TECHNISCHE UNIVERSITÄT MÜNCHEN

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics: Games Engineering

Comparison of Spiking Neural Network Reinforcement Learning Methods for Autonomous Locomotion of a Snake-Like Robot

Dmitry Kochikiyan

TECHNISCHE UNIVERSITÄT MÜNCHEN

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics: Games Engineering

Comparison of Spiking Neural Network Reinforcement Learning Methods for Autonomous Locomotion of a Snake-Like Robot

Vergleich der Reinforcement-Learning-Methoden in gepulste neuronale Netze für autonome Fortbewegung von Schlangenroboter

Author:	Dmitry Kochikiyan
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Zhenshan Bing, M.Sc.
Submission Date:	16.01.2019

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

Munich, 16.01.2019 Dmitry Kochikiyan

Abstract

In the past decades there was a great advancement in the development of second generation of Artificial Neural Networks (ANN) which, coupled with ever-growing hardware performance, moved them from research phase to mainstream use. However, a major obstacle on the way of further expansion of ANN applications is their significant power consumption. This is due to high numbers of neurons, which means high amount of computation, needed for problem solving in such areas as autonomous robots, vehicles and mobile devices. And autonomous devices have a limited energy resource. A possible solution to the power draw could be ANNs of third generation called Spiking Neural Networks. They much more closely imitate biological neurons, but therefore differ significantly from conventional ANNs. Instead of processing information continuously they do so in discrete spikes imitating biological synaptic connections between neurons, which enables incorporation of spatial-temporal information. One one hand it makes them more efficient, but on the other hand it makes it difficult to apply learning algorithms developed for 2nd generation ANNs. This is the problem which current ongoing research is trying to solve.

Contents

1 Introduction

A great progress in autonomous robotics was achieved in the last decades. The leading causes for this are ever-improving hardware, sensors price reduction and development of new, more efficient Artificial Neural Networks (ANN). Usage of ANNs — computational systems based on biological neural circuits constituting a brain — becomes more widespread thanks to their ability to solve such complex problems as autonomous driving or facial recognition. Advancements in hardware and training methods enabled ANNs to process huge volumes of data much more efficiently than conventional algorithms. In cases when training data is not available, e.g. movement control, reinforcement learning can be used to train networks. In past few years ANNs have been trained to successfully manage robot control or play video games [2], [25], [17]. Still, training as well as executing complex neural networks consumes a lot of computational resources and can not always be done in real time. Problems such as autonomous vehicle driving are very resource-consuming — car sensors produce gigabytes of data every second which then require thousands of Watts to process. In comparison human brain consumes about 20W [9]. One way to reduce energy cost and increase performance is by designing application-specific or ANN-specific hardware. Another way is developing of artificial neural networks that more closely imitate biological neural networks. Commonly used today Convolutional Neural Networks (CNN) differ significantly from biological neural circuit. One of the reasons is that biological brains do not process information discretely as CNNs do. Instead data is processed and transmitted in impulses also called spikes, asynchronously and in continuous time. Artificial neural networks based on this concept called Spiking Neural Networks [27]. Spike generation in SNN is determined by differential equations representing multiple biological processes, such as membrane potential of a neuron cell. When a neuron potential reaches a certain threshold, it produces spike, and its potential resets. SNNs belong to a so called third generation of neural networks and while they promise improvements in terms of efficiency, they still are in active research phase [20]. In this work, two different Reward-Modulated Spike-Time-Dependent-Plasticity learning schemes were implemented and used to train a snake-like robot navigating a wall-enclosed environment scenario. The environment was developed in Virtual Robot Experimentation Platform (V-REP) simulator [31]. Subsequently, the performance and ability to adapt to changing world of trained networks was verified.

2 Theoretical Background

There are many different methods of problem solving, ANNs stand out by relying on mathematical models mimicking to a different degree of accuracy biological neural circuits, instead of precise algorithmic approaches. Although now ANNs are widely used in practical applications, their development began as a theoretical research to better our understanding of neurological structures and learning mechanisms. Despite all the efforts, insight in functioning of a brain is still far from complete, yet even the progress made has greatly expanded the realm of possible in informational technologies. Today ANNs enable solving complicated tasks, e.g. image recognition and categorisation, and in near future solving even more complex problems, such as control of complicated robotic systems, will become possible.

2.1 Biological Background

Modern understanding of nervous system stems from work of Spanish neuroscientist and pathologist Santiago Ramón y Cajal. In 1888 he published his discovery that nervous system consists of discrete neuron cells [12]. And even though more than 100 years have passed there is still huge space for research in this field — just recently, in August 2018, a new neuron type, Rosehip Neuron, was discovered [7]. For all the diversity of neuron types and their structural complexity, in essence they all are simple information processing units. They receive information as electrical pulses, process it and output pulses of their own. By aggregating in gigantic networks they create complex dynamic systems necessary to interact with the environment. The size of such networks varies greatly from species to species, but even small insects like Drosophila melanogaster have connectome comprised of about 140 000 neurons [3]. Neuron cell is made of a cell body (or soma), dendrites and an axon. Dendrites are short branched extensions of neuron that receive the electrochemical signals from other neural cells and pass them to the cell body. Axons are long and thin, they conduct electrochemical impulses known as action potentials from the soma to other neurons. Neurons can have many dendrites but a maximum of one axon, which can connect to several other neural cells. Signal exchange between axon and dendrite is enabled by synapses — structures on their ends, capable of transferring signal either electrochemically releasing neurotransmitters (chemical synapse) or through electric current (electrical synapse),

chemical synapse is the prevalent type. Synapses also divide in excitatory and inhibitory, meaning they may either excite or inhibit the postsynaptic neuron. Therefore synapses are not simple signal transmitters, moreover they play a role in memory formation. Signals received by the dendrites change the voltage gradient of cell membrane. If the change reaches a certain threshold, it generates electrochemical impulse called an action potential which travels along the axon to the synapses. After an action potential has occurred refractory period ensues, during which neuron can't send out pulses in reaction to stimuli.

2.2 Generations of ANNs

The first step to artificial neural networks was made in 1943 by a neurophysiologist Warren McCulloch and a mathematician Walter Pitts [26]. They wrote a paper proposing a computational model for neural networks based on mathematics and algorithms called threshold logic. These first generation neural networks were capable of modeling basic OR/AND/NOT functions by summing binary inputs and outputting a 1 if the sum exceeds a threshold, and 0 otherwise. The second generation of ANNs came when neuron model was extended by a non-linear continuous activation function. The function was applied to the sum of weighted inputs, and produced a continuous set of output values. Majority of ANNs applied today are of 2nd generation and are trained using gradient descent algorithms and backpropagation [14]. They are employed for various tasks such as speech processing and image recognition. The third generation of neural networks introduced spiking neurons as well as plastic synapses. These neural networks model the biological neuron circuits more closely in comparison to first and second generation neurons.

2.3 Classic Reinforcement Learning

In order for a neural network to do anything it must first go through a learning process. There are three general types of learning: supervised, unsupervised and reinforcement. The latter imitates natural learning process of living organisms. Animal brains have evolved to seek rewarding stimuli. Sensory receptors translate stimuli, such as light, sound, smell e.t.c. into action potentials which then are processed by the brain. They are evaluated depending on their potential effect on survival and procreation and result in appropriate reward signals. Similarly, in reinforcement learning the goal is to maximize the reward generated on interaction with environment. To do so, neural network has to learn how its actions affect rewards and which of them are optimal in different situations [34]. The learning process is an iteration of observing the environment,

executing an action and receiving a reward. It can be described as a Markov Decision Process (MDP) — control process which at each time step is in some state s, and the actor may choose any action a that is available in this state. At the next time step process randomly switches to a new state s', and gives a reward $R_a(s, s')$ to the actor. MDP is a 5-tuple $(S; A; P; R; \gamma)$, where S is a finite set of states, A is a finite set of actions, $p^a_{s,s'} \in P$ is the probability that action a in state s at time t will lead to state s' at time $t+1$, $r^a_{s,s'} \in R$ is the immediate reward received after transitioning from state s to state s, due to action a and γ is the discount factor representing the relative value of current rewards to future rewards. The goal of MDP is to find an optimal "policy" *pi* for the actor — an optimal state-action mapping that maximizes actor's reward. In order to be able to maximize reward a value function is required that predicts future reward resulting from sequence of actions.

$$v_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s] \tag{2.1}$$

Due to nature of reinforcement learning problems, rewards are usually spatially and temporally distributed, so a look-ahead is required in order to be able to effectively maximize the reward. This requires the actor states to incorporate information on all possible future rewards, which is impossible for many problems. In order to restrict the amount of future rewards that are calculated, the discount factor γ is used. Reformulating of Eq. 2.3 in recursive form produces *Bellman Expectation Equation*

$$v_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s')|s] \tag{2.2}$$

For which, the optimality equation is formulated as follows:

$$v_{\pi^*}(s) = max_{a \in A}(r^a_{s,s'} + \gamma \Sigma_{s' \in S} p^a_{s,s'}, v_{\pi^*}(s')) \tag{2.3}$$

Bellman equations show the duality of reinforcement learning problems, where actor has to evaluate future action based on policy π and has to optimize its future actions, hence find the optimal policy π^*. This two problems are compounded by the fact, that environments can be incomplete and so the state transition probabilities are not known. The optimality equation (2.3) cannot be solved in closed form, but can be approximated with iterative methods such as *Value Iteration* or *Policy Iteration*. The aforementioned methods need to know the transition probabilities in advance In order to predict environment's future development, models are constructed. Other methods, like $SARSA$ and $Deep\ Q-Learning$ do not require exact model and can learn about

the environment in the process. As seen in this chapter, classical reinforcement learning operates in discrete realm of state-action pairs and time.

2.4 Spiking Neural Networks

The spiking neural network, also called the 3rd generation brings a wider biological plausibility with itself. It expands not only neuron models but also the synapses connecting them, adding dynamics and properties closely modeled on real biological processes. Consequently, the concept of time is adapted as part of the model. Neurons no longer fire continuously but rather discharge (spike) when their membrane potential exceeds a certain boundary. In theory, spiking neural networks have been shown to have greater computational potential then the current ANNs but it comes with certain drawbacks: Gradient Descent does not directly work on non-differentiable spike trains, although some adaptations were propsed by [35], [24].

2.4.1 Hodgkin–Huxley model

A mathematical neuron model was propsed by Hodgkin and Huxley in [16] described the propagation of neuron's action potentials with a set of four nonlinear differential equations that took many biological properties into account simultaneously making this model extremely computationally intensive. The equations describe the response of a neuron to external stimuli. The membrane potential is described by:

$$C_m \frac{dV}{dt} = I_{ext} - \overline{g}_{Na} m^3 h (V - E_{Na}) - \overline{g}_K n^4 (V - E_K) - \overline{g}_L (V - E_L) \tag{2.4}$$

where I_{ext} is the simulation current per unit area; V is the membrane potential; Cm is the capacitance per area of the membrane; E_{Na}, E_k and E_L is the maximum conductance of voltage-gated channel sodium and potassium channel and leak channel; E_{Na} and E_k represent the equilibrium potential for sodium and potassium ions, and E_L is the reverse potential for the leak conductance; m, n, h are the gating variables which regulate the conductance's of ionic channels [10]. Gate transition equation is:

$$\frac{dp_0}{dt} = \alpha.(1 - p_0) - \beta.p_0 \tag{2.5}$$

Due to its complexity this model is extremely slow when computing the state of a population of neurons. This fact led to proposals of many other neuron models that focus on parts of the Hodkin-Huxley model, most common of which is the Leaky Integrate-And-Fire neuron model.

2.4.2 Leaky Integrate-and-Fire Neuron Model

Integrate and Fire and it's variant, Leaky Integrate and Fire neuron is widest used spiking neuron model proposed in [32]. The state of neuron is determined by its voltage V, which models real neuron's membrane potential with $V = 0$ being the idle state. Each time a neuron receives an impulse, its membrane potential increases:

$$V(t_i) = V(t_i) + h \tag{2.6}$$

where h is a constant value. The integrate and fire neuron would constantly hold its charge until it would reach given threshold value V_0, when it will emit a single spike (discharge) and return to the resting state. LIF neuron does not hold its charge without stimuli, it decays according to exponential law:

$$V(t_i + \Delta t) = V(t + i)e^{-\Delta t/\tau} \tag{2.7}$$

where τ is refraction constant. This particularity allows the neuron to "forget" and so adjust to different stimuli distributed in time.

2.4.3 Hebbian Learning

Hebbian learning is based on the theory describing adaptation of synaptic efficacies in the brain. It postulates, that synapses between neurons with highly correlated outputs are strengthened [13] which in essence means that if firing of pre-synaptic neuron leads with high probability to firing of post-synaptic neuron, the connection between them is improved. However, the pre-synaptic neuron influence post-synaptic neuron firing unless it fires before the post-synaptic neuron. That means, that if pre- and post-synaptic neurons fire at the same time, the synapse is not strengthened. Moreover, not only the spike timing but also the spike order is instrumental as experimentally shown in [4]. If pre-synaptic neuron fires before the post-synaptic neuron, the connection is facilitated, while in reverse case the connection is depressed. This is the concept underlying the Spike-Time-Dependent-Plasticity. Learning strategies using STDP are considered to fall under Unsupervised Learning, since no critic, error function or goal is available. Consequently STDP was applied in input categorization and pattern recognition tasks [15]. STDP rule is defined as weight update function dependant on the time difference between pre- and post-synaptic spike

$$\begin{aligned}
\Delta t &= t_{post} - t_{pre} \\
\Delta w_+ &= A_+ \cdot e^{-\frac{\Delta t}{\tau_+}} \quad if \quad \Delta t > 0 \\
\Delta w_- &= A_- \cdot e^{-\frac{\Delta t}{\tau_-}} \quad if \quad \Delta t < 0,
\end{aligned}$$

where A_+,A_- - positive constants scaling the weight changes for facilitation and depression, τ_+, τ_- - positive constants defining the width temporal window, during which STDP is active.

2.4.4 Reward-Modulated STDP

Florian in [11] has shown, that modulation of STDP learning rule with a global reward signal leads to reinforcement learning. He developed a simple reward-modulated STDP rule and applied them to solve XOR-Problem with a network of integrate-and-fire neurons. Based on this, Potjans et al. [28] developed a synapse model that modified the STDP rule (Eq. 2.4.3). The new rule does not update the weights straight away but collects them into an eligibility trace:

$$\begin{aligned}\Delta w &= c(n-b)\\ \Delta c &= -\frac{c}{\tau_c} + STDP(\Delta t)\delta(t - s_{pre/post})C_1\\ \Delta n &= -\frac{n}{\tau_n} + \frac{\delta(t-s_n)}{\tau_n}C_2\end{aligned}$$

where c is the eligibility trace, n is the neuromodulator concentration, b is neuromodulator baseline, $s_{pre/post}$ - timing of pre- or post-synaptic spike, s_n - timing of neuromodulatory spike, C_1, C_2 - constant coefficients, δ - Dirac delta function and τ_c, τ_n are eligibility and neuromodulator concentration time constants. $STDP(\Delta t)$ is defined in Eq. 2.4.3. In this model, the neuromodulator concentration is always present, which is reflected by the neuromodulator baseline constant b. Observation of base firing rates in neurons led to conclusion that global reward can be expressed as neuromodulator concentration deviation from its mean value $\bar{b}$.

2.4.5 Additive and Multiplicative Reward-Modulated STDP

Reward-modulated STDP rule introduced in previous section was used by Shim and Peng [33] to teach a car robot to drive towards goal avoiding collisions. In their work, additive and multiplicative RSTDP learning schemes were used. Additive RSTDP learning rule is defined as

$$\frac{dw_{ij}(t)}{dt} = \gamma r(t) z_{ij}(t),$$

where γ is reward factor, $r(t)$ is reward signal and $z_{ij}(t)$ is the eligibility trace. This learning rule is exactly the rule introduced by Florian in [11]. Shim and Peng suggest a modification to this rule in form of multiplicative RSTDP

$$\frac{dw_{ij}(t)}{dt} = \gamma w_{ij}(t) r(t) z_{ij}(t).$$

This rule incorporates current weight $w_{ij}(t)$ in weight change calculation. This change has been shown to produce stable uni-modal weight distributions. In this work both of this learning schemes are implemented and tested.

2.4.6 Classical Reinforcement Learning and SNN

Classical reinforcement learning as described in Section 2.3 relies on existence of discrete states, actions and time. Spiking neural networks however operate on asynchronous spike events distributed in continuous time. This makes gradient descent and temporal difference learning methods unusable without in SNNs without modifications. Such modifications lead to complex, computationally intensive systems. It constitutes an active research topic. In works by Potjans et al. [35] and Fremaux et al. [24] an actor-critic approach was used and Temporal Difference was adapted. However, the tasks solved by this networks are not easily transferable to the context of this work. As such, in this work the focus is on the aforementioned additive and multiplicative RSTDP.

3 Methodology

In this chapter the tools and methodology of proposed model are summarized and the critical parts are detailed. In the Section 3.1 the simulation environment VREP its settings, DVS and ACM-R5 models are introduced. In Section 3.2 the simulation scenarios are presented. In Section 3.3 the NEST Simulator is introduced, transition from DVS data from DVS sensor to network input is discussed and SNN architecture is outlined. In the Section3.4 the interactions between controller and environment are detailed.

3.1 Simulation Environment

3.1.1 V-REP

V-REP is a robot simulation environment with an integrated development environment developed by Coppelia Robotics. It is based on a distributed control architecture that allows a user to control each object separately via variety of APIs or a user-specific solution[30]. It is free for education purposes. The simulation is run using physics engine Bullet 2.78, with a time step of 50 ms. V-REP also provides many built-in models, such as HiBot's ACM-R5 snake-like robot model, DVS128 model, proximity sensor models used in this work. It also allows to import custom objects and models. For this work, custom shaped meshes were created and textured using OpenSCAD IDE to act as walls. The objects in V-REP can be attached child scripts - embedded simulation scripts, written in Lua that provide an ability to specify dynamic and static behaviours. This scripts are executed every simulation step.

3.1.2 Robot Operating System

The Robot Operating System is a framework for developing robot software [29]. V-REP provides a ROS plugin, that allows user programs to communicate with simulation environment over ROS topics. ROS publishers and subscribers must be created both in simulation environment and external controller as well as roscore must be loaded on the host machine. In order to send data trough a ROS topic, it must be published and to receive it, user must subscribe to the topic and provide a callback function to

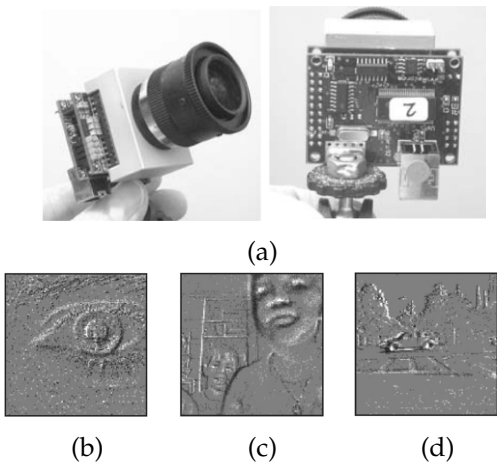


Figure 3.1: (a) - DVS sensor, (b) - Eye shot with DVS sensor, 7500 events in 300 ms, (c) - Faces, 8000 events in 26 ms, (d) - Driving scene, 4000 events in 29 ms as presented by Lichtsteiner et al in [19]

handle the data processing. On the V-REP side, publishers and subscribers are defined in embedded scripts, like ACMR child script in this work. On the user side, python ROS bindings rospy was used. In this work ROS Melodic Morenia was used to facilitate communication between V-REP and python controller.

3.1.3 Dynamic Vision Sensor

Classic video sensors are operating on a frame-by-frame basis, recording fixed-size images with a fixed frame-rate. This comes with storage overhead dependant on the frame-rate, required by the application. Also, in order to use images, recorded with classic video sensor in an SNN application, conversion method is required to adapt pixel data to spike trains. In [19] Lichtsteiner et al. introduced a Dynamic Vision Sensor, that abandons frame-based vision in favour of biologically inspired vision process. Each pixel of this sensor encodes the change in local relative intensity, asynchronously delivering information about its status change on address-event bus. This results in reduction of data redundancy while preserving precise event time information, since each pixel is temporally independent and only produces an event if its status changed.

In Fig. 3.1 the DVS hardware and DVS data frames are depicted. Table 3.1 lists devise properties. In this work a V-REP model of DVS128 sensor provided by iniLabs [1] was used. This model does not exactly match the real hardware as proposed in [19] since simulating DVS with aforementioned specifications is difficult. Instead, the sensor is simulated by comparing subsequent frames of a video stream, and then retaining only

Table 3.1: Specifications of a Dynamic Vision Sensor by Lichtsteiner et al. [19]

Dynamic range:	120dB
Contrast threshold mismatch:	2.1%
Pixel array size:	128×128 pixels of 40μm × 40μm
Photoreceptor bandwidth:	$\geq$3 kHz
Event saturation rate:	1 million events per second
Power consumption:	23mW
Minimum latency:	15μs

pixels, where the absolute difference is greater then a threshold value as done in [18]. However, this method compounded with simulating in time steps undermines the key features of event independence and microsecond temporal resolution of a real DVS sensor. The pixel events are now emitted every simulation time step with the same time stamp and as such the simulated DVS data differs from real data rather significantly as shown by Kaiser et al. [18] in 3.2. They proceed to argue, that a robust SNN would be able to cope with such inputs.

The DVS model uses AER Protocol to transmit its data. AER Protocol specifies a simple format for event messages: (x, y)-address, ON/OFF state, and a time stamp. The model provided by iniLabs uses 14 bits for address, 1 bit for state, 16 bits for time stamp and leaves 1 bit unused. In this work, in order to simplify simulation, only address information was used. Also, only the walls were made detectable, while the floor and other objects were made invisible for the DVS sensor. The typical simulation DVS frame is shown in Fig. 3.3.

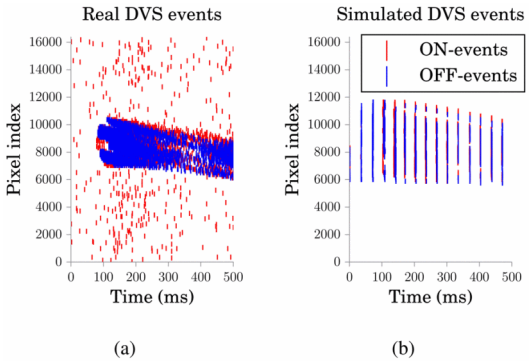


Figure 3.2: (a) - real DVS data, (b) - simulated DVS data [18]

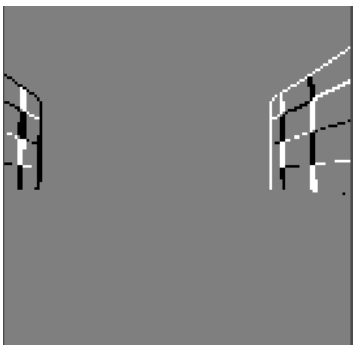


Figure 3.3: Simulated DVS frame

3.1.4 HiBot ACM-R5 Snake Robot

ACM-R5 is an amphibious snake-like robot developed by HiBot Corporation [8]. For this robot, a V-REP model is available with default V-REP installation (Fig. 3.4). It is composed of 9 segments connected with 2 revolving joints, that turn round y and z axis, allowing robot to operate in water. In this work, the snake robot is only used on a horizontal plane. The joints turn angles are controlled by a child script associated with ACM-R5 in V-REP. In a work by Bing et al.[5] a slithering gate movement model was developed which is used here. The following 2D extended gait equations

$$\begin{aligned} \alpha(n,\ t) &= C + P \cdot A \cdot \sin(\Omega \cdot n + \omega \cdot t) \\ P &= (\frac{n}{N} \cdot z + y) \in [0,\ 1], \quad \forall n \in [0,\ N], \end{aligned} \tag{1}$$

where $\alpha(n,t)$ is the joint angle of the nth joint at time t, C - body shape offset, A - wave amplitude, P - linear reduction coefficient, Ω - spatial frequency, ω - time frequency, N - robot module amount, n - module subscript, y, z - linear coefficients govern the motion of the snake-like robot. Slithering gate induces the head module to move with substantial amplitude from side to side while staying directed forward. This can distort DVS image and make learning unstable. In order to minimize that effect, linear reduction coefficient P modifies wave amplitude A such that it is minimal at the head module and linearly increases up to the tail module. According to Bing et al. [5] linear reduction modifies shape of the robot from cylinder to frustum and reduces the head swing motion by 70%. The body shape offset C is a control parameter, added as a bias to 3.1.4 control snake turn radius. It is defined as

$$C = \frac{l_0 \cdot \Sigma_{k=1}^{N} cos(A \cdot \Sigma_{p=1}^{k} sin(\Omega \cdot p))}{r}$$

where l_0 is length of a single snake module and r is the turn radius.

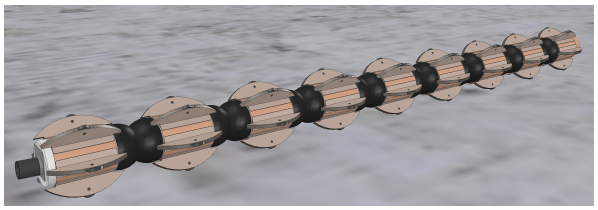


Figure 3.4: ACM-R5 model in V-REP with a DVS

3.2 Simulation Scenarios

In order to generate SNN training data and to subsequently test the trained networks two V-REP scenarios were developed. The ACM-R5 model was modified to include a DVS128 sensor as well as 5 proximity sensors mounted on the first (head) module.

3.2.1 Modified ACM-R5 Model

ACM-R5 model was extended to include DVS sensor and 5 ray-type proximity sensors in its head module. The DVS sensor was used to generate spiking neural network input while the proximity sensors were used to compute reward signal (as explained in Section 3.4.1). On Fig. 3.5 the configuration of proximity sensors with relation rectangle, bootstrapped by DVS near and far clipping plane as well as perspective angle α is shown. Proximity sensors are arrayed in such a way, that the angle between left and right sensors match the perspective angle α. The angles between sensors β are all equal.

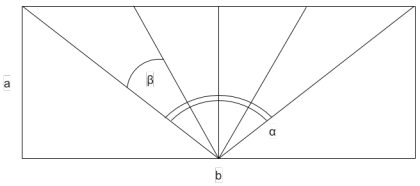


Figure 3.5: A DVS sensor volume with proximity sensors cutting it into 4 sectors

In the Table 3.2 the measurements are listed.

Table 3.2: Proximity sensor position properties

Outter left proximity sensor detection length:	$1.5m$
Outter right proximity sensor detection length:	$1.5m$
Inner left proximity sensor detection length:	$1.1m$
Inner right proximity sensor detection length:	$1.1m$
Perspective angle α:	100°
Angle between proximity sensors β:	25°
Near to far clipping plane distance a:	$1m$
Frame length b:	$1m$

3.2.2 Training Scenario

First V-REP scenario is for SNN training. It consists of 3 different stages with stage 1 and 3 split into 2 phases. Table 3.3 provides measurements:

Table 3.3: Training scenario 1 wall measurements

Wall-to-wall distance:	$1.5m$
Turning radius for outer walls:	$10m$
Turning radius for inner walls:	$5m$
Walls height:	$0.5m$

Robot starts in stage 1, which is realised in 2 phases. In phase 1 it is positioned on the left-hand side of the U-Turn and learns the right U-Turn. Once completed in phase 2 it is set on the right-hand start of the U-Turn to learn the left U-Turn. After successfully completing U-Turn in both directions, it is set on a straight lane (stage 2). Finally, it is moved to complete left turn and subsequently right turn in the stage 3 phase 1 and phase 2 respectively. The training routine is specified in ACMR child script and makes use of V-REP distance calculation module. Each stage uses dummy objects to calculate the minimal distance between the ACM-R5 robot and itself. If the distance becomes smaller then 2 meters, the current phase is completed and robot is moved to the next phase or stage. In Fig. 3.6 the starting points of each stage and phase are labelled. Once all stages are completed, the training is stopped by publishing completion message to '/simStopped' topic.

3.2.3 Testing Scenario

The testing scenario is used to verify the networks trained in the training scenario. It is similar in form to scenario 1 proposed by Bing et al. [6]. In order to test the ability

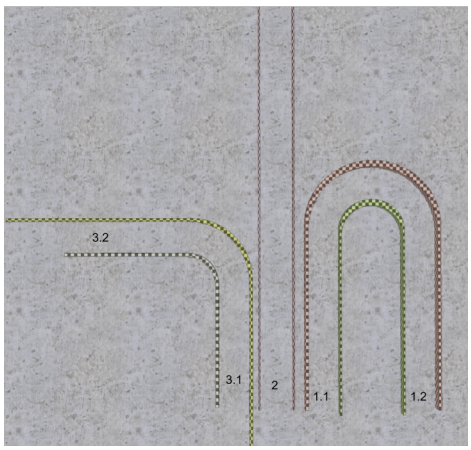


Figure 3.6: V-REP Scenario 1 with stage starting points labelled

of trained networks to adapt to a changed world, wall to wall distances, wall heights and textures are modified. In Fig. 3.7 the different wall-to-wall distance segments are enumerated. The snake robot starts in the segment 1 and moves through subsequent turns.

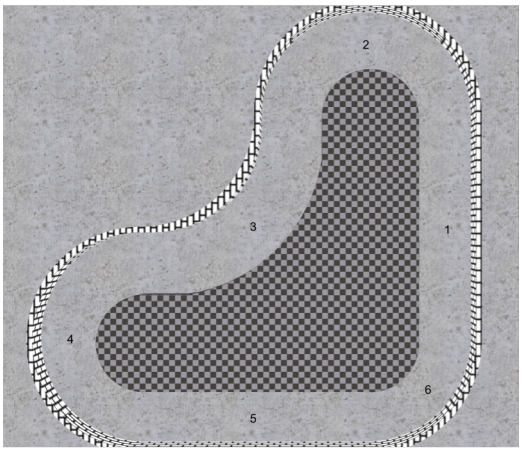


Figure 3.7: V-REP Scenario 2

Test run is completed, once the robot reaches starting position. For that same approach as in the training scenario is used - once the segment 6 is passed and head module of ACM-R5 is closer to the dummy object then 0.3 meters, simulation

completion state is published to '/simStopped' topic. Table 3.4 provides per-segment wall-to-wall distances and wall heights. Additionally, outer wall's texture was changed to a brick wall.

Table 3.4: Training scenario 2 wall-to-wall distances for segments 1-6 and wall heights

Inner wall height:	$0.35m$
Outer wall height:	$0.55m$
Segment 1:	$1.4m$
Segment 2:	$1.7m$
Segment 3:	$1.8m$
Segment 4:	$1.6m$
Segment 5,6:	$1.4m$

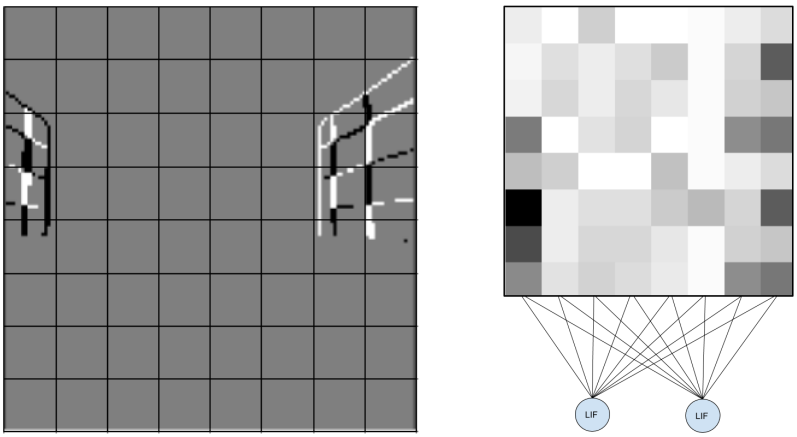
3.3 Neural Network

3.3.1 NEST Simulator

The spiking neural networks for this work were developed using NEST Simulator developed by Nest Initiative. NEST provides full-fledged framework for spiking neural network creation and simulation. In this work NEST 2.16 was used to define spiking neural network and simulate it.

3.3.2 Transforming DVS output into SNN input

DVS sensor model has resolution of 128×128 pixels. As shown in Section 3.1.3 due to simulation environment and model constraints, the pixels are not temporally independent and the data is transmitted every 50ms with the same time stamp. In order to effectively use it as input for the spiking neural network, the events must be spread in time. Bing et al. in [6] proposed a method of adapting the raw DVS output. For the purposes of this work this method was slightly modified. The DVS data is scaled down from 128×128 pixles to an 8× frame, where each pixel is fused from original pixels with changed state, regardless of polarity. Subsequently the value of each pixel in the new 8×8 frame is spread in time by clamping it to maximum value of 300 and using it to set the rate of poisson spike generator connected in one-to-one fashion to parrot neurons with static, unchanging synapses. The steps are shown in Fig. 3.8



(a) (b)

Figure 3.8: Scaling of original DVS frame to 8×8 frame
(a) - original simulated frame, (b) - fused 8×8 frame

3.3.3 Spiking Neural Network Architecture

Following spiking neural network architecture based on Kaiser et al. [18] and Bing et al. [6] was trained using 2 different reward signal functions and reward-modulated STDP schemes. It consists of 8×8 Poisson spike generators generating input spikes as discussed in previous section, connected to 8× parrot neurons. The input neurons are subsequently connected in all-to-all fashion with dopamine-modulated STDP synpases to 2 LIF action neurons modelling the "turn left" and "turn right" state. The output spikes of action neurons are used to calculate the turning radius of the snake. Instead of generating the dopaminergic signal by exciting a user-specified pool of neurons and using volume transmitter to deliver it to synapses as proposed by Potjans, Wiebke et al.[28], the neuromodulator concentration n was set directly in each simulation time step. This trade-off in biological plausibility allows for significant simulation simplification while still maintaining the underlying dynamics of dopamine modulation. For additive scheme, connections to the left action neuron $conn_{left}$ received and right action neuron $conn_{right}$ were updated using Eq. 2.4.5, such that the neuromdulator concentration was set to be opposite for left and right connections. For multiplicative scheme, the Eq. 2.4.5 was used. The neurons and synapses parameters can be found in

3.4 Controller

All the parts discussed in previous sections are driven from a PyDev project consisting of training.py, controller.py, environment.py, networkARMSTDP.py, networkMRMSTDP.py and helper files. Following chapters will detail project architecture, V-REP to python controller communication, training and testing flows.

3.4.1 Reward Calculation

In order to train the network a reward signal based on sensory input must be generated. The aim of this work was not only to compare the performance of different learning rules but also to develop a simulation model, that will not depend on any information, that is not directly sensed by the robot. To this purpose, 2 reward functions are proposed and implemented in environment.py. First function uses cubic function of approximation of lane center defined as follows: the distance sensed by left and right proximity sensor was projected on near clipping plane of DVS, center was approximated and then used as variable for the cubic function:

$$d_{left} = p_{left} \cdot cos(50^\circ) \tag{3.1}$$

$$d_{right} = p_{right} \cdot cos(50^\circ) \tag{3.2}$$

$$r = (\frac{d_{left} - d_{right}}{2} - d_{left})^3 \tag{3.3}$$

Second function uses all 5 proximity sensors to calculate reward signal as a cubic function of difference between sum of area of triangles formed by outer left, inner left and forward sensor and outer right, inner right and forward sensor.

$$a_{left} = \frac{1}{2} \cdot p_{left} \cdot p_{innerleft} \cdot cos(25^\circ) + \frac{1}{2} \cdot p_{innerleft} \cdot p_{forward} \cdot cos(25^\circ) \tag{3.4}$$

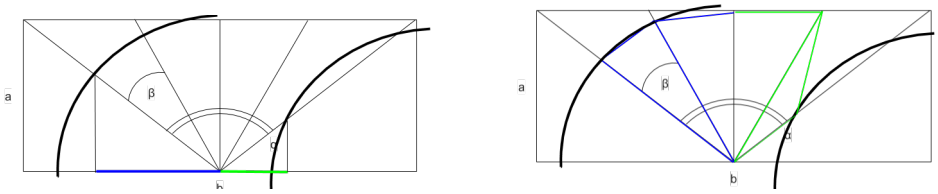
$$a_{right} = \frac{1}{2} \cdot p_{right} \cdot p_{innerright} \cdot cos(25^\circ) + \frac{1}{2} \cdot p_{innerright} \cdot p_{forward} \cdot cos(25^\circ) \tag{3.5}$$

$$r = (a_{left} - a_{right})^3 \tag{3.6}$$

In Fig. 3.9 both functions are depicted. The comparative advantages and drawbacks of both functions are discussed in Chapter 4.

3.4.2 Snake Turning Model

Snake robot is directly controlled by the difference between n_{left} and n_{right} spike count read from left and right action neuron respectively. Justification for that is that robot



(a) (b)

Figure 3.9: Reward function
(a) - center approximation based, (b) - area based

must learn to turn away from neuroactive regions of the input frame, since a more neuroactive region would induce the respective action neuron to fire more then its counterpart. When travelling between 2 walls it results in a the robot striving to stay in the center but it also induces robot to avoid obstacles.

3.4.3 Project Architecture

The project architecture is same as in RSTDP controller developed by Claus Meschede for his Master's Thesis [21] (Fig. 3.10) and is depicted below. In this work, python 3.7 was used instead of python 2.7 as well as ROS Melodic Morenia and NEST 2.16. The simulation was hosted on an Archlinux machine. Archlinux was chosen due to hardware constraints of the host machine. However, this should not provide any difficulties in replication of this work on a common Ubuntu machine. The workflow of a single controller execution is depicted in Fig. 3.11.

3.4.4 V-REP to Controller communication

The communication between V-REP simulation and python controller is handled via ROS messages. Both python environment handler and V-REP ACM-R5 child script define a number of ROS publishers and subscribers. On the V-REP side, proximity sensors data, DVS data, collision flag and simulation completion flag are published, while simulation itself subscribes to environment's handler radius and reset topics. Environment handler subscribes to data topics provided by V-REP simulation and handles the incoming data.

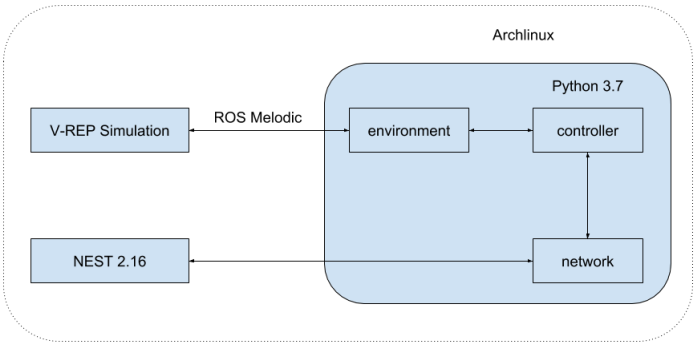


Figure 3.10: Controller architecture used in this work

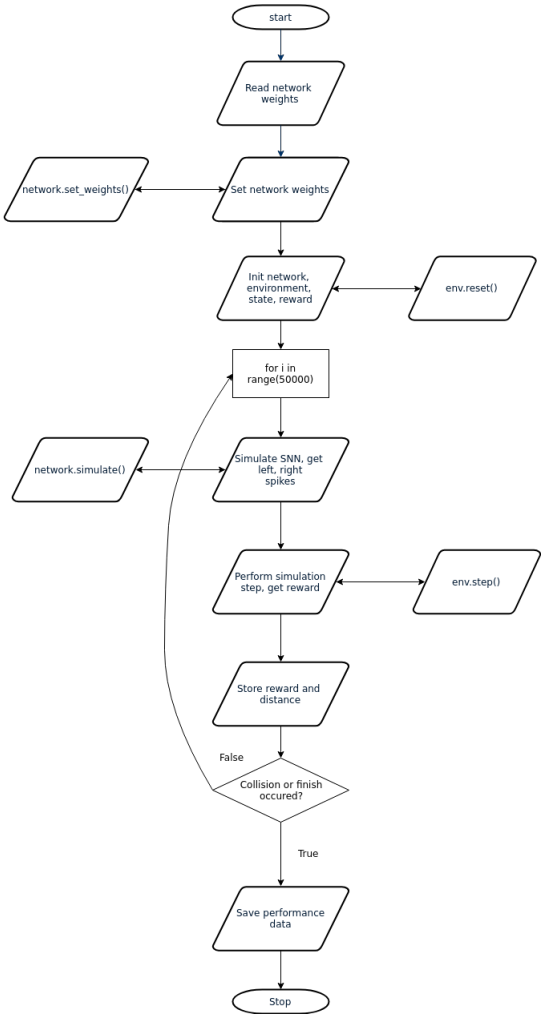


Figure 3.11: Testing flow

4 Discussion

In this chapter the choice of learning schemes is discussed as well as the performance of the neural networks trained accordingly is evaluated. The reward functions are analysed and their impact on training and testing performance is measured.

4.1 Training Performance

As mentioned in Section 2.4.6, the classical reinforcement learning approach based on MDP is not directly applicable to asynchronous spike trains in continuous time. Adapted algorithms relying on actor-critic network architecture and place cells as state space representation have been shown to work well for relatively small state spaces (a 5×5 grid-world, [35], [24], [23]) but they fail for high-dimensional tasks.
Reward-modulated STDP learning rules as discussed in Section 2.4.5 provide biologically inspired reinforcement learning framework based on dopaminergic facilitation or depression of synapses, modifying their plasticity. In this work, additive reward-modulated STDP and multiplicative reward-modulated STDP learning rules were used to train a neural network to control snake-like robot navigating in an environment closed by walls. The training model developed in this paper focuses on an agent that only relies on sensory input, without any critic or model information. As described in Sections 3.3.2 and 3.4.1 the reward function is chosen in such a way, that the signal it produces correlates directly with the DVS input to SNN. With neuromodulator baseline of 0 the networks strives to minimize its reward to 0, which reflects the desired behaviour of staying in the center of an enclosed space.

4.1.1 ARM-STDP with center-approximating reward function

The most simple training model proposed is the ARM-STDP learning rule with center-approximating reward function as specified in (3.1). The reward function has some drawbacks, which can potentially break the training or testing sequences. In particular, if snake moves directly at the wall, the distance approximation would be roughly the same on both sides, leading to reward signal vanishingly small. This will mean, that the snake will travel in a straight line until it hits the wall. Another drawback is that if the snake moves into a turn while being close to the inner turning wall, the reward

signal would be incorrect in a sense, that it would induce the snake to turn away from the inner wall and depending on how narrow the turn is, it will potentially lead to the snake unable to avoid collision in time. The training sequence was completed in 5700 simulation steps with 2 collisions in the beginning of the right U-Turn, at the step 180 and in the middle of the left U-Turn, at the step 2603. In Fig. 4.9 the reward development through time is plotted. The moment of collision can be seen as extreme dips on the plot. In Fig. 4.1 the development of weights during the training is plotted with weights of synapses connected to the left motor on top and to the right motor on the bottom. One can see that the shapes of left and right weights change are mimicking each other. The collision observed in episodes 180 and 2603 can see inducing significant changes, that stabilize afterwards.

In Fig. 4.2 it is noticeable, that the weights reflect a DVS simulation frame, with left and right wall shapes mirrored in weights distribution.

4.1.2 ARM-STDP with triangle area reward function

In order to improve on the drawbacks of the simple center approximation function an area-based function is proposed as specified in Section 3.4.1. While the movement straight at the wall still is unresolved, the slithering gait induces the area changes to produce a slight turning behaviour before impact. The second drawback however is mitigated by the fact, that this reward function uses all 5 proximity sensors to produce a well distributed reward signal. In Fig. 4.9, 4.3, 4.4 the reward during testing, weights evolution and final weight distribution are plotted. The training sequence was complete in 5400 steps with 2 collisions in left and right U-Turn stage at step 487 and 2187.

4.1.3 MRM-STDP with center-approximating reward function

Multiplicative reward modulation behaves differently compared to additive as described in [33]. It produces weight distributions that are more stable under input distortion. Weights also reach their final form quicker. Here, the training results for MRM-STDP with center-approximating and area-based reward function are shown. From the plots, it is obvious that the weights do indeed reach its stable distribution much quicker and the reward signal form reflects that as well. The MRM-STDP with center-approximating function finished training sequence in 4900 steps with 2 collisions at step 210 and 770, while the area-based variant finished in 4800 steps and 1 collision at step 771. (See Fig. 4.9, 4.5, 4.6, 4.7, 4.8)

4.2 Testing Performance

The 4 variants of trained SNNs were tested on adaptiveness in a second V-REP scenario. All 4 conrollers were able to complete test scenario on the first try. Subsequently each controller was run for 50000 steps on a testing scenario, while logging instantaneous reward, distance traveled, each collision as fail and each completed lap as success. Each controller was able to navigate the testing scenario without fail, running on average 1188.4 meters or 27 laps.

4.2.1 Performance Summary

It proved to be difficult to define a sane metric to compare testing performance, since the robot's task was to advance in a wall-enclosed environment without collisions. Each controller was able to complete 50000 simulation steps without encountering a collision. However in training performance, the multiplicative rule shows clear advantages in learning speed. It completes the training sequence on average quicker then additive rule based network as well as reaches stable weight distribution faster. Training data also seems to suggest, that an area-based reward function behaves more stable then center approximation based function.

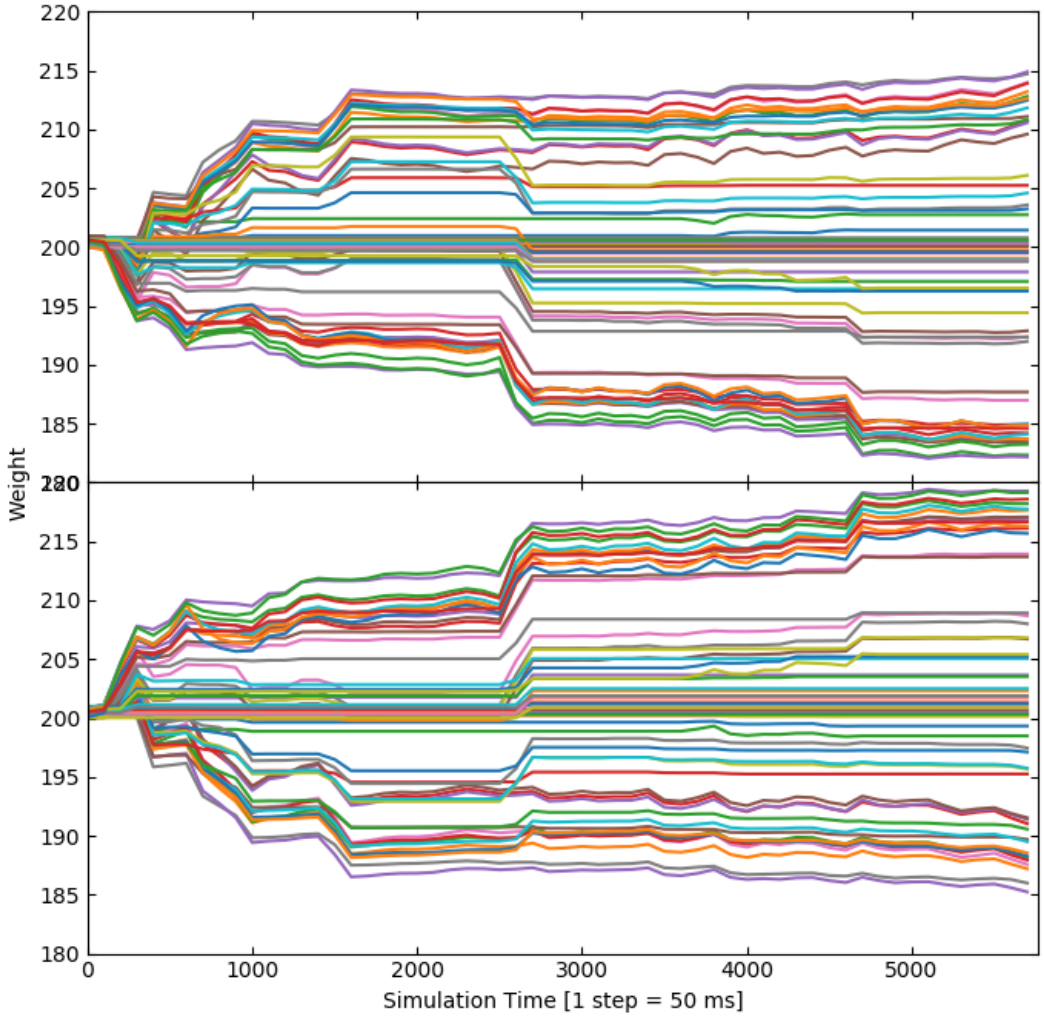


Figure 4.1: Weights development of ARM-STDP with center approximating reward function over simulation time

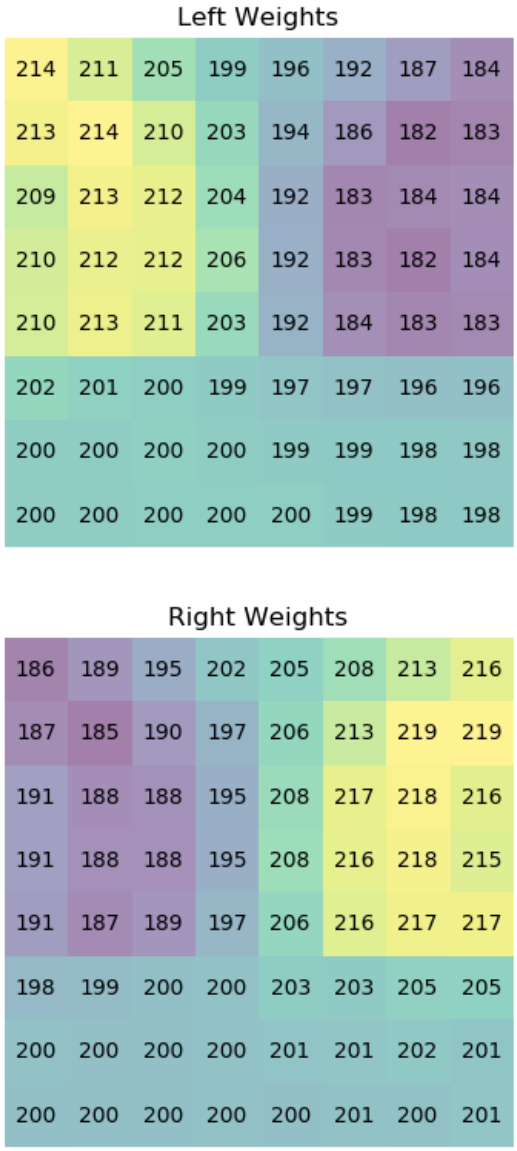


Figure 4.2: Final weight distribution of ARM-STDP with center approximating reward function over simulation time

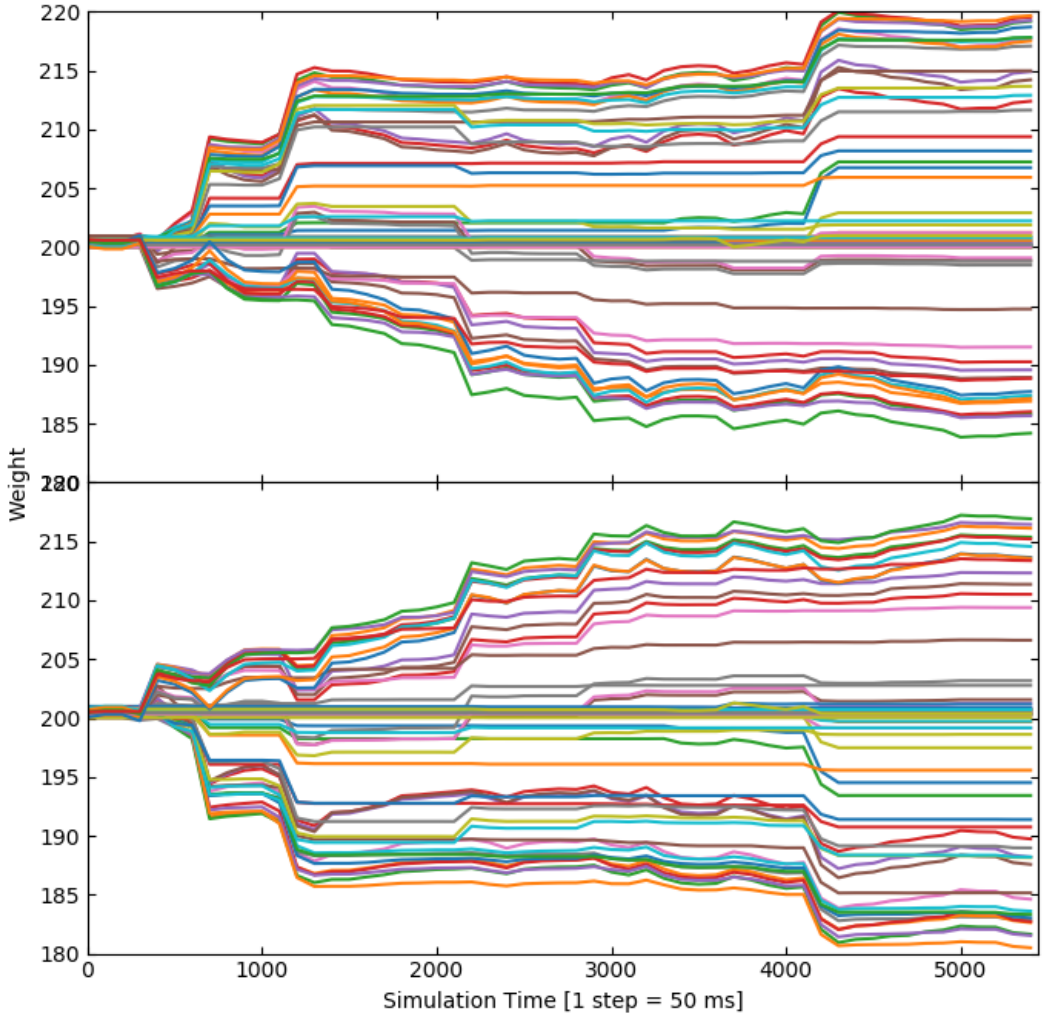


Figure 4.3: Weights development of ARM-STDP with area-based approximating reward function over simulation time

Figure 4.4: Final weight distribution of ARM-STDP with area-based approximating reward function over simulation time

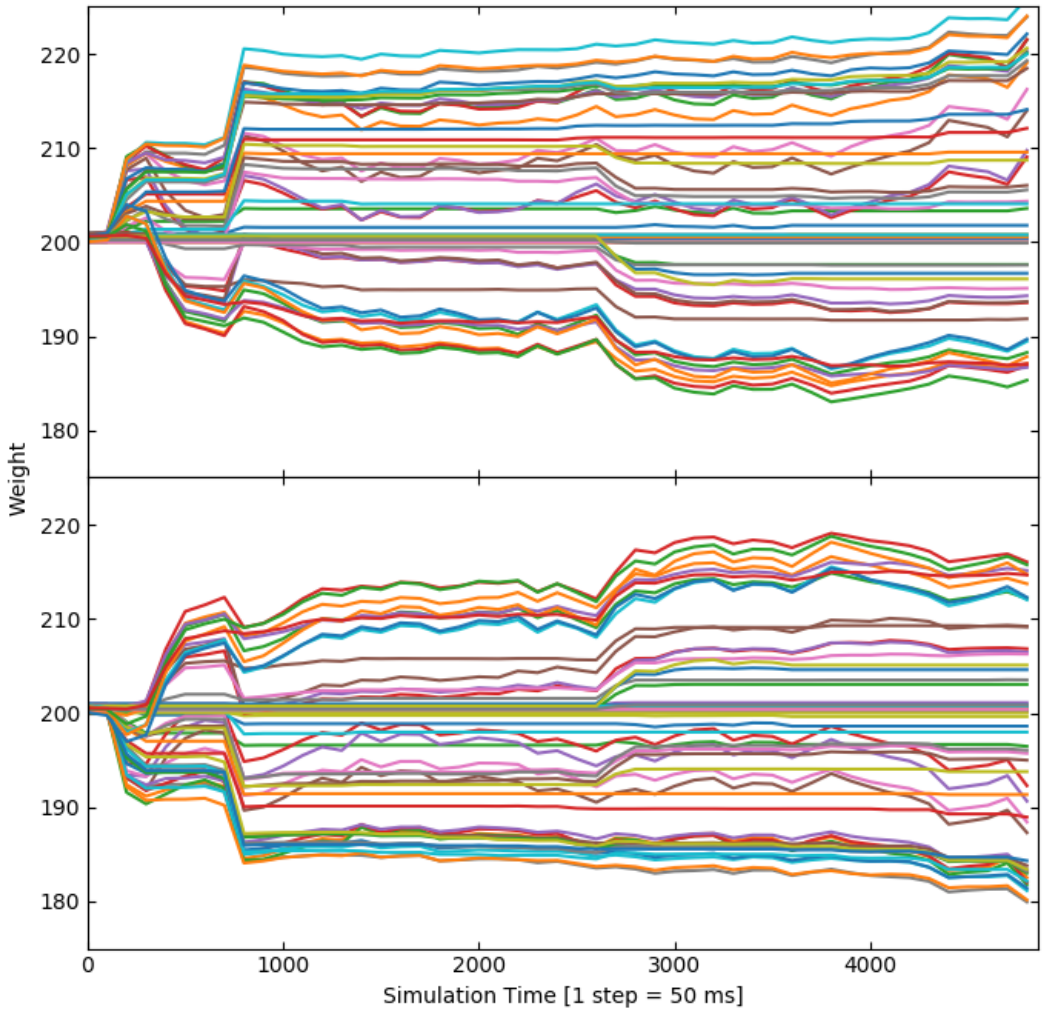


Figure 4.5: Weights development of MRM-STDP with center-approximating approximating reward function over simulation time

Figure 4.6: Final weight distribution of MRM-STDP with center-approximating approximating reward function over simulation time

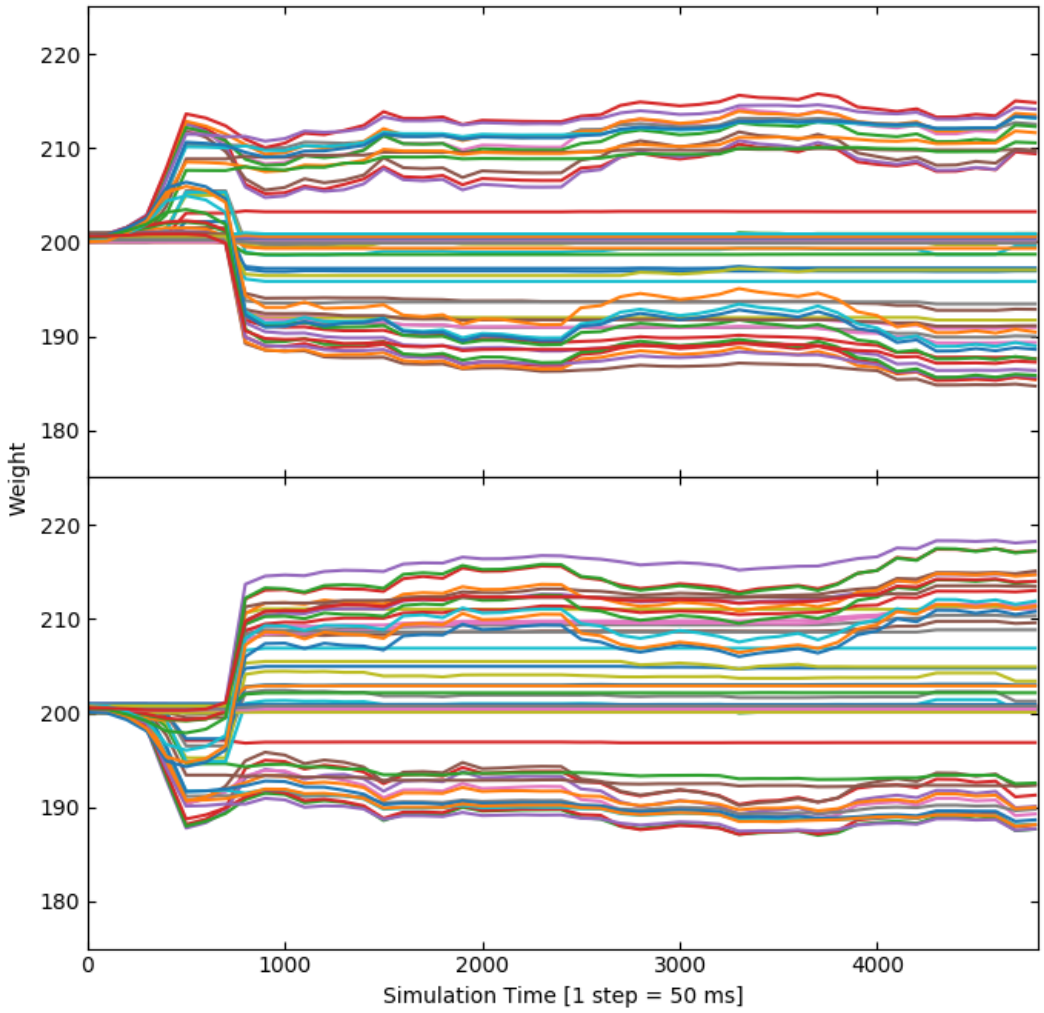


Figure 4.7: Weights development of MRM-STDP with area-based approximating reward function over simulation time

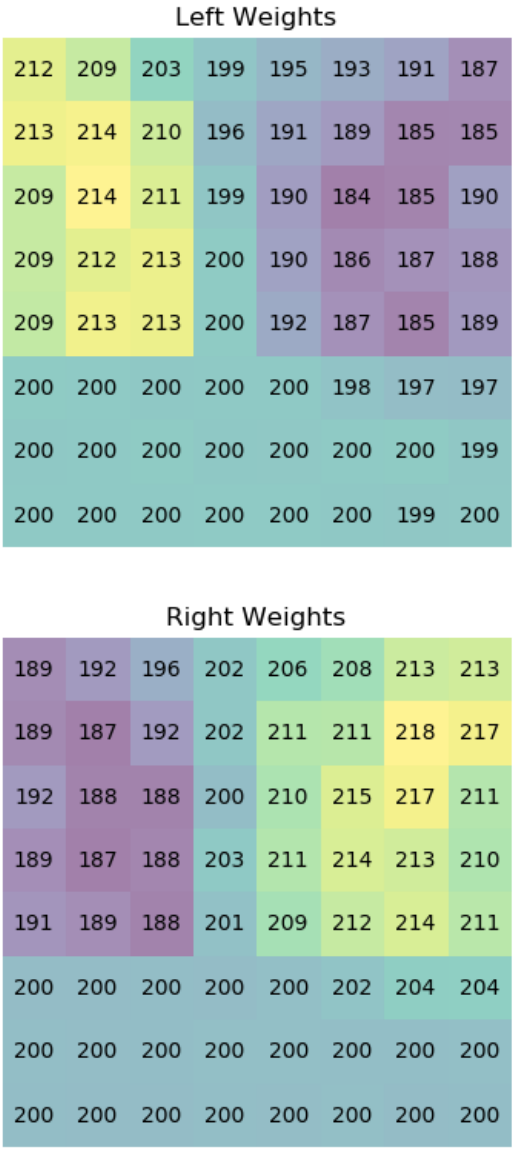
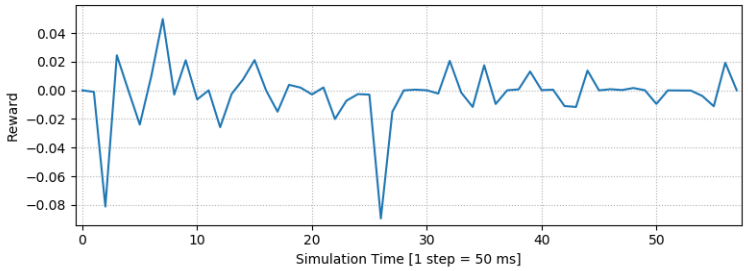
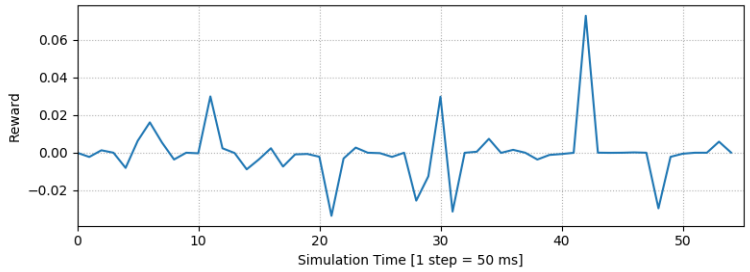


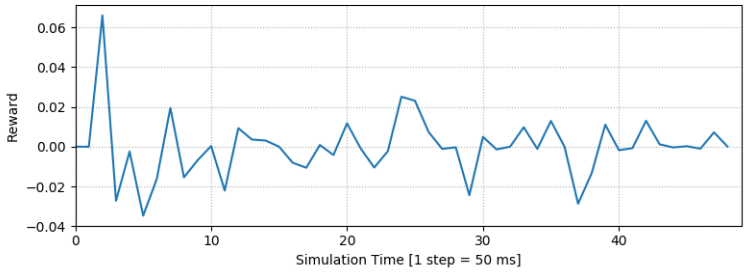
Figure 4.8: Final weight distribution of MRM-STDP with area-based approximating reward function over simulation time



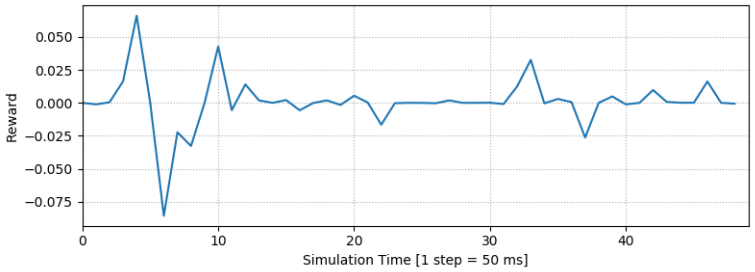
(a)



(b)



(c)



(d)

Figure 4.9: Reward signals: (a) - ARM-STDP center-approximating, (b) - ARM-STDP area-based, (c) - MRM-STDP, center-approximating, (d) - MRM-STDP area-based

5 Conclusion and Outlook

In this work a python controller using a 2-layer SNN was implemented based on controller used in [30] to drive a Pioneer robot. It was altered to allow reinforcement learning and control based on additive and multiplicative reward-modulated STDP rules of a snake-like robot without knowledge of external world's model other then the sensory inputs provided by DVS and proximity sensors. 5 proximity sensors were used to provide reward signal, with 2 different calculation methods. The aim was to mimic biologically plausible scenario of autonomous locomotion. Consequently, the training and testing performance of ARM-STDP and MRM-STDP was reviewed. While there was no clear winner in terms of testing performance, since each configuration was able to run for 50000 simulation steps without colliding with a wall, training data shows that MRM-STDP rule is able to learn marginally faster. Area-based reward function performed stabler then center-approximating function. In conclusion, relatively simple model is able to learn autonomous locomotion task correlating its own sensory inputs and not relying on external reward signal. Additive and multiplicative reward-modulation schemes produce equivalent networks. Further improvements of reward function might improve learning significantly - for example, during experiments, a function using 9 proximity sensors was considered to deliver a interpolated wall silhouette, calculate the corresponding areas and use their difference as variable for cubic reward function. This approach was however left behind due to its complicating the snake model. Another interesting topic for further examination is the adaptation of method developed by Nakano et al. in [22].

List of Figures

List of Tables

Bibliography

[1] iniVation AG. *DVS128*. Jan. 2019, Accesed 14.01.2019. URL: https://inivation.com/support/hardware/dvs128/.

[2] AlphaGo. *AlphaGo*. Jan. 2019, Accesed 14.01.2019. URL: https://deepmind.com/research/alphago/.

[3] A. AP, C. M, C. GM, G. RJ, R. ML, and Y. R. "The Brain Activity Map Project and the Challenge of Functional Connectomics." In: *Neuron* 74.6 (June 2012), pp. 970–974. DOI: 10.1016/j.neuron.2012.06.006.

[4] G.-q. Bi and M.-m. Poo. "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type." In: *Journal of Neuroscience* 18.24 (1998), pp. 10464–10472. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. eprint: http://www.jneurosci.org/content/18/24/10464.full.pdf.

[5] Z. Bing, L. Cheng, K. Huang, Z. Jiang, G. Chen, F. Röhrbein, and A. Knoll. "Towards autonomous locomotion: Slithering gait design of a snake-like robot for target observation and tracking." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Feb. 2017, pp. 2698–2703. DOI: 10.1109/IROS.2017.8206095.

[6] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll. "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 (ICRA)*. May 2018, pp. 1–8. DOI: 10.1109/ICRA.2018.8460482.

[7] E. Boldog, T. E. Bakken, R. D. Hodge, M. Novotny, B. D. Aevermann, J. Baka, S. Bordé, J. L. Close, F. Diez-Fuertes, S.-L. Ding, N. Faragó, Á. K. Kocsis, B. Kovács, Z. Maltzer, J. M. McCorrison, J. A. Miller, G. Molnár, G. Oláh, A. Ozsvár, M. Rózsa, S. I. Shehata, K. A. Smith, S. M. Sunkin, D. N. Tran, P. Venepally, A. Wall, L. G. Puskás, P. Barzó, F. J. Steemers, N. J. Schork, R. H. Scheuermann, R. S. Lasken, E. S. Lein, and G. Tamás. "Transcriptomic and morphophysiological evidence for a specialized human cortical GABAergic cell type." In: *Nature Neuroscience* 21 (Aug. 2018), pp. 1185–1195.

[8] H. Corporation. *ACM-R5H*. Jan. 2019, Accessed 14.01.2019. URL: https://www.hibot.co.jp/ecommerce/prod-detail/14.

[9] D. Drubach. *The Brain Explained*. Prentice Hall, 2000. ISBN: 978-0137961948.

[10] D. E, N. Pradhan, S. R, and A. Sreedevi. "Analysis of the dynamic behaviour of a single Hodgkin-Huxley neuron model." In: *2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*. Dec. 2015, pp. 441–446. DOI: 10.1109/ERECT.2015.7499056.

[11] R. V. Florian. "Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity." In: *Neural Comput* 19.6 (June 2007), pp. 1468–1502. ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.6.1468.

[12] F. M. García-López P García-Marín V. "The discovery of dendritic spines by Cajal in 1888 and its relevance in the present neuroscience." In: *Progress in Neurobiology* 83.2 (Oct. 2007), pp. 110–130. DOI: 10.1016/j.pneurobio.2007.06.002. eprint: http://www.ane.pl/linkout.php?pii=7146.

[13] D. O. Hebb. *Organization of Behavior*. Psychology Press, 1949. ISBN: 978-0805843002.

[14] R. Hecht-Nielsen. "Theory of the backpropagation neural network." In: *Neural Networks* 1.1 (Jan. 1988), pp. 445–448. DOI: 10.1016/0893-6080(88)90469-8.

[15] G. Hinton and T. J. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press, 1999. ISBN: 0-262-58168-X.

[16] A. L. Hodgkin and A. F. Huxley. "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo." In: *The Journal of Physiology* 116.4 (Apr. 1952), pp. 449–472. DOI: 10.1016/0893-6080(88)90469-8.

[17] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning." In: (July 2018). eprint: arXiv:1807.01281.

[18] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. M. Zöllner. "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)*. Dec. 2016, pp. 127–134. DOI: 10.1109/SIMPAR.2016.7862386.

[19] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128×128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor." In: *IEEE Journal of Solid-State Circuits* 43.2 (Feb. 2008), pp. 566–576. ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

[20] W. Maass. "Networks of spiking neurons: The third generation of neural network models." In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7.

[21] C. Meschede. "Training Neural Networks For Event Based End To End Robot Control." 2017.

[22] T. Nakano, M. Otsuka, J. Yoshimoto, and K. Doya. "A Spiking Neural Network Model of Model-Free Reinforcement Learning with High-Dimensional Sensory Input and Perceptual Ambiguity." In: *PLoS Computational Biology* 10.3 (Mar. 2015). DOI: https://doi.org/10.1371/journal.pone.0115620.

[23] E. Nichols, L. J. McDaid, and N. Siddique. "Biologically Inspired SNN for Robot Control." In: *IEEE Transactions on Cybernetics* 43.1 (Feb. 2013), pp. 115–128. ISSN: 2168-2267. DOI: 10.1109/TSMCB.2012.2200674.

[24] W. G. Nicolas Frémaux Henning Sprekeler. "Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons." In: *PLoS Computational Biology* 21.2 (Apr. 2013), pp. 301–339. DOI: https://doi.org/10.1371/journal.pcbi.1003024.

[25] OpenAI. *OpenAI Five*. Jan. 2019, Accesed 14.01.2019. URL: https://blog.openai.com/openai-five/.

[26] W. S. M. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. eprint: https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf.

[27] K. A. Ponulak F. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning." In: *Acta Neurobiol Exp (Wars)* 71.4 (July 2011), pp. 409–433. eprint: http://www.ane.pl/linkout.php?pii=7146.

[28] W. Potjans, A. Morrison, and M. Diesmann. "Enabling Functional Neural Circuit Simulations with Distributed Computing of Neuromodulated Plasticity." In: *Frontiers in Computational Neuroscience* 4.141 (Nov. 2010), pp. 566–576. DOI: 10.3389/fncom.2010.00141.

[29] C. Robotics. *ROS Melodic Morenia*. Jan. 2019, Accessed 14.01.2019. URL: http://www.ros.org/about-ros/.

[30] C. Robotics. *V-REP*. Jan. 2019, Accessed 14.01.2019. URL: http://www.coppeliarobotics.com/.

[31] E. Rohmer, S. P. N. Singh, and M. Freese. "V-REP: A versatile and scalable robot simulation framework." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[32] B. Segee. "Methods in Neuronal Modeling: from Ions to Networks, 2nd Edition." In: *Computing in Science Engineering* 1.1 (Jan. 1999), pp. 81–81. ISSN: 1521-9615. DOI: 10.1109/MCISE.1999.743629.

[33] M. S. Shim and P. Li. "Biologically inspired reinforcement learning for mobile robot collision avoidance." In: *2017 International Joint Conference on Neural Networks (IJCNN)*. May 2017, pp. 3098–3105. DOI: 10.1109/IJCNN.2017.7966242.

[34] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2000. ISBN: 9780262193986.

[35] A. M. Wiebke Potjans and M. Diesmann. "A Spiking Neural Network Model of an Actor-Critic Learning Agent." In: *Neural Computation* 21.2 (Feb. 2009), pp. 301–339. DOI: https://doi.org/10.1162/neco.2008.08-07-593.