Value	-10000.0	
	Maximum Initial Random Synapse Value	201.0	
	Minimum Initial Random Synapse Value	200.0	
	Time Constant of Reward Signal	200.0	ms
	Time Constant of Eligibility Trace	1000.0	ms
	Reward Factor scaling the Reward Signal	0.00025	
	Constant scaling Strength of Potentiation	1.0	
	Constant scaling Strength of Depression	1.0	
Snake Turning Model	Initial Radius	0.0	
	Maximum Input Activity	25.0	
	Minimum Turning Radius	2.0	m

Bibliography

[1] L. F. Abbott and Sacha B. Nelson. "Synaptic plasticity: Taming the beast." In: *Nature Neuroscience* 3.11s (2000), pp. 1178–1183. ISSN: 15461726. DOI: 10.1038/81453.

[2] Zhenshan Bing et al. "Towards autonomous locomotion: Slithering gait design of a snake-like robot for target observation and tracking." In: *IEEE International Conference on Intelligent Robots and Systems* 2017-September.September (2017), pp. 2698–2703. ISSN: 21530866. DOI: 10.1109/IROS.2017.8206095.

[3] Hermann Blum, Alexander Dietm, and Moritz Milde. "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor." In: *Robotics science and systems (RSS)* (2017). ISSN: 2330765X. URL: http://rss2017.lids.mit.edu/static/papers/02.pdf.

[4] Erwin Coumans. *Real-Time Physics Simulation*. URL: http://bulletphysics.org/wordpress/. (accessed: 26.06.2018).

[5] Daniel Drubach. *The Brain Explained*. Pearson, 1999. ISBN: 9780137961948.

[6] Etienne Dumesnil, Philippe Olivier Beaulieu, and Mounir Boukadoum. "Robotic implementation of classical and operant conditioning within a single SNN architecture." In: *Proceedings of 2016 IEEE 15th International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2016* (2017), pp. 322–330. DOI: 10.1109/ICCI-CC.2016.7862055.

[7] Richard Evans. "Reinforcement Learning in a Neurally Controlled Robot Using Dopamine Modulated STDP." In: September (2015). arXiv: 1502.06096. URL: http://arxiv.org/abs/1502.06096.

[8] Răzvan V. Florian. "Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity." In: *Neural Computation* 19.6 (2007), pp. 1468–1502. ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.6.1468. URL: http://www.mitpressjournals.org/doi/10.1162/neco.2007.19.6.1468.

[9] Open Source Robotics Foundation. *Robot Operating System*. URL: http://www.ros.org/. (accessed: 26.06.2018).

[10] Wulfram Gerstner and Werner M. Kistler. "Mathematical formulations of Hebbian learning." In: *Biological Cybernetics* 87.5-6 (2002), pp. 404–415. ISSN: 03401200. DOI: 10.1007/s00422-002-0353-y.

[11] Coppelia Robotics GmbH. *V-REP Virtual Robot Experimentation Platform*. URL: http://www.coppeliarobotics.com/. (accessed: 28.08.2018).

[12] H. Hagras et al. "Evolving spiking neural network controllers for autonomous robots." In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004* January (2004), 4620–4626 Vol.5. ISSN: 1050-4729. DOI: 10.1109/ROBOT.2004.1302446. URL: http://ieeexplore.ieee.org/document/1302446/.

[13] Donald Olding Hebb. *Organization of Behavior*. Wiley & Sons, 1949. ISBN: 9780805843002.

[14] Suzana Herculano-Houzel. "The human brain in numbers: a linearly scaled-up primate brain." In: *Frontiers in Human Neuroscience* 3.November (2009), pp. 1–11. ISSN: 16625161. DOI: 10.3389/neuro.09.031.2009. URL: http://journal.frontiersin.org/article/10.3389/neuro.09.031.2009/abstract.

[15] J. H. Holland. "Genetic algorithms." In: *Scholarpedia* 7.12 (2012). revision #128222, p. 1482. DOI: 10.4249/scholarpedia.1482.

[16] Eugene M. Izhikevich. "Solving the distal reward problem through linkage of STDP and dopamine signaling." In: *Cerebral Cortex* 17.10 (2007), pp. 2443–2452. ISSN: 10473211. DOI: 10.1093/cercor/bhl152.

[17] Jacques Kaiser et al. "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, SIMPAR 2016* (2017), pp. 127–134. DOI: 10.1109/SIMPAR.2016.7862386.

[18] Alex Krizhevsky. "One weird trick for parallelizing convolutional neural networks." In: (2014). ISSN: 1053-5888. arXiv: 1404.5997. URL: http://arxiv.org/abs/1404.5997.

[19] UNC Vision Lab. *ImageNet Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)*. URL: http://image-net.org/challenges/LSVRC/2017/index. (accessed: 26.08.2018).

[20] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. "Training deep spiking neural networks using backpropagation." In: *Frontiers in Neuroscience* 10.NOV (2016). ISSN: 1662453X. DOI: 10.3389/fnins.2016.00508. arXiv: 1608.08782.

