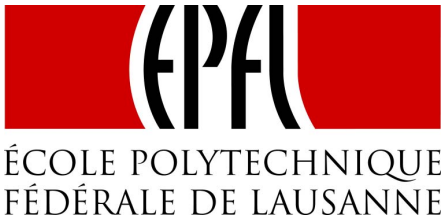
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
SCHOOL OF LIFE SCIENCES



Master project in Bioengineering

Development of Spiking Neural Networks on SpiNNaker platform for detection of direction of motion

Carried out in the Centre for Bio-Inspired Technology
at Imperial College London
Under the supervision of Dr Konstantin Nikolic

Done by

FAUSTINE GINOUX

Under the direction of
Prof. Felix Schürmann
The Blue Brain Project

EPFL

London, June 29, 2018

Acknowledgements

Firstly, I would like to thank my project supervisor Dr Konstantin Nikolic of the Centre for Bio-Inspired Technology (CBIT) in the Department of Electrical and Electronic Engineering at Imperial College London. Thank you for welcoming me at the CBIT and offering such an interesting and challenging project. Thank you for always having your office door open whenever I needed support and advice about my research. Last but not least, thank you for always having a positive attitude, making me feel that I was doing a great job, and motivating me to go even further.

Additionally, I would like to acknowledge my project director Prof. Felix Schürmann, co-director of the Blue Brain Project at the École Polytechnique Fédérale de Lausanne (EPFL). Thank you for showing enthusiasm in the project when I first came to ask you to be the EPFL responsible teacher. Thank you for taking the time to discuss the project and providing constructive criticism despite being hundreds of kilometers away. Finally, thank you for pushing me forward when I needed a boost halfway through the project.

Furthermore, I would like to express my profound gratitude to my parents for providing me with wholehearted support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without you. Thank you.

Summary

SpiNNaker is a massively parallel computing platform inspired by biological neural tissue. The SpiNNaker hardware can be used to model and efficiently simulate Spiking Neural Networks (SNNs). To aid in its use with a variety of existing neural network simulators, a special simulator-independent model description language has been developed, named PyNN. In this project the platform is used to simulate SNNs which take as input recordings of a moving ball generated by a Dynamic Vision Sensor (DVS) and give as output decision spikes about the direction of the ball movement. The DVS128 is a 128x128 pixel temporal contrast CMOS sensor that generates asynchronous data representing changes in illumination. The camera sends ON- and OFF-events corresponding to the addresses of the pixels which detected the illumination change. The general aim of this project is to develop skills to work with neuromorphic hardware that have applications in neuroscience, robotics and computer science. A specific aim of the project is to implement a biologically-inspired machine learning algorithm on a SpiNNaker board. The algorithm developed will be useful for the project that creates a robot which can stop a ball moving in a direction close to it (robot goalkeeper). In this project, the motor control of the robot is not studied. It focuses on the algorithm for recognition and prediction of the end-position of the moving ball. Leaky-Integrate-and-Fire (LIF) neurons and a Spike-Timing Dependent Plasticity (STDP) learning mechanism are implemented in a network consisting of three layers: an input layer of 1024 neurons with spikes from DVS data, a middle layer of 48 LIF neurons recognizing partial trajectories, and an output layer of two LIF neurons predicting the ball end-position. After training the SNN with four recorded files of trajectories ending in two distinct positions, the final model has an accuracy of 83% for end-position recognition and 53% for its prediction with half-trajectories.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine intelligence, more commonly known as Artificial intelligence, is one of the most fascinating research issues today and involves the study of cognitive processes and models. The fundamental difference between humans and machines is the ability of the human brain to adapt quickly to different environments and learn from experiences [1][2][3]; human beings are still orders of magnitude more intelligent than any computer program [4]. Although ethically questionable [5], a possible ultimate research objective could be the development of an intelligent agent [6]: a robot that perceives and communicates, through natural language, vision, sensors and movement. A challenge is to make the robot able to represent its own knowledge and reason, with the faculty to plan and act, by assimilating new knowledge from experience and interaction with its environment, and, globally, make a robot able to perform any task that we tend to consider typical of intelligent living beings [7].

During the last century, continuous progress in neuroscience research improved our knowledge of the structure and function of the human brain [8][9][10] and Artificial Neural Networks (ANNs) [3][11][12] have been developed in an attempt to capture and model its impressive computational capabilities. ANNs can adapt the structure and weights of their architecture, and thereby learn from examples and extract information autonomously, becoming able to organize from completely new data [12]. Advances in understanding the biological processes that govern learning in the brain have inspired researchers to develop new models of neurons and neural networks to implement machine learning mechanisms [13][14]. Inspired by the spiking behavior of biological neurons, by which high amounts of data can be encoded in the relative timing of spikes [15][16], a third generation of ANNs has been developed: Spiking Neural Networks [17][18].

As the complexity of neural networks increases, their simulation on conventional general-purpose computers becomes less efficient and specialized hardwares need to be developed [19][20][21]. One hardware of interest in this project is SpiNNaker [22][23], a novel computer architecture developed at the University of Manchester in the context of the Human Brain Project [8], which has its base at EPFL. SpiNNaker is inspired by the working of the human brain, which can be seen as a massively parallel computer, having billions of cells (neurons) all computing at the same time. This parallelism vastly multiplies the processing power of the brain, and a small SpiNNaker board (72 cores) makes it possible to simulate a network of tens of thousands of spiking neurons, process sensory input and generate motor output, all in real-time and in a low power system [23], which makes it ideal to use in robotic systems.

This report presents an outline of a neural network model for the visual input processing of a humanoid goalkeeper robot. Using inputs from a spiking silicon retina directly inspired

from the way biological retinas work [24], the neural network is simulated in real-time on a SpiNNaker board. These two components of the bio-inspired system mimic respectively the human eyes and brain. Learning is implemented through an unsupervised mechanism based on Spike-Timing Dependent Plasticity (STDP) [25][26][27], a biological process believed to be at the foundation of learning in the human brain [28] and the neuron model used throughout the project is the Leaky Integrate-and-Fire (LIF) neuron [29][30][31], one of the simplest - yet powerful - neuron models. In this project, the aim is to build a neural network able to predict the end-positions of recorded ball trajectories, when provided with visual inputs from a spiking silicon retina.

Project development

The first task of the project was to collect the data to be used as input for the neural network. Therefore, a simulation of a moving ball was realized, and the animation was recorded using a spike-based silicon retina dynamic vision sensor (DVS). Before being used as input, the data required pre-processing: noise filtering, dimensionality reduction, file conversion.

After some practice to get accustomed to the SpiNNaker device and the Python library for neural network PyNN [32], by reading tutorials and creating small neural networks (tasks of [33]), the first model to be implemented was a small two-layer neural network (counting the input layer) in which STDP learning was assisted by a supervisory signal. Throughout the project, this model is referred to as *Simple supervised model*. It is supervised in the sense that the desired output of the neural network is known: for each ball trajectory, as the files are recorded beforehand, we know the end-position of the ball and this information (label) is provided to the network.

When the physical goalkeeper robot will be built and receive live ball rolling towards it, labeled data will not be available. Therefore the aim of this project is to develop an unsupervised learning algorithm for the prediction of the end-position of the ball. If the supervisory signal of the simple supervised model is removed (as in the *Simple unsupervised model*), STDP learning is not efficient and the neural network is not able to correctly predict the end-positions of the ball. The proposed solution is to implement a three-layer neural network model.

The implementation of the three-layer model was done in two steps. First, the second layer was created to recognize parts of whole trajectories (*Unsupervised model with larger second layer*). Once this layer had learned and was able to discriminate different trajectories, the third layer was implemented to reconstruct complete trajectories and output the predicted end-position of the ball (*Unsupervised model with three layers*).

Throughout the project, training is done with four different ball trajectories, from two start-positions to two end-positions. Evaluation of the neural network performance was done by testing the model using unknown data:

- Same trajectories as for the training but other recordings
- Trajectories with shifted start-positions
- First half of recorded trajectories.

Chapter 2

Background

2.1 Ball catching models

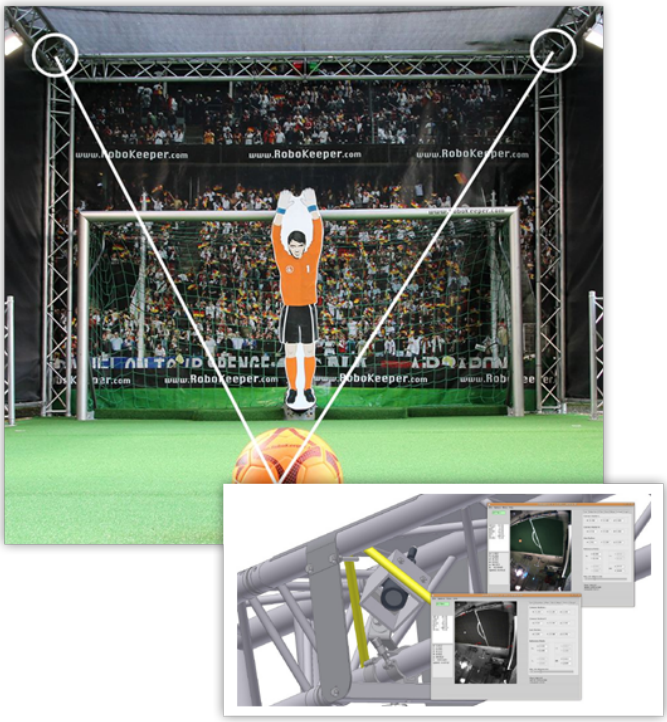


Figure 2.1: **RoboKeeper.**
Image from `http://www.robokeeperme.com`. RoboKeeper is a computer-controlled soccer goal-keeper that is (almost) impossible to beat. Even professional football players have experienced how hard it is to score against the RoboKeeper. The prediction mechanism is based on the trajectory calculation which yields the end-position coordinates of the ball.

Robot goalkeepers have already been developed and proved to excel in their task of stopping balls: even the best football professionals like Messi and Neymar have failed to defeat the robot goalie RoboKeeper[1][2]. This robot's efficiency relies in its detection mechanism. It uses two cameras to snap an image of the ball (Figure 2.1), which is transmitted to a computer every 0.02 second (50 Hz). The system extracts the ball coordinates from each frame and uses them to calculate the projectile's complete ballistic curve and predict the 3D end-position of the ball.

[1] `http://www.robokeeperme.com`
[2] `https://www.youtube.com/watch?v=NSP8J5jlfnM`

This information is then transmitted to the motor control that controls the goalie's motion, which is deployed to the predicted point of impact.

Although this detection mechanism is very efficient, algorithms such as image processing, computer vision, object recognition and ball trajectory prediction are not implemented in a biologically plausible way. Indeed, a human being trying to catch a ball does not calculate the trajectory to predict the end-position [34][35][36]. Instead, he or she tries to maintain the ball at a certain alignment in his or her field of view (Figure 2.2). This can be mathematically modeled although it is not consciously calculated by the catcher. If the ball trajectory is parabolic, the ball can only be caught if the acceleration of the tangent of the elevation angle (α) between the ball and the person is zero [37]:

$$\frac{d^2(tan(\alpha))}{dt^2} = 0. \tag{2.1}$$

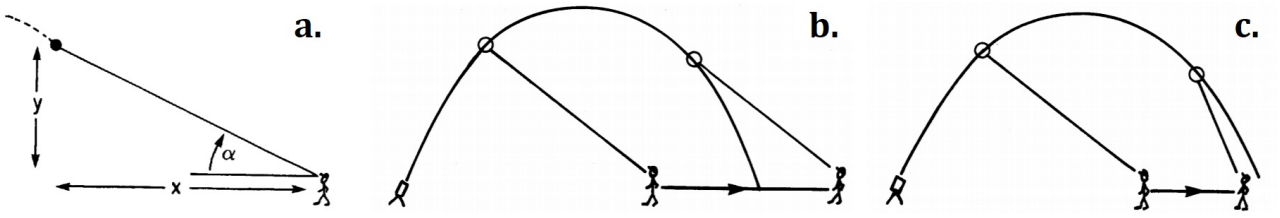


Figure 2.2: **Catching a ball with parabolic trajectory.**
From [38]: **a)** Angle of elevation in the human ball-catching model. **b)**) Maintaining α constant is a poor strategy: the catcher is running backwards instead of running forwards. **c)** Maintaining the acceleration of $tan(\alpha)$ constant is more effective. A human being does not consciously keep this relationship, he or she manages to do so by keeping the ball at the same position in his or her field of view.