[21] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback." In: *PLoS Computational Biology* 4.10 (2008). ISSN: 1553734X. DOI: 10.1371/journal.pcbi.1000180.

[22] Da Li et al. "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs." In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)* (2016), pp. 477–484. DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.76. URL: http://ieeexplore.ieee.org/document/7723730/.

[23] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128x128 120 dB 15 micro sec latency asynchronous temporal contrast vision sensor." In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576. ISSN: 00189200. DOI: 10.1109/JSSC.2007.914337.

[24] S. Lowel and W. Singer. "Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity." In: *Science* 255 (Jan. 1992), pp. 209–212. DOI: 10.1126/science.1372754.

[25] iniLabs Ltd. *iniLabs*. URL: https://inilabs.com/. (accessed: 26.06.2018).

[26] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models." In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7.

[27] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. ISSN: 00074985. DOI: 10.1007/BF02478259. arXiv: arXiv:1011.1669v3.

[28] Claus Meschede. "Training Neural Networks for Event-Based End-to-End Robot Control." In: (2017), p. 110.

[29] Kristina D. Micheva et al. "Single-synapse analysis of a diverse synapse population: Proteomic imaging methods and markers." In: *Neuron* 68.4 (2010), pp. 639–653. ISSN: 08966273. DOI: 10.1016/j.neuron.2010.09.024. URL: http://dx.doi.org/10.1016/j.neuron.2010.09.024.

[30] Garrick Orchard et al. "Converting static image datasets to spiking neuromorphic datasets using saccades." In: *Frontiers in Neuroscience* 9.NOV (2015), pp. 1–15. ISSN: 1662453X. DOI: 10.3389/fnins.2015.00437. arXiv: 1507.07629.

[31] Alexander Peyser et al. *NEST 2.14.0*. Oct. 2017. DOI: 10.5281/zenodo.882971. URL: https://doi.org/10.5281/zenodo.882971.

[32] Filip Ponulak and Andrzej Kasiński. "Introduction to spiking neural networks : Information processing , learning and applications." In: (2011), pp. 409–433. URL: https://www.ncbi.nlm.nih.gov/pubmed/22237491.

[33] Alexander D. Rast et al. "Behavioral Learning in a Cognitive Neuromorphic Robot: An Integrative Approach." In: *IEEE Transactions on Neural Networks and Learning Systems* May (2018). ISSN: 21622388. DOI: 10.1109/TNNLS.2018.2816518.

[34] Eric Roberts. *Neural Networks*. URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/index.html. (accessed: 12.05.2018).

[35] N. A. Schmajuk. "Classical conditioning." In: *Scholarpedia* 3.3 (2008). revision #73199, p. 2316. DOI: 10.4249/scholarpedia.2316.

[36] Myung Seok Shim and Peng Li. "Biologically Inspired Reinforcement Learning for Mobile Robot Collision Avoidance." In: (2017), pp. 3098–3105. DOI: 10.1109/IJCNN.2017.7966242.

[37] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: (2014), pp. 1–14. ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.005. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[38] Thomas Splettstoesser. *Schematic of a synapse*. URL: https://commons.wikimedia.org/w/index.php?curid=41349083. (accessed: 07.07.2018).

[39] J. E. R. Staddon and Y. Niv. "Operant conditioning." In: *Scholarpedia* 3.9 (2008). revision #91609, p. 2318. DOI: 10.4249/scholarpedia.2318.

[40] Guangzhi Tang and Konstantinos P. Michmizos. "Gridbot: An autonomous robot controlled by a Spiking Neural Network mimicking the brain's navigational system." In: July (2018). DOI: 10.1145/3229884.3229888. arXiv: 1807.02155. URL: http://arxiv.org/abs/1807.02155%7B%5C%%7D0Ahttp://dx.doi.org/10.1145/3229884.3229888.

[41] Simon J. Thorpe and Michel Imbert. "Biological constraints on connectionist modelling." In: *Connectionism in perspective* July 2016 (1989), pp. 1–36. DOI: 10.1.1.96.6484. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.6484%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[42] J. M. Valeton. "Photoreceptor light adaptation models: An evaluation." In: *Vision Research* (1983). ISSN: 00426989. DOI: 10.1016/0042-6989(83)90168-2.

[43] Jilles Vreeken. "Spiking neural networks, an introduction." In: *Institute for Information and Computing Sciences, ...* March 2003 (2002), pp. 1–5. URL: http://www.cs.uu.nl/groups/ADA/pubs/2002/spiking%7B%5C_%7Dneural%7B%5C_%7Dnetworks%7B%5C_%7Dan%7B%5C_%7Dintroduction-vreeken.pdf.

[44] Xiuqing Wang et al. "The wall-following controller for the mobile robot using spiking neurons." In: *2009 International Conference on Artificial Intelligence and Computational Intelligence, AICI 2009* 1 (2009), pp. 194–199. DOI: 10.1109/AICI.2009.448.