Note that parabolic trajectories are not the most common in a soccer game and are usually used for scoring from a long distance [39]. In the simulations used throughout the project, the ball is rolling on a surface and trajectories are straight. Also, the camera is fixed and does not follow the ball as a human goalkeeper would move his head and body.

2.2 Physiology of neurons and neural networks

The human brain is composed of more than 86 billions of neurons [40], considered to be the most important type of cells in the brain [41]. Each neuron cell consists of a soma, an axon and one or several dendrites (Figure 2.3). Neurons are connected to each other by specialized electrical or chemical connections called synapses [42] (over 100 trillions in the human brain [43]), the junctions between the axon of the pre-synaptic neuron and dendrite(s) of the post-synaptic neuron. Neurons communicate using electrochemical pulses known as action potentials, transmitted through the synapses: most neurons receive signals via their dendrites and send out signals down their axon. Synapses can be excitatory or inhibitory (increasing or decreasing the activity in the target cell) and, depending on the signal pattern they received, many synapses are able to change their strength. This activity-dependent modification of synapses is believed to be the fundamental mechanism for learning and memory in the human brain [42].

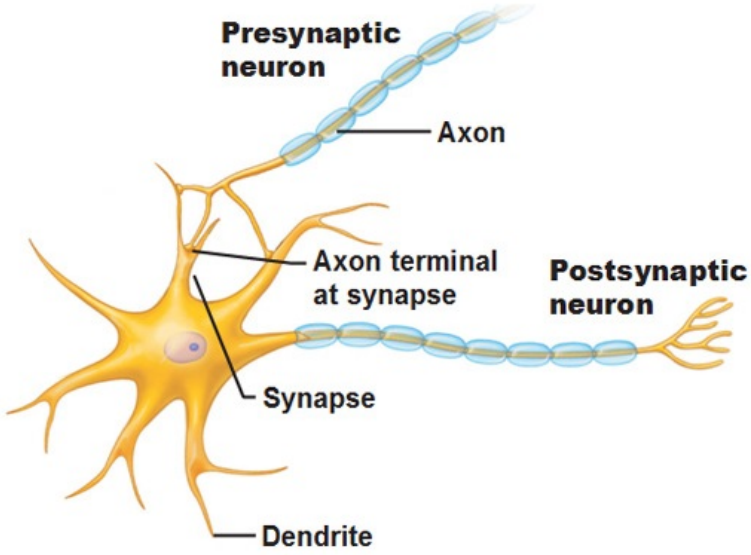


Figure 2.3: **Two neurons connected by synapses.**
From Pearson Education, Inc©. A neuron cell consists of a soma (cell body, with the nucleus shown in purple), an axon and dendrites. The connection between a neuron's axon and another neuron's dendrite is called a synapse. The signal flows in one direction, along the axon of the pre-synaptic neuron to the post-synaptic neuron, which can in turn propagate the signal along its own axon if the conditions to trigger an action potential are met.

Neurons are electrically excitable and their membrane potential is used for transmitting signals, generated by the opening or closing of ion channels in the membrane [44]. As many biological cells, the difference in electric potential between the interior and the exterior of a neuron is non-zero, instead this membrane potential is around -65 mV [45] (resting potential). The cell membrane is an ion barrier between the intracellular and extracellular environments, and ion flow across the membrane through ion channels influences the neuron state and is at the origin of action potentials [46]. When an external stimulus triggers the opening of sodium channels, sodium ions enter the cell, increasing the membrane potential (depolarization). If depolarization is sufficiently large, it can evoke an action potential. Once saturation is reached, potassium channels open, and potassium ions flow out of the cell (hyperpolarization), bringing the membrane potential to its reset potential, around -70 mV (Figure 2.4). Many factors influence the membrane potential of neurons, including various types of ion channels, which can be chemically-gated and voltage-gated [46]. Voltage-gated ion channels are controlled by the membrane potential, and the membrane potential itself is influenced by these ion channels, and the mechanisms behind neurotransmission and action potentials are complex. As this section is a brief introduction to neurons and neural networks, it will not go into deeper details.

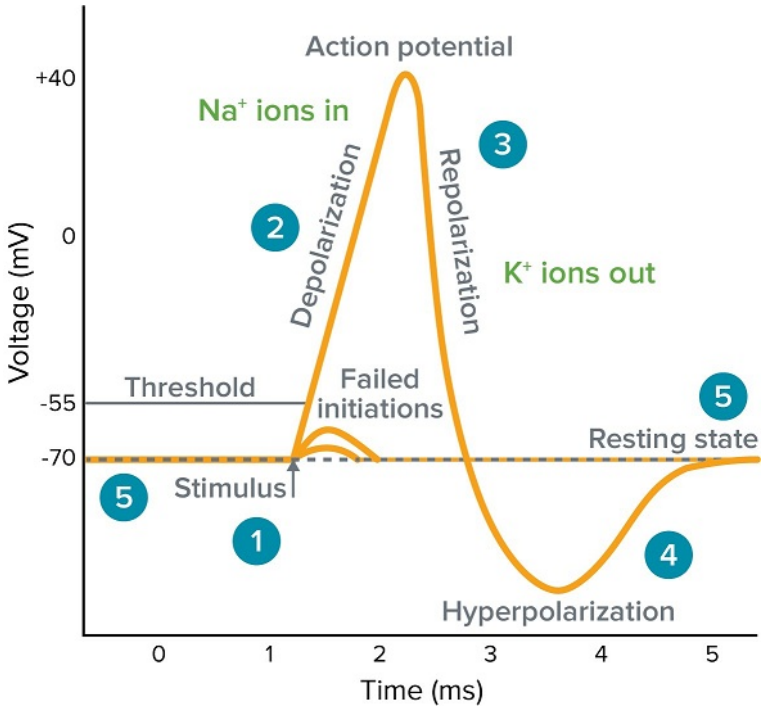


Figure 2.4: **Evolution of the membrane potential during an action potential.** From `https://www.moleculardevices.com/`. An action potential is a rapid rise and subsequent fall in voltage or membrane potential across a cellular membrane. **1.** Stimuli start the rapid change in voltage, known as action potential. Membrane potential must be raised above the threshold voltage to start membrane depolarization (rapid rise in membrane potential). **2.** Depolarization is caused by the opening of sodium channels in the cellular membrane, resulting in a large influx of sodium ions. **3.** Repolarization results from rapid sodium channel inactivation as well as a large efflux of potassium ions resulting from activated potassium channels. **4.** Hyperpolarization is a lowered membrane potential caused by the efflux of potassium ions and closing of the potassium channels. **5.** Resting state is when membrane potential returns to the resting voltage, the same as before the stimuli occurred.

2.3 Spiking Neural Networks

Spiking Neural Networks (SNNs) are a type of Artificial Neural Networks (ANNs) which aim to reproduce realistically the behavior of the mammalian brain [47]. SNNs consist of neurons which communicate through spikes [48], the timing of the spikes being used to encode and compute information [49][50][51]. A neuron fires - produces a spike or action potential - only when its membrane potential reaches a specific threshold value. After firing, the signal from the pre-synaptic neuron propagates to post-synaptic neurons, usually with a delay to represent the propagation along the axon, which is not modeled. The membrane potential of post-synaptic neurons is influenced by the incoming signal of one or several pre-synaptic neuron(s) (Figure 2.5). As in biological neurons, if the pre-synaptic neuron is excitatory, the membrane potential of each post-synaptic neuron increases (Excitatory Post-Synaptic Potential, EPSP), whereas if the pre-synaptic neuron is inhibitory, the membrane potential of each post-synaptic neuron decreases (Inhibitory Post-Synaptic Potential, IPSP). In turn, if the membrane potential of the post-synaptic neuron reaches its threshold value, it produces a post-synaptic spike, which can be transmitted to other neurons (becoming the pre-synaptic spike for these neurons), and so on. Thus, excitatory pre-synaptic neurons encourage post-synaptic neurons to fire and inhibitory pre-synaptic neurons make it harder for post-synaptic neurons to fire. When no signals arrive to a neuron, its membrane potential decays until it reaches its resting potential, and stays stable as long as no other spikes arrive. Furthermore, as in biological systems, each neurons can be modeled to enter a refractory period upon firing, during which it is unable to produce a new spike.

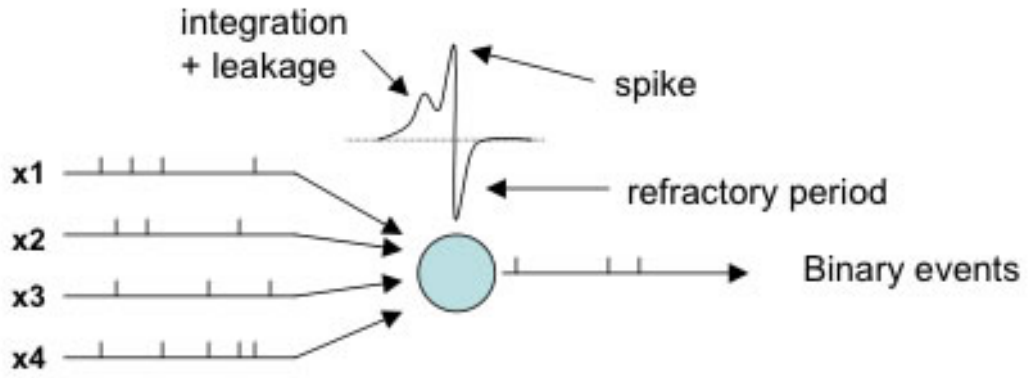


Figure 2.5: **Spike integration.**
Pre-synaptic spikes arriving from neurons **x1**, **x2**, **x3** and **x4** influence the membrane potential of the neuron (in blue). When no spikes arrive at the neuron, the membrane potential decays until new spikes arrive. When enough excitatory spikes trigger the membrane potential rises to the threshold, the neuron fires and its membrane potential is reset to a lower voltage. It then enters the refractory period during which it cannot fire again. Then, the integration process starts again.

Numerous Spiking neuron models have been described in literature [48][52] and this project uses the Leaky Integrate-and-Fire (LIF) model [29][30][31] and a Spike-Timing Dependent Plasticity (STDP) learning rule [26][27].

Leaky Integrate-and-Fire model

The LIF model is one of the most common and simplest spiking neuron models and is easy to analyze and simulate [48]. As the action potential is due to the flow of ionic charge across the neuron membrane [53], the neuron can be modeled by a basic resistor-capacitor (RC) circuit composed of a resistor and a capacitor in parallel (Figure 2.6), driven by a current $I(t)$, as described by:

$$I(t) = \frac{v(t)}{R_m} + C_m \frac{dv}{dt} \tag{2.2}$$

Equation 2.2 can be multiplied by R_m to introduce the time-constant $\tau_m = R_m C_m$ of the leaky integrator. This yields the standard differential equation form:

$$E = \tau_m \frac{dv}{dt} = -v(t) + R_m I(t) \tag{2.3}$$

In this equation, $v(t)$ represents the instantaneous membrane potential of the neuron, R_m the membrane resistance of the neuron, and τ_m is referred to as its membrane time-constant. The neuron is leaky, meaning its membrane potential decays over time to a resting potential V_{rest}. Without the resistance term, equation 2.2 describes a simple integrate-and-fire model [30], with no time-dependent memory. Therefore the integration of $I(t)$ depends on the membrane capacitance, and the leakage comes from the membrane resistance, as it is not a perfect insulator. In both cases, the spikes are not modeled explicitly. Instead, when $v(t)$ reaches the spiking threshold V_{thresh}, it is instantaneously reset to a voltage V_{reset}, typically lower than V_{rest}. The models become more realistic when a refractory period is implemented, maintaining the membrane potential at a low voltage during τ_{refrac}, before the integration process is started again.

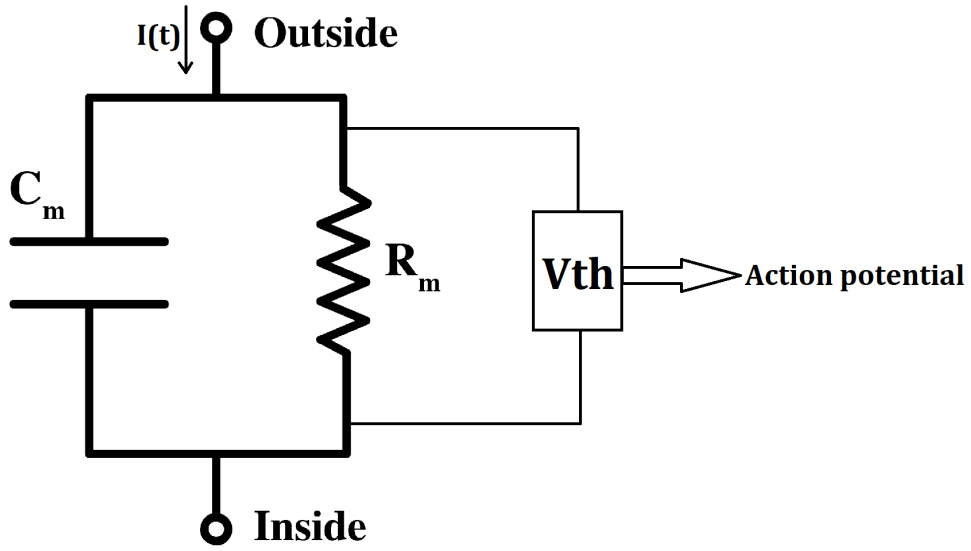


Figure 2.6: **Equivalent electrical circuit of the LIF neuron cell membrane.** C_m represents the membrane capacitance and R_m the membrane resistance. $I(t)$ is the input current and depends on pre-synaptic spikes. When the membrane potential reaches the threshold V_{th}, the neuron emits a spike. Its membrane potential is then reset to the voltage V_{reset}, and the neuron is prevented from firing during a refractory period τ_{refrac}. After this time, the integration process can start again.

Spike-Timing Dependent Plasticity

Spike-Timing Dependent Plasticity (STDP) [25] is an adaptation of the Hebbian Learning Rule [27] and takes into account the relative timing of the pre- and post-synaptic action potentials [54][55][56], respectively the spikes received and emitted by a neuron. Synapses are plastic and their weights are changed depending on the relative timing of pre- and post-synaptic spikes. If the pre-synaptic neuron fires slightly before the post-synaptic neuron, it is presumed that the pre-synaptic spike triggered the post-synaptic spike; therefore, the synaptic strength between the two neurons is enhanced, a process known as Long-Term Potentiation (LTP). On the contrary, if the pre-synaptic neuron fires slightly after the post-synaptic neuron, the spike received after emission cannot have triggered the emitted spike; the synaptic strength is weakened, a process known as Long-Term Depression (LTD). This is believed to be a fundamental mechanism for learning and memory in the human brain [28][55][57].

The relative timing of the spikes used to update the weights is based on pairs of pre- and post-synaptic spikes (spike pair rule). The strengthening and weakening of synapses with STDP depend on time-dependence and weight-dependence rules corresponding to the STDP learning window (Figure 2.7). The total weight change Δw_j of a synapse (j) from a pre-synaptic neuron induced by a stimulation protocol with pairs of pre- and post-synaptic spikes is then [58][59]:

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f), \tag{2.4}$$

where t_j^f labels the pre-synaptic spike arrival times at synapse j for f pre-synaptic spikes and t_i^n corresponds to the n firing times of the post-synaptic neuron at synapse i.

The time-dependence rule is described by the following equations:

$$W(\Delta t) = A_+ \exp(\frac{-\Delta t}{\tau_+}), \text{ if } \Delta t < 0 \tag{2.5}$$

$$W(\Delta t) = A_- \exp(\frac{-\Delta t}{\tau_-}), \text{ if } \Delta t > 0 \tag{2.6}$$

Δt is the time difference $t_i^n - t_j^f$ corresponding to $t_{post} - t_{pre}$ as in equation 2.4, A_+ and A_- determine the maximum amounts of synaptic modification and τ_+ and τ_- determine the ranges of pre-to-post synaptic interspike intervals over which synaptic change occurs. Intuitively, the closer in time the pre- and post-synaptic spikes are, the larger the weight change is: the change in weight drops off exponentially as the time between the spikes gets larger (Figure 2.7). Some researchers have also suggested a correlation between triplets and quadruplets of pre-synaptic and post-synaptic spikes to trigger synaptic potentiation or depression [60][61], but these will not be discussed here as the spike pair rule is used throughout the project.

There are two common weight dependence rules for the update of the synaptic weight Δw_{ij}: the additive weight dependence rule and the multiplicative weight dependence rule (Figure 2.8). In the additive case, the weight is updated simply by adding to the current weight. The actual amount added or subtracted depends on the timing of the spikes, as determined by the timing rule, in which A_+ and A_- are constants. In the multiplicative condition, the weight change depends on the difference between the current weight and the maximum weight allowed (w_{max}) for potentiation, and on the difference between the current weight and the minimum weight possible (w_{min}, typically zero) for depression. The value of A_+ and A_- are then respectively multiplied by this difference to give the maximum weight change; again the

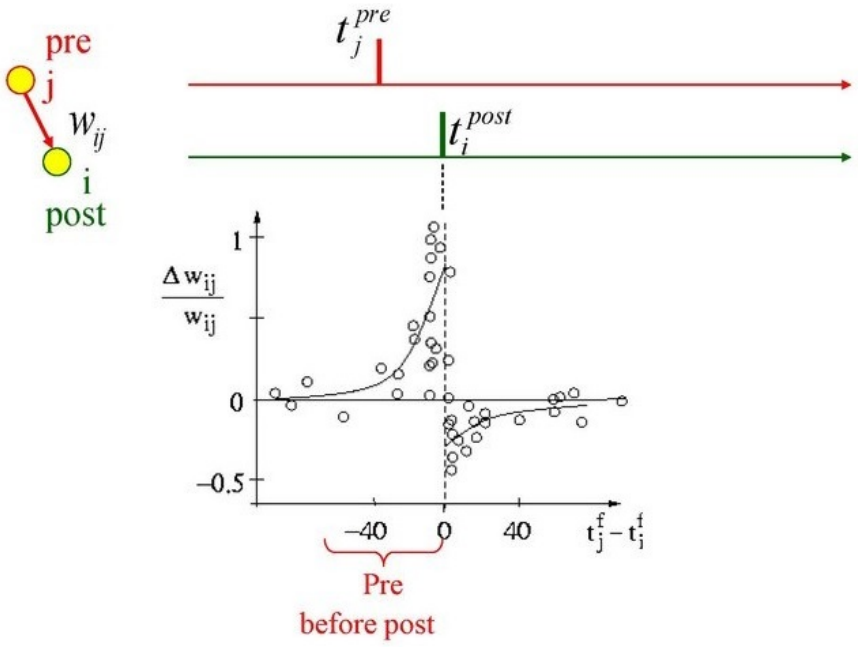


Figure 2.7: **Spike-Timing Dependent Plasticity learning curve.**
From Sjöström and Gerstner (2010), Scholarpedia. The change of the synapse plotted as a function of the relative timing of pre- and post-synaptic action potentials is called the STDP function or learning window and varies between synapse types. The rapid change of the STDP function with the relative timing of spikes suggests the possibility of temporal coding schemes on a millisecond time scale.

actual value depends on the timing rule and the time between the spikes, but the closer the weight is to the maximum or minimum weight allowed, the smaller the amount of weight added or subtracted is. In equations 2.5 and 2.6, the parameters A_+ and A_- are replaced by Δw_+ and Δw_- as follows:

$$\Delta w_+ = A_+ * (w_{max} - w) \tag{2.7}$$

$$\Delta w_- = A_- * (w - w_{min}) \tag{2.8}$$

Note that in usual STDP implementations, STDP is triggered both on pre-synaptic spike arrival (LTD) and post-synaptic spike emission (LTP), known as pre-post-sensitive scheme (Figure 2.9). However, the event-address mapping and the distributed synaptic weight storage schemes used in parallel neuromorphic hardware such as SpiNNaker make the pre-post-sensitive scheme of STDP implementation inefficient [62]. In the alternative implementation on SpiNNaker, referred to as pre-sensitive scheme, STDP is triggered only on pre-synaptic spike arrival (LTD and LTP). Also, in the implementation of STDP on SpiNNaker, the plasticity mechanism is only activated when the second pre-synaptic spike is received at the post-synaptic neuron. Thus at least two pre-synaptic spikes are required for the mechanism to be activated[4].

[4] `http://spinnakermanchester.github.io/spynnaker/4.0.0/NewPlasticityRules-LabManual.pdf`

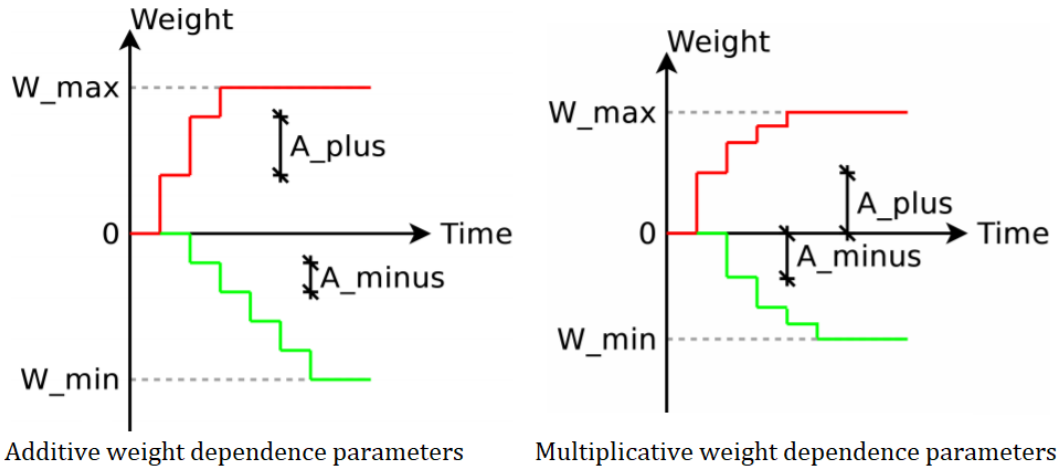


Figure 2.8: **Spike-Timing Dependent Plasticity weight dependence parameters: additive vs multiplicative update.**
From SpiNNaker Manchester[3]. In the additive weight dependence rule, the weight is updated by adding a constant weight to the current weight. In the multiplicative condition, the weight change depends on the difference between the current weight and the maximum/minimum weight allowed for potentiation/depression. The value of A_+ and A_- are then respectively multiplied by this difference to give the maximum weight change: the closer the weight is to the maximum or minimum weight allowed, the smaller the weight change is. In both rules, the actual value depends on the timing rule and the time between the spikes.

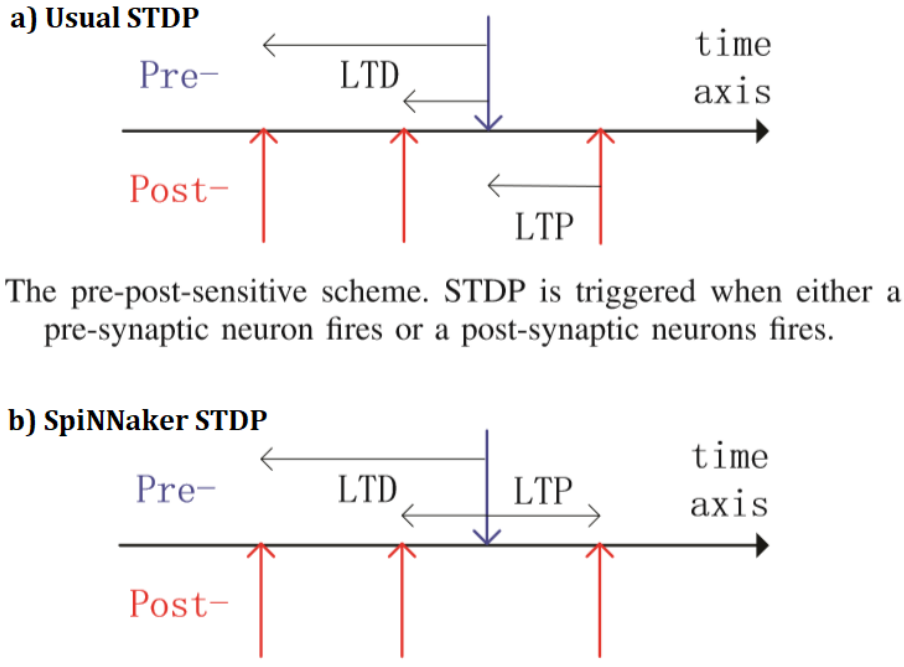


Figure 2.9: **Spike-Timing Dependent Plasticity activation: pre-post-sensitive scheme vs pre-sensitive scheme.**
From [62]. **a.** In usual STDP implementations, STDP is triggered both on pre-synaptic spike arrival (LTD) and post-synaptic spike emission (LTP). **b.** However, this is not possible on SpiNNaker since it uses a distributed memory system: in the SpiNNaker implementation, STDP is triggered only on pre-synaptic spike arrival (LTD and LTP).

2.4 Similar project by Bichler et al. (2011)

In the paper *Unsupervised Features Extraction from Asynchronous Silicon Retina through Spike-Timing-Dependent Plasticity* by Bichler et al. [28], a task similar to the aim of this project is performed. Based on input data from AER dynamic vision sensor [24] recordings of cars passing under a bridge (Figure 2.10), a feed-forward multilayer spiking neural network is implemented, using a fully unsupervised STDP learning rule with lateral inhibition and 10 parameters in all for the neurons. The neural network learns to count the cars passing on each traffic lane by extracting the temporally correlated features. The middle layer recognizes chunks of the trajectories and the output layer can identify entire traffic lanes by recombining partial trajectories, in order to count the number of passing cars. The neural network proves to be very efficient at this task: after 10 minutes of real-life data, it is able to detect cars with a 95% accuracy rate.

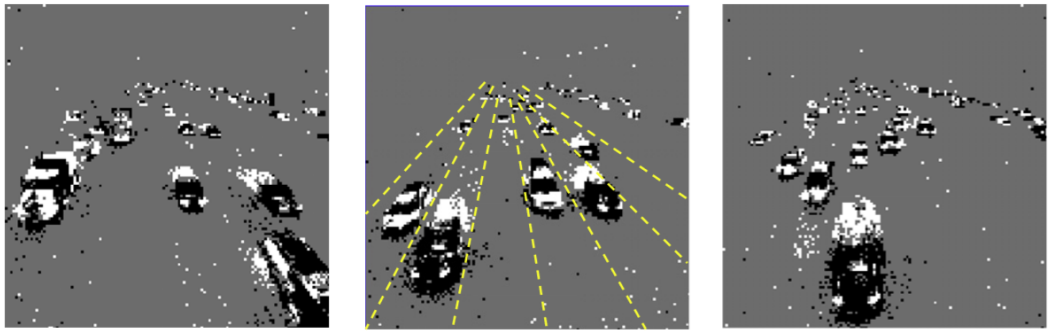


Figure 2.10: **Illustration of the dataset used by Bichler et al.**
From [28]. The dataset used by Bichler et al. are AER sequences of cars passing under a bridge in Pasadena recorded with the DVS128 silicon retina (same camera as in this project). In the center image, the delimitations of the traffic lanes are materialized with yellow dashed lines. White pixels represent ON-events (positive change in illumination) and black pixels OFF-events (negative change).

Neural network implementation

The input layer consists of the 128x128 spiking neurons extracted from the spiking silicon retina recordings. The middle layer is composed of 60 neurons and the output layer of 10 neurons (Figure 2.11). Every input neuron requires two synapses, to send the ON- and OFF-events respectively (see section 3.1), thus the total number of synapses is $2 * 128 * 128 * 60 + 60 * 10 = 1,966,680$ synapses for which STDP learning is implemented. In addition to these plastic synapses, lateral inhibition connections are present in both the middle and output layers: within a layer, each neuron is connected to every neuron with an inhibitory static synapse, without spatially specific inhibition.

Two unsupervised learning strategies are successively tested. In the first one, which the authors call "global learning", the two layers learn concurrently and lateral inhibition is always enabled. In the second strategy, "layer-by-layer" learning is implemented: only the middle layer is active in a first step, and once the learned features are stable, lateral inhibition and STDP are disabled for this layer. Only after this step is the output layer allowed to learn and lateral inhibition is also removed afterwards. The second method proved to be more efficient, but the learning time is doubled.

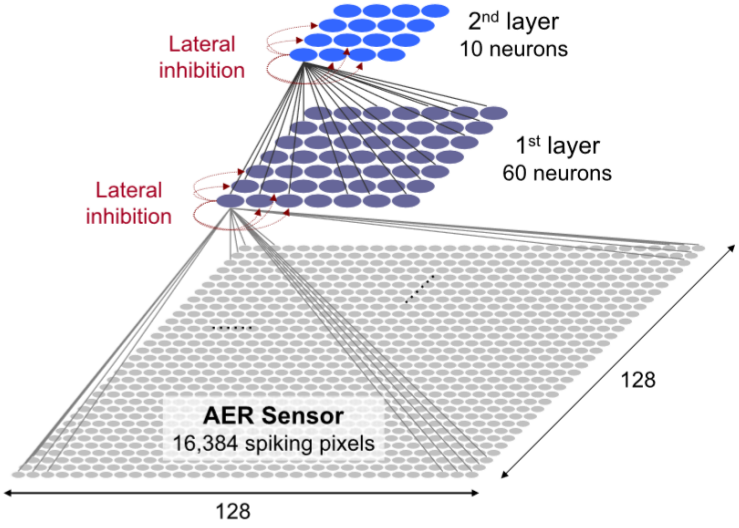


Figure 2.11: **Bichler et al.'s neural network topological overview.**
From [28]. The neural network is a two-layer feedforward fully connected network, with complete lateral inhibition, from each neuron to every other neuron. There is no spatially specific inhibition between neurons. Strictly speaking, the bottom layer is not considered as a layer of the neural network and represents the AER sensor.

Similarities

As in the paper by Bichler et al., the dataset used as input in this project is recorded with a spiking silicon retina, the DVS128 camera. The same neuron model is used (LIF neurons) as well as the same learning rule (STDP, although the implementation is different). The objectives are also similar: extracting temporally correlated features to recognize trajectories (a ball in this project and cars in their case). Therefore, the neural network architecture developed in this project is inspired by the one presented in the paper. Both final models consist of an input layer from the AER sensor, a middle layer recognizing partial trajectories, and an output layer reconstructing whole trajectories. The learning is done in a "layer-by-layer" strategy, in which only the middle layer is active in a first step. Once the middle layer has been trained, learning is disabled in this layer (no more STDP and lateral inhibition) and the training of the output layer is implemented.

Differences

In the car counting task, the neural network is very efficient at detecting cars in different traffic lanes because there is no particular correlation between trajectories of different lanes (cars rarely change lanes). Therefore, two groups of synapses belonging to different traffic lanes cannot in average be potentiated together and neurons necessarily become more sensitive to one of the groups. However, in our ball end-position prediction task, ball trajectories spatially overlap. They can have the same start-position but different end-positions, and even if the start- and end-positions of two trajectories are different, trajectories can still cross each other. This means that the same group of neurons can fire for different trajectories.

Another difference lies in the neural network implementation. Bichler et al. developed a special purpose C++ event-based simulator and use a simplified STDP learning rule (Figure 2.12). In this custom rule, all the synapses of a neuron are equally depressed upon receiving a post-synaptic spike, except for the synapses that were activated with a pre-synaptic spike

a short time before, which are strongly potentiated. Thus, there is no LTD time-constant (parameter τ_-) restricting the time-window of synaptic depression. In the PyNN STDP implementation used throughout this project, it is impossible to do so and strict limitations apply to the parameters range.

Note that in this project, ball trajectories are presented one by one and do not timely overlap, as there is only one ball on a soccer field. Thus, if lateral inhibition is not deactivated after the learning, it does not prevent the recognition of the end-position of the ball, whereas in the counting car task, some might be missed.

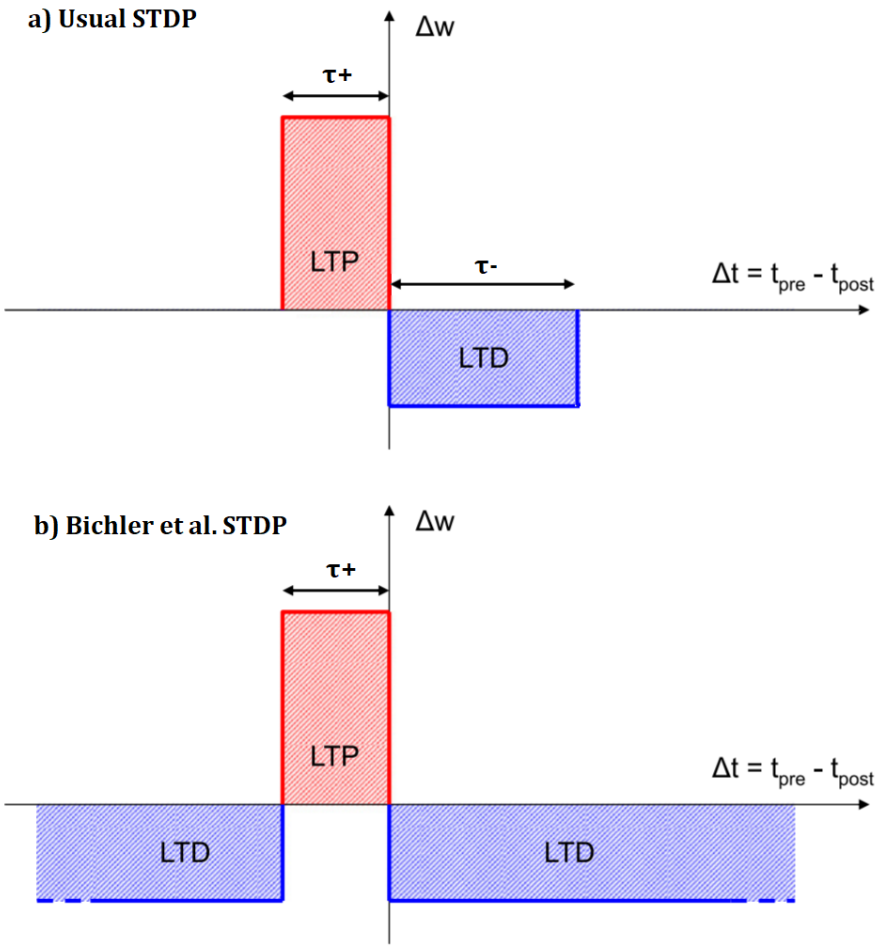


Figure 2.12: **Spike-Timing Dependent Plasticity learning rule: usual time-windows vs Bichler et al's simplified rule.**
Adapted from [28]. **a.** The synapse undergoes Long-Term Potentiation (LTP) when $0 \leq t_{post} - t_{pre} \leq \tau_+$ and Long-Term Depression (LTD) when $0 \leq t_{pre} - t_{post} \leq \tau_-$. **b.** The synapse undergoes Long-Term Potentiation (LTP) when $0 \leq t_{post} - t_{pre} \leq \tau_+$ and Long-Term Depression (LTD) otherwise.

Chapter 3

Materials and methods

3.1 Data collection

Visual simulation of a moving ball

The first task of the project was to record trajectories of a ball rolling towards a camera. The use of a virtual simulation for this task has numerous advantages. First, on the practical aspects, it is much more convenient than rolling a ball at a camera in person. Additionally, a simulation is more reliable as it increases precision in the ball positions, which are also better controllable and thus the trajectories are exactly reproducible. Finally, environmental noise such as changing light levels or shadows can be circumvented.

The simulation was implemented on Simulink[1], a graphical programming environment developed by MathWorks for modeling, simulating and analyzing multi-domain dynamical systems. This modeling engine uses a GUI and does not require to write pure code. Different components of the simulation are inserted as blocks and the inputs and outputs can be connected in a logical manner (Figure 3.1). It works together with MATLAB and the virtual world editor V-Realm Builder [63]. The output of the script, corresponding to the current ball position, is fed into the 3D virtual world where it controls the position of a white sphere object representing the ball (Figure 3.2). The ball position being continuously updated according to its speed, start-position and end-position parameters provided in Simulink, the sphere in the VR world moves and the animation can be recorded.

The input of the function is a dimensionless position descriptor. It is produced by integrating the output of a built-in velocity block with respect to time. This position input is multiplied by a speed parameter, defined by the user, which yields the distance traveled by the ball at a given time. According to the start- and end-positions parameters provided by the user, the x- and z- coordinates of the ball are updated, corresponding to its lateral and longitudinal positions respectively, and the y-coordinate, its vertical position, remains constant as the ball rolls on a flat field. Once the traveled distance exceeds a threshold, the ball position is reset and the animation restarted.

The ball is represented by a white sphere of radius 2 with high luminosity to make it clearly visible (Figure 3.2). Note that the brown surface is made for the aesthetics and that only the ball movement is recorded (see paragraph Dynamic Vision Sensor in section 3.1). In the project design, I decided to record trajectories beginning at five distinct start-positions and finishing at three distinct end-positions, in a narrower alignment as a goal net would be on a real soccer

[1] `https://www.mathworks.com/products/simulink.html`

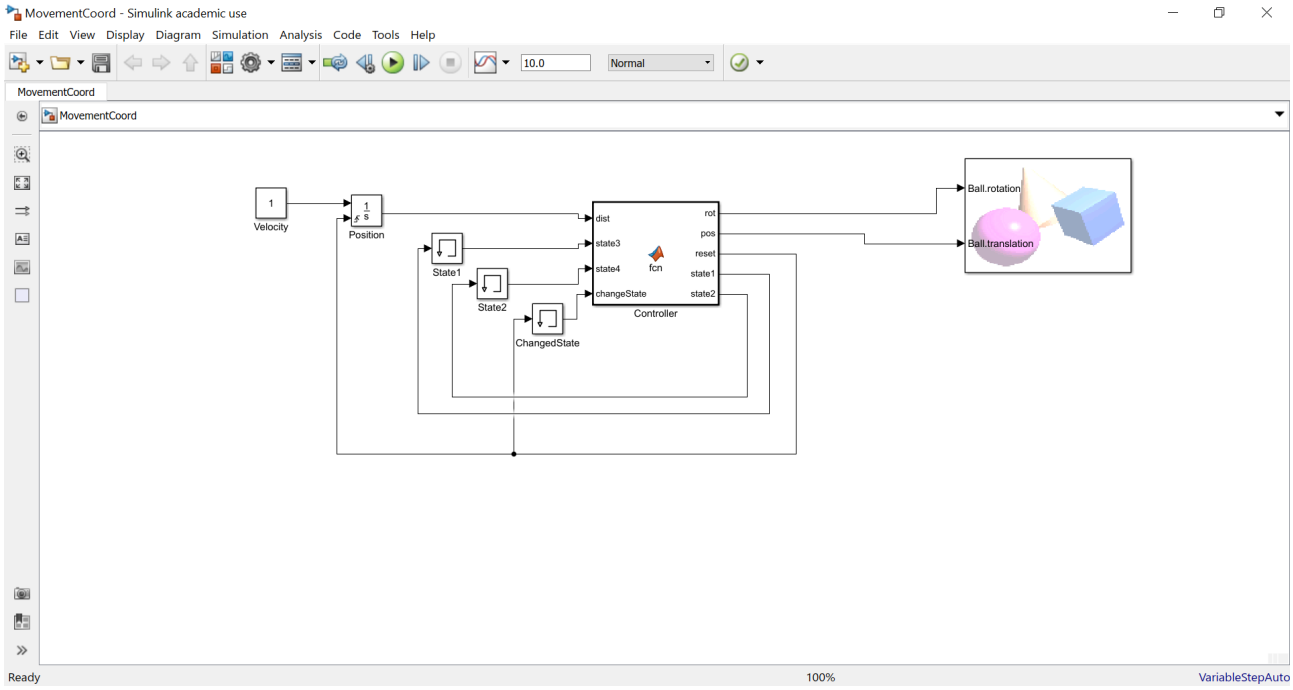


Figure 3.1: **Simulink environment.**

The graphical programming environment Simulink was used to create the simulations of a rolling ball. Different components of the simulation are inserted as blocks and the inputs and outputs can be connected in a logical manner. The output of the velocity block is integrated with respect to time to yield the position. This position is used as an input of the script function. The function calculates the ball position, depending on the provided speed, start- and end-positions parameters. The position and rotation outputs are fed into the V-Realm Builder block, to update the visual simulation.

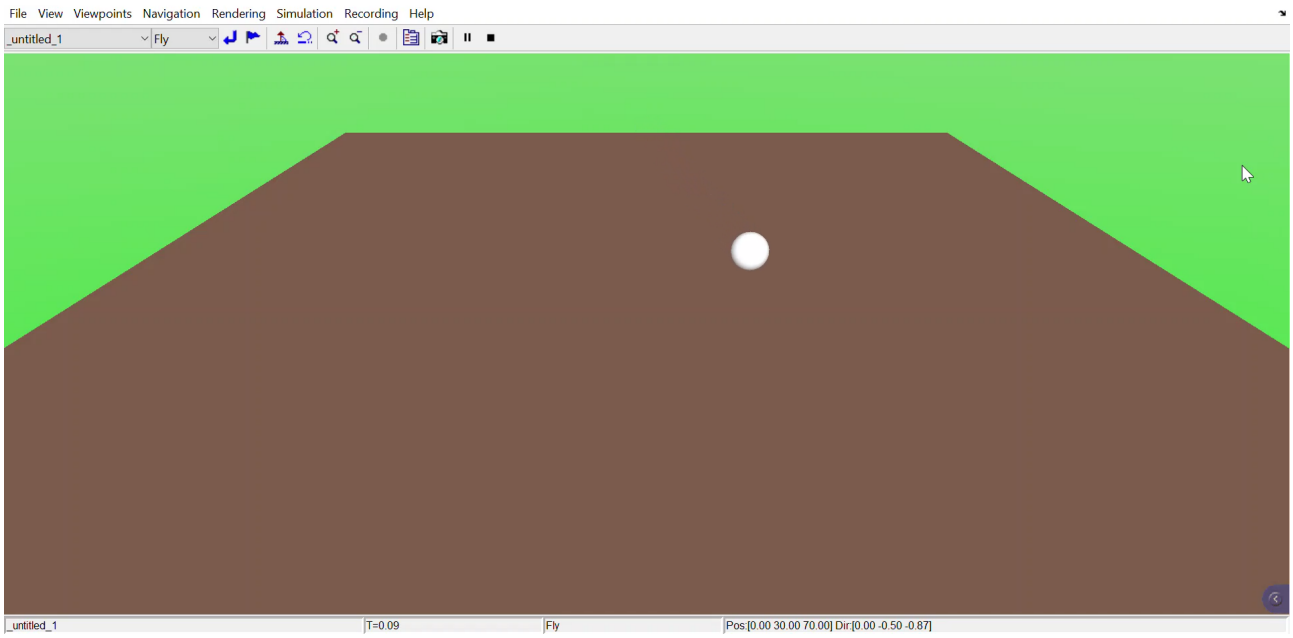


Figure 3.2: **VR world simulation.**

The 3D VR world consists of a brown surface and a white sphere. The position of the sphere is continuously updated and the animation of the moving ball can be recorded.

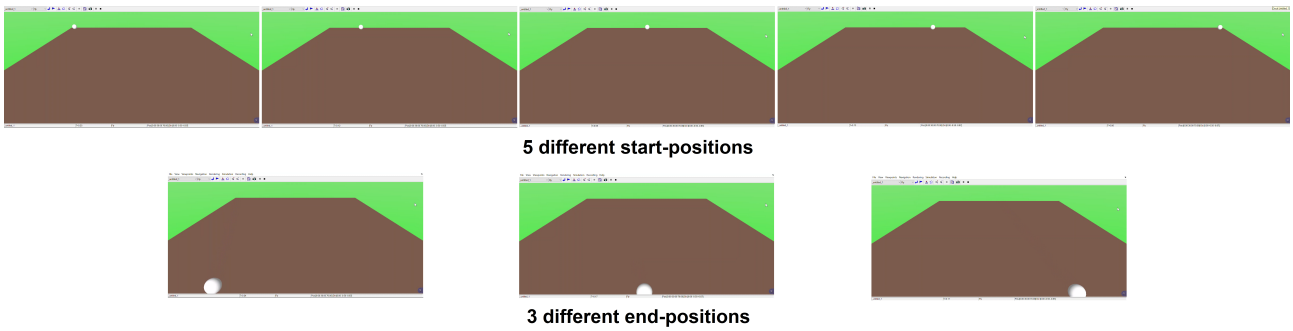


Figure 3.3: **Start- and end-positions used in the simulation.**
Trajectories can begin at five distinct start-positions and finish at three distinct end-positions, and each of the 15 trajectories was recorded three times.

field (Figure 3.3). Each trajectory was recorded three times, yielding a total of 45 recording files for 15 distinct trajectories. Another choice was to set the view point of the simulation slightly above the surface's height, in order to have a better overview of the field and a better vision of a ball's trajectory. Other simplifications include a static view point, both in position and direction, in opposition to a goalkeeper that moves laterally and is able to turn his head. Also, the ball trajectories are rolling on a flat surface and the aim of this project is to find a position on a line, without taking height into account. Table 3.1 provides a summary of the parameters used for the 3D world visualization.

	x position(s)	**y position**	**z position**
View point	0	30	70
Start-positions	[-50, -25, 0, 25, 50]	2.1	-50
End-positions	[-20, 0, 20]	2.1	50

Table 3.1: **Parameter values of the 3D world simulation.**
The parameters in this table can be used to reproduce the visual simulation with V-Realm Builder. Note that the view point orientation is $x = -1$ and that it has a rotation of 30°. The 2.1 values for y positions come from the fact that the ball has a radius of 2 and the brown surface a thickness of 0.1.

Dynamic Vision Sensor

Dynamic Vision Sensors (DVS) are a new way of doing machine vision overcoming the limitations of conventional machine vision [24]. DVS cameras are inspired by the human retina and the brain, and present unprecedented advantages compared to conventional frame-based cameras (or static vision sensors): ultra-low response latency, low data rates, high dynamic range and low power consumption [64]. The DVS128 camera[2] (Figure 3.4) is a neuromorphic visual system developed by iniLabs[3] (now iniVation[4]) and was used to record the data during the project.

[2] `https://inivation.com/support/hardware/dvs128/`
[3] `https://inilabs.com`
[4] `https://inivation.com`

Figure 3.4: **DVS128 camera: a camera that behaves like the human eye.**
Power, data storage and computational requirements are drastically reduced, and the dynamic sensor range is increased by orders of magnitude due to the local processing - no sending out of entire images at fixed frame rates. Only the local pixel-level changes caused by moving in a scene are transmitted, at exactly the time they occur. The result is a stream of events at microsecond time resolution, equivalent to or better than conventional high-speed vision sensors running at thousands of frames per second. More information on the DVS128 camera: `https://inivation.com/support/hardware/dvs128/`.

For the analysis of moving objects, only the local changes at the pixel level are necessary, whereas a large amount of redundant information is produced by conventional cameras. These redundancies must be further eliminated by video compression algorithms, a process that is computationally expensive [21][65][66]. The redundant information leads to wasted memory access and storage, and energy and CPU are wasted by redundant computation. Moreover, the high latency requires waiting for unnecessary frames. DVS cameras are an efficient alternative as they are event-based and respond to relative changes in luminosity in their field of view. DVS128 uses Address-Event-Representation (AER) communication protocol and records the luminosity changes as spikes at the individual pixel level (128x128 resolution), sampled at 1 MHz corresponding to the microsecond time-resolution. AER is based on the principle that the firing of a neuron is modeled as a pure asynchronous event. The only information saved are the spike location, explicitly encoded as an address (x and y coordinates), the time at which the event occurred (implicit time-stamp), as well as its polarity (ON or OFF). Respectively, an ON-event corresponds to an increase in luminosity and an OFF-event to its decrease.

To visualize, record and save the data, the jAER[5][6] Java open source software developed by iniLabs was used. Recorded AER data are saved in the AEDAT 2.0[7] format (*.aedat* extension) which uses 8 bytes for each event (Figure 3.5). Currently, the AEDAT data format cannot interface directly with the SpiNNaker platform. Therefore, a class (*aedatObj*) with its set of methods and a set of functions (*handleAER.py*) were implemented to extract and manipulate the AEDAT files. It is used to perform data processing and convert the recordings to MAT files to be further used by SpiNNaker.

[5] `https://github.com/SensorsINI/jaer`
[6] `https://inivation.com/support/software/jaer/`
[7] `https://inivation.com/support/software/fileformat/`

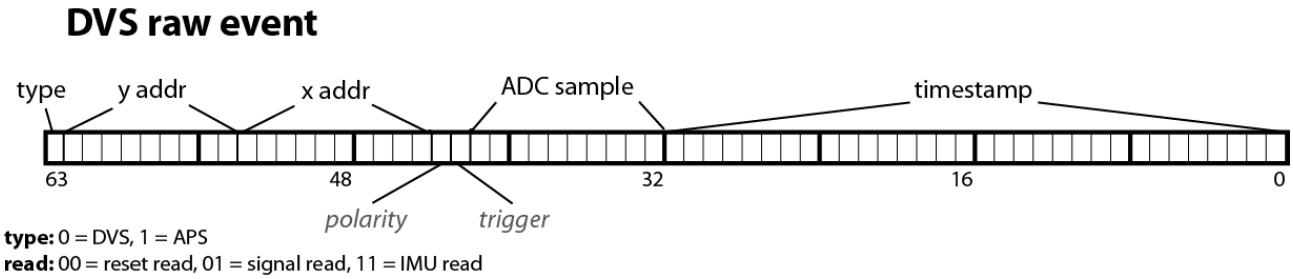


Figure 3.5: **AEDAT format.**
Each series of [address, time-stamp] pairs correspond to a spike event. The address and the time-stamp are both 32 bit wide, for a total of 8 bytes per event. The time-stamp is in microseconds, while the address has to be interpreted according to a specific jAER AEChip class definition of that address. In the scope of this project, a class has been implemented to "translate" AEDAT files into MAT files. More information on the AEDAT format: `https://inivation.com/support/software/fileformat/`.

3.2 Data processing

Noise filtering

The data recorded with the DVS camera is noisy and spikes unrelated to the moving ball must be filtered out. Although the ball is clearly recognizable on the original noisy recordings, the problem arises when the resolution is reduced. In this case, if all spike events are kept, the noise density is amplified by 16 on a 32x32 recording and by 64 on a 16x16 recording! Hence, a filtering algorithm is implemented on the original 128x128 recordings (Figure 3.6).

The approach to apply the filter is defining a square mask of a given size and move it across the 128x128 spatial frame for each time-step. As the number of spikes per microsecond is only of the order of 10 in the whole spatial frame, all spikes occurring within a given time-window are considered. Concretely, the filtering function goes through each recorded spike-event, checks for other events falling within a 20 ms window and within a 5x5 pixel mask, and counts the number of ON and OFF events. If the number of spikes exceeds 12, the middle pixel is set to an ON or OFF event according to the polarity of the majority of the events. The 20 ms time-window is chosen based on the exposure time of the DVS camera, the size of the mask and value of the spike threshold are determined empirically after trying out several values. The decision is based on the filtering efficiency evaluated by looking at the images created.

Resolution reduction

Spatial resolution

The resolution of the DVS camera is 128x128 pixels. Each pixel is represented by one neuron, yielding a total of 16384 neurons for the first layer only, which exceeds the number of neurons the 4-node SpiNNaker board is able to handle. Therefore, the resolution of the recordings has to be downsampled to 32x32, by compressing all spike-events into a smaller x-y space. Note that a 64x64 pixel resolution is still too large to use with SpiNNaker as the number of synapses per neuron is relatively large.

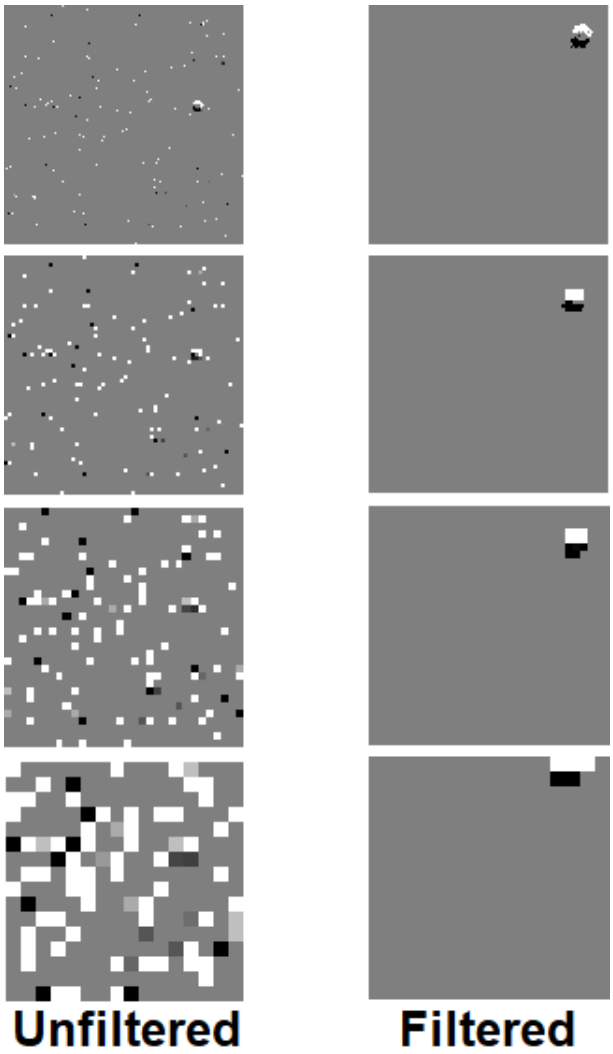


Figure 3.6: **Comparison of noisy data to filtered data.**
Image made with jAER. Resolution from top to bottom: 128x128, 64x64, 32x32, 16x16. Although the ball is clearly recognizable on the original noisy 128x128 picture, it becomes more difficult when resolution is reduced as noise is amplified. From visual validation, we can see that the filtering algorithm is very efficient at removing unrelated spikes.

Time resolution

The sampling rate of the DVS camera is 1 MHz, which means that changes are recorded every microsecond, but the time-resolution of SpiNNaker is limited to 1 millisecond. Therefore, all time-stamps of the spike events must be converted to the millisecond scale by dividing their values by 1000 and truncating the result to an integer. In this simplification, events from neurons that are spiking more than once within a millisecond are lost. To prevent the loss of important information, only ON-events are kept prior to the resolution reduction. Since OFF-events correspond to the disappearance of the object in this part of the field of view, and ON events to its appearance, keeping both can be considered as having redundant information. Indeed, in our case, the only feature that matters is when the ball appears. How long it remains at this specific position is not important as we know that the ball is moving at a constant speed and cannot be in two places at once. It was demonstrated by a previous student that using only ON-events or only OFF-events does not affect significantly the training and accuracy of the neural network. Note also that all recordings are normalized to 3000ms. The longest original recording is 5.9s long, and it was previously shown that speeding it up to a factor two is the limit.

Conversion to MAT files

The SpiNNaker platform is not compatible with AEDAT format from jAER, therefore information needs to be extracted from the AEDAT file before being exported to SpiNNaker. The AER data is unpacked and the events' address, polarity and time-stamps are saved in a MAT file, together with other useful information: filename, resolution, maximum number of events (for safety), recording length, end-position of the trajectory. A MAT file is a binary data container format used by MATLAB, and data can be easily loaded from or written to these files in Python using the *scipy.io* Python library (*loadmat* and *savemat* functions).

To handle AEDAT files, the Python library *paer*[8] had been developed by a previous student. It is supposed to be used to extract, unpack, downsample, normalize and convert AEDAT files, however it often fails to extract data. All events in the AEDAT format are ordered by their time-stamps, and should guarantee time-stamps monotonicity, meaning the next event will always have an equal or greater time-stamp[9]. However, jAER sometimes generates empty new line and spike events with incorrect time-stamps in the data stream, therefore the monotonicity is not respected anymore. To cope with this issue, I created my own *aedatObj* class in *handleAER.py* to read and use AEDAT files, as well as to perform other useful tasks such as pre-processing (downsampling, noise filtering), conversion, truncation or concatenation... This class and its methods as well as other functions are inspired from several sources (including the *paer* and *processAEDAT*[10] libraries), and functions were created and added to this file when required throughout the project.

[8] https://github.com/darioml/python-aer

[9] `https://inivation.com/support/software/fileformat/`

[10] `https://github.com/SensorsINI/processAEDAT/blob/master/misc/loadaerdat.m`

3.3 Neural network simulation

Neuromorphic hardware: SpiNNaker

SpiNNaker is a neuromorphic hardware developed at Manchester University in 2006 [22][23]. Its novel computer architecture is inspired by the fundamental structure and function of the human brain [67], which itself is composed of billions of simple computing elements communicating using unreliable spikes[11]. SpiNNaker provides a platform for high-performance massively parallel processing appropriate for the simulation of large-scale neural networks in real-time. The largest SpiNNaker machine will be composed of over 50 000 chips (one million processors, or cores [68]) and capable of simulating up to a billion simple neurons, or millions of neurons with complex structure and internal dynamics.

A small SpiNNaker board is a suitable hardware for the development of a robot goalkeeper as it is mobile and makes it possible to simulate a network of tens of thousands of spiking neurons, process sensory input and generate motor output, all in real-time and in a low-power system. It breaks the rules followed by traditional supercomputers that rely on deterministic, repeatable communications and reliable computation[12]. Its nodes, housing the multiprocessor system-on-chip, communicate using simple messages (spikes) that are inherently unreliable: data communication between cores (single processors) take place via multicasting packets which are source-routed, meaning that they only contain the information of the issuer and the network infrastructure is responsible for delivering them to their destinations. Therefore, the communication in SpiNNaker is Address-Event-Represented asynchronously. In other words, each chip of a SpiNNaker node incorporates a router for the AER-based routing of the multicast packets. The multicast router of each chip can replicate the data packets where necessary and distribute to different cores, so that one pre-synaptic neuron can connect to multiple post-synaptic neurons, and it is this fast computation protocol that allows the processing of several thousands of spiking neurons in real-time [23]. Note that there is no common shared memory which can be accessed by every simulated neuron on a SpiNNaker engine.

The SpiNNaker used in this project is a single printed circuit board (PCB) with four nodes (SpiNN-3, Figure 3.7). Each of these chips contains 18 ARM processor cores (Figure 3.8), and each core is able to emulate up to 255 virtual artificial neurons. Therefore, in theory, the SpiNNaker board can simulate up to 18360 simple neurons, however out of the 72 cores, typically 64 are deployed as application cores, four as monitor processors and there remain four spare cores. This small SpiNNaker board requires a 5V 1A supply, and can be powered from some USB2 ports. The control and I/O interface is a single 100 Mbps Ethernet connection. For other architectural and technical aspects, please visit `http://apt.cs.manchester.ac.uk/projects/SpiNNaker/architecture/`.

PyNN - Python library for Neural Networks

PyNN is a high-level neural network descriptive and modeling language that is simulator-independent [32], which means that the same code written using the PyNN API and the Python programming language can be run without modification on any simulator that PyNN supports (currently NEURON, NEST and Brian) as well as on certain neuromorphic hardware systems, such as SpiNNaker. Although running the code written for this project with another simulator

[11] `http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/`
[12] `http://apt.cs.manchester.ac.uk/projects/SpiNNaker/`

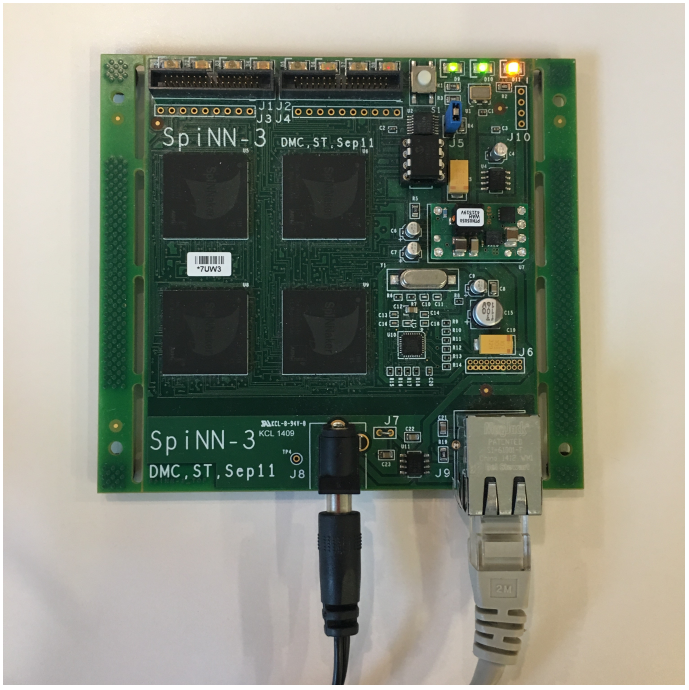


Figure 3.7: **SpiNNaker SpiNN-3 board.**
SpiNNaker is a neuromorphic hardware developed at Manchester University. Its architecture is inspired by the human brain. It is used to simulate large-scale neural networks in real-time. The SpiNN-3 is a single PCB with four nodes. It requires a 5V/1A supply and the control and input/output interface is a single 100 Mbps Ethernet connection.

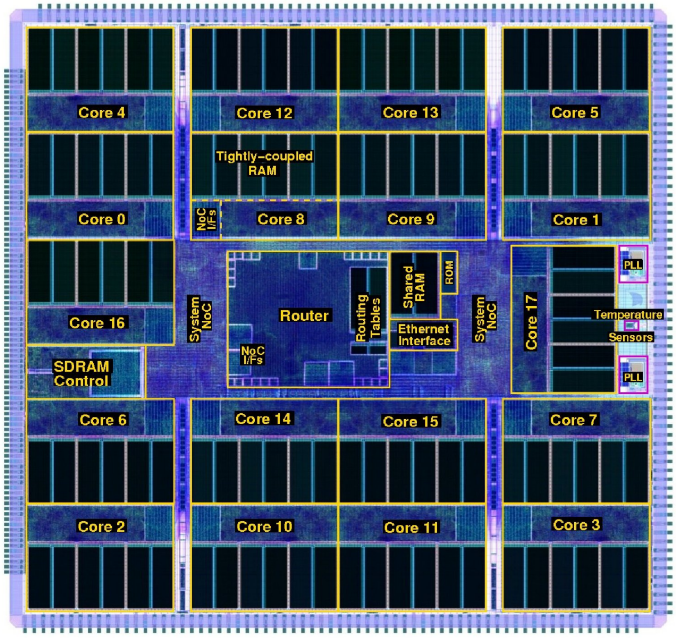


Figure 3.8: **SpiNNaker chip architecture.**
Each chip contains 18 ARM processor cores and each core is able to emulate up to 255 virtual artificial neurons. 16 cores are deployed as application cores, one as monitor processor, and there remains one spare core. The SpiNN-3 board contains four chips.

is not planned for the moment, using sPyNNaker - the PyNN implementation for SpiNNaker - presents the advantage of providing built-in neuron and synapse models, without having to specify any mathematical description (differential equations) of the models, solely the parameters.

PyNN implementation for SpiNNaker: sPyNNaker[13]

Whereas PyNN provides models that are available in at least two of the simulation engines it supports, sPyNNaker imposes several limitations on neuron types, spike sources, synapse types and wiring methods. Below are listed the models supported by sPyNNaker. Description of how to use these models (and other models supported by PyNN) as well as their source code can be found on `http://www.neuralensemble.org/docs/PyNN/index.html`. Further limitations of sPyNNaker can be found on `http://spinnakermanchester.github.io/spynnaker/4.0.0/SPyNNakerLimitations.html`.

1. Neuron model types:
    - Current based Leaky-Integrate-and-Fire with exponential decay
    - Conductance based Leaky-Integrate-and-Fire with exponential decay
    - Current based dual Leaky-Integrate-and-Fire with exponential decay (**2** excitatory and 1 inhibitory exponentially decaying synaptic input per neuron)
    - Current based Izhikevich with exponential decay
    - Conductance based Izhikevich with exponential decay
    - Current based Leaky-Integrate-and-Fire with 1 excitatory and 1 inhibitory **delta** synaptic input per neuron
2. External input - injecting spikes into a PyNN model:
    - Spike source array - input of a predefined set of spikes
    - Spike source Poisson - input of randomly generated spikes at a predefined mean rate generated from a Poisson distribution
3. Connectors:
    - All neurons in the pre-population to all neurons in the post-population connector
    - Connectivity specified from file
    - Connectivity specified from list
    - Distance dependent probability connector
    - Fixed number of randomly selected connections
    - Fixed number of randomly selected neurons in the post-population connector (to all neurons in the pre-population)
    - Fixed number of randomly selected neurons in the pre-population connector (to all neurons in the post-population)
    - Fixed probability connectivity between all neurons in the pre-population to all neurons in the post-population

[13] `https://github.com/NeuralEnsemble/PyNN/blob/master/doc/connections.txt`

- One to one connector (requires same number of neurons in pre- and post-populations)
- Small world connector

4. Plasticity - Spike-Timing-Dependent Plasticity (STDP) mechanism:

   - Timing dependence rules:
     - Pfister spike triplet rule - considers sets of three spikes (i.e., two pre- and one post-spikes or one pre- and two post-spikes)
     - Spike pair rule - the amount of potentiation or depression decays exponentially with the time between each pair of pre- and post-spikes
     - Spike nearest pair rule - similar to the Spike pair rule, but only the nearest pair of pre- and post-spikes are considered
   - Weight dependence rules:
     - Additive weight dependence - change in weight related only to the timing between the spikes determined by the timing rule
     - Multiplicative weight dependence - change in weight related additionally to the difference between the current and the maximum / minimum weight allowed as specified in the rule parameters

Chapter 4

Implementation

4.1 Simple supervised model

The first model to be implemented is a two-layer neural network (Figure 4.1), in which the first layer corresponds to the inputs of the spiking silicon retina and the second layer outputs the expected end-position of the ball. Additionally, a stimulating population (teaching neurons) is implemented as in the STDP Network of [33] to supervise the learning of the output layer. The visual inputs having a resolution of 32x32, the first layer consists of 1024 spiking neurons. In the second layer, there is one neuron for each end-position, and as much in the stimulating population.

The spikes of the neurons of the first layer are provided by a spike source array, each pair of address and time-stamp from the AER data respectively corresponding to a neuron and its firing times. The teaching neurons are also initialized by a spike source array comprised of one spike per trajectory for the neuron corresponding to the expected end-position of this trajectory, and spiking at the end of it (supervisory signal).

The second layer (output layer) consists of Leaky Integrate-and-Fire (LIF) neurons with the parameters summarized in Table 4.1, connected to the first layer in an all-to-all fashion with Spike-Timing Dependent Plasticity (STDP, parameters of Table 4.2) and connected to each other by lateral inhibition (weight of -10: difference between V_{thresh} and V_{reset}). This layer is also connected to the stimulating population in a one-to-one fashion with a static synapse (weight of 100: strong to make sure that the target neuron spikes). This results in the output layer spiking as expected during the learning phase; then, the stimulation is stopped (no more supervisory signal from teaching neurons) and if learning was effective, the output layer continues to spike in an appropriate manner in response to the inputs. Initially, the synaptic connections between the input and output layers come from a uniform weight distribution. At each time-step, using the multiplicative weight-dependence rule, connections which are believed to be involved in the spiking of the post-synaptic neuron are strengthened, and connections which are not involved are weakened. Note that the STDP implementation on SpiNNaker corresponds to a pre-sensitive scheme (Figure 2.9 in Section 2.3 of Background Chapter), meaning that if a neuron never fires, the weights of its synapses remain constant. Thus, randomly high initial weights can remain high despite the neuron not being involved in the trajectory recognition.

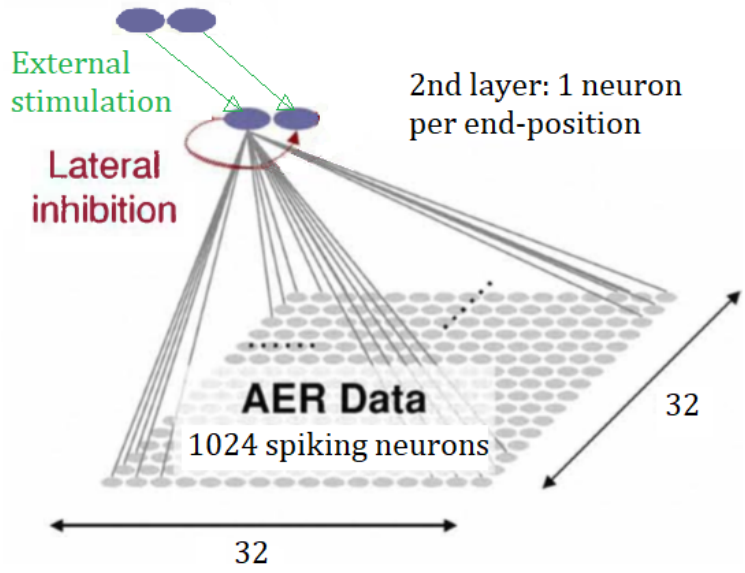


Figure 4.1: **Simple supervised model: with external stimulation.**
Figure design inspired by [28]. The first neural network to be implemented is a feedforward fully-connected network in which each neuron of the output layer is connected to all neurons of the input layer by plastic synapses ($1024 * 2 = 2048$ synapses in total). Lateral inhibition between neurons of the output layer is complete, from each neuron to every other neuron, without spatially specific inhibition. When a post-synaptic neuron spikes, its synapses that were the most recently activated are potentiated and its synapses activated slightly after are depressed. Due to lateral inhibition, the integration of other post-synaptic neurons is inhibited. To drive the firing of neurons in the output layer, a stimulation layer is implemented. As the end-position for each trajectory is known, the teaching neuron is used to make the corresponding output neuron fire accordingly, in order to help the network know which synapses to strengthen, thus to drive the learning towards the expected neural network response.

Parameter	Value	Unit	Description
C_m	70	nF	Capacitance of the LIF neuron
τ_m	20	ms	Membrane time-constant
$\tau_{refract}$	20	ms	Refractory period - membrane potential is clamped to V_{reset}
V_{reset}	-70	mV	Reset potential after spike
V_{rest}	-65	mV	Membrane rest potential
V_{thresh}	-60	mV	Threshold voltage at which spike is emitted
$\tau_{syn,E}$	2	ms	Excitatory input current decay time-constant
$\tau_{syn,I}$	25	ms	Inhibitory input current decay time-constant
i_{offset}	0	nA	Offset current: base input current to add at each time-step

Table 4.1: **Parameter values of the LIF neurons in the output layer of the supervised model.**
Note that τ_m cannot be larger than 300ms, the time between two samples in the recordings. If it were, spikes from two trajectories could interfere.

Parameter	Value	Unit	Description
$delay$	1	ms	Synaptic delay
τ_+	30	ms	Long-Term Potentiation time-constant
τ_-	30	ms	Long-Term Depression time-constant
A_+	0.5	nA	Maximum positive weight change
A_-	0.5	nA	Maximum negative weight change
w_{min}	0	nA	Minimum synaptic weight
w_{max}	10	nA	Maximum synaptic weight

Table 4.2: **Parameter values of the STDP function in the supervised model.** The synaptic delay corresponds to the spike travel time between the pre- and post-synaptic neurons, to model the signal propagation along the neuron axon, which is not modeled. The LTP and LTD time-constants (τ_+ and τ_-) determine the interval over which synaptic potentiation and depression respectively occur, in this case, the STDP window is symmetric. Due to the limitations of the SpiNNaker hardware, τ_+ and τ_- are limited; if they are too large, packets are lost meaning that spikes do not reach their target neurons. Note that synaptic weights are in nA and not µS as the current-based LIF neuron is used and not the conductance-based.

4.2 Simple unsupervised model

The same neural network as described in section 4.1 is used, but this time the stimulation population is removed (Figure 4.2). The activity of the second layer only depends on the visual input layer and weight update is now fully unsupervised. As detailed in the results chapter, this network does not seem to be able to learn to recognize different trajectories and end-positions. The hypothesis advanced is that the recordings are too long compared to the learning window of the output neurons, which cannot be increased due to SpiNNaker hardware limitations. This means that one neuron is not able to learn a whole trajectory, therefore a neural network with more neurons in its second layer is the next model to be implemented.

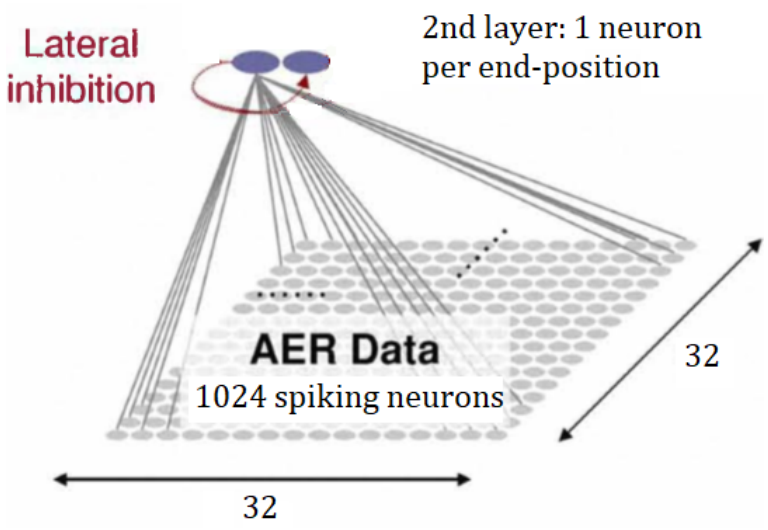


Figure 4.2: **Simple unsupervised model: no external stimulation.**
Figure design inspired by [28]. The neural network is a feedforward fully-connected network in which each neuron of the output layer is connected to all neurons of the input layer by plastic synapses ($1024 * 2 = 2048$ synapses in total). Lateral inhibition between neurons of the output layer is complete, from each neuron to every other neuron, without spatially specific inhibition. When a post-synaptic neuron spikes, its synapses that were the most recently activated are potentiated and its synapses activated slightly after are depressed. Due to lateral inhibition, the integration of other post-synaptic neurons is inhibited.

4.3 Unsupervised model with larger second layer

As one output neuron is not able to recognize a whole trajectory, a neural network with a larger second layer is implemented (Figure 4.3), so that each trajectory is encoded by a specific sequence of spiking neurons. The choice of 48 neurons corresponds to 12 neurons per trajectory if no neuron spikes for two different trajectories. Having 48 neurons spiking mean that each neuron responds to 250 ms of one trajectory ($3000/12 = 250$), which is less than the membrane time-constant ($\tau_m = 299ms$) that should be slightly larger than the pattern to be learned [28].

However, it is unlikely that the 48 neurons spike as trajectories are not completely uncorrelated: some neurons fire for partial trajectories present in several whole trajectories. Thus the aim is to have neurons that are specialized and able to recognize parts of a complete trajectory, before implementing a third layer to recombine these trajectory parts to predict the end-position of the ball based on the specific firing pattern of layer 2. In this implementation, the neuron and STDP parameters are modified as compared to the small neural network model and are described in Tables 4.3 and 4.4.

In this implementation, the maximal initial weight must be smaller than the maximal weight of the neural network, to avoid strong synaptic weights triggering noisy spikes unrelated to the trajectory. This is done for two reasons. Firstly, the effect of LTD is small, and large initial weights are not decreased enough. Secondly, in the pre-sensitive STDP implementation on SpiNNaker, if a neuron never fires, the weights of its synapses remain constant. This means that randomly high initial weights can remain high, despite the input neuron not being involved in the trajectory recognition. Therefore, once all trajectories have been presented, all weights that were not updated are set to zero.

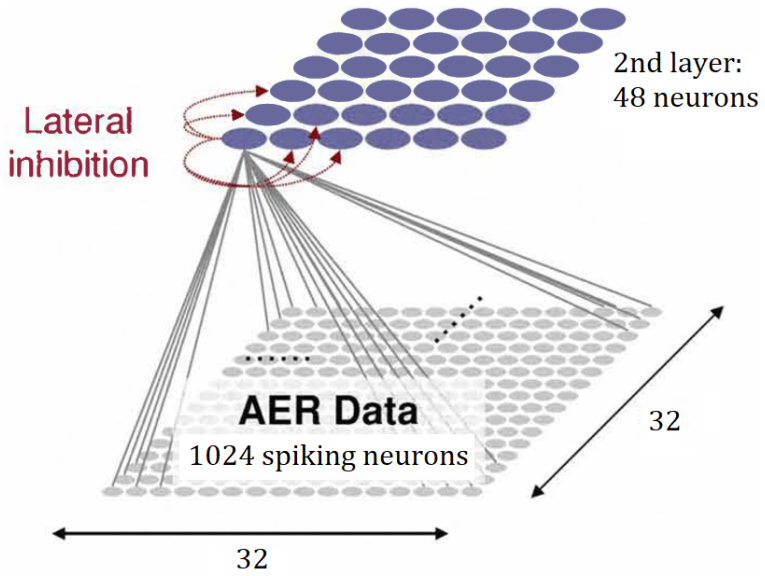


Figure 4.3: **Unsupervised model with larger second layer.**
Figure design inspired by [28]. The neural network is a feedforward fully-connected network in which each neuron of the output layer is connected to all neurons of the input layer by plastic synapses ($1024 * 48 = 49152$ synapses in total). Lateral inhibition between neurons of the output layer is complete, from each neuron to every other neuron, without spatially specific inhibition. When a post-synaptic neuron spikes, its synapses that were the most recently activated are potentiated and its synapses activated slightly after are depressed. Due to lateral inhibition, the integration of other post-synaptic neurons is inhibited.

Parameter	Value	Unit	Description
C_m	70	nF	Capacitance of the LIF neuron
τ_m	299	ms	Membrane time-constant
$\tau_{refract}$	300	ms	Refractory period - membrane potential is clamped to V_{reset}
V_{reset}	-70	mV	Reset potential after spike
V_{rest}	-65	mV	Membrane rest potential
V_{thresh}	-55	mV	Threshold voltage at which spike is emitted
$\tau_{syn,E}$	2	ms	Excitatory input current decay time-constant
$\tau_{syn,I}$	25	ms	Inhibitory input current decay time-constant
i_{offset}	0	nA	Offset current: base input current to add at each time-step

Table 4.3: **Parameter values of the LIF neurons in the output layer of the unsupervised model.**
Note that τ_m cannot be larger than 300ms, the time between two samples in the recordings. If it were, spikes from two trajectories could interfere. Compared to the parameters of the neurons in the smaller neural network model, $\tau_{refract}$ has been increased. If it is too short, a single neuron starts sending bursts of spikes, which prevents other neurons from firing (therefore learning) due to lateral inhibition. It should be approximately as long as the length of the partial trajectories to be learned by one neuron. The initial maximal weight value is set to 2 nA to prevent unrelated neurons from spiking after the training, because the effects of LTD are smaller than those of LTP (i.e. weights increase more than they decrease).

Parameter	Value	Unit	Description
$delay$	1	ms	Synaptic delay
τ_+	5	ms	Long-Term Potentiation time-constant
τ_-	30	ms	Long-Term Depression time-constant
A_+	0.3	nA	Maximum positive weight change
A_-	0.5	nA	Maximum negative weight change
w_{min}	0	nA	Minimum synaptic weight
w_{max}	10	nA	Maximum synaptic weight
$w_{max,i}$	2	nA	Maximum synaptic weight

Table 4.4: **Parameter values of the STDP function in the unsupervised model.**
Due to the limitations of the SpiNNaker hardware, τ_+ and τ_- are limited. τ_- cannot be larger than 30 ms as in the smaller neural network model, thus τ_+ has to be reduced to observe a significant effect of LTD on the learning. Although τ_+ is smaller, it is possible to set an initial maximum weight weaker than the maximum weight of the network. Most weights are not strongly modified but some reaches the maximum weight.

4.4 Unsupervised model with three layers

Once the second layer has been trained, a third layer is implemented to predict the end-position of the ball (Figure 4.4). As in the first two models, the output layer consists of as many neurons as there are end-positions. STDP learning and lateral inhibition are disabled for the middle layer, similarly to the "layer-by-layer" learning strategy of Bichler et al.

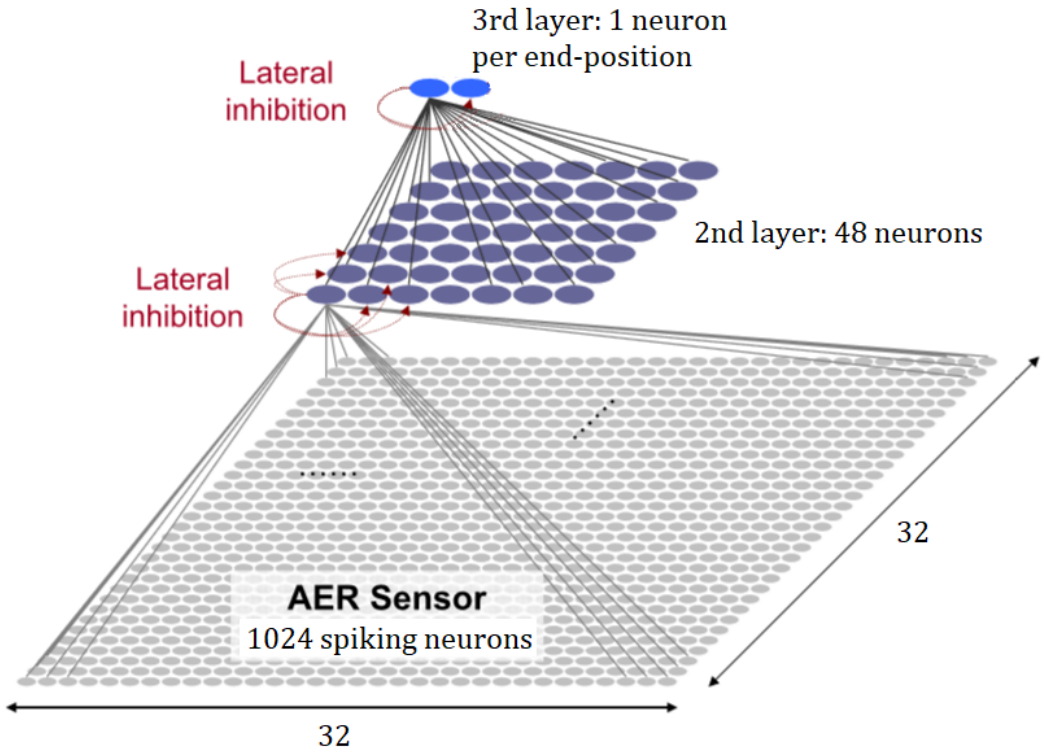


Figure 4.4: **Unsupervised model with three layers.**
Figure design inspired by [28]. The neural network is a feedforward fully-connected network in which each neuron of the middle layer is connected to all neurons of the input layer by plastic synapses, and each neuron of the output layer is connected to all neurons of the middle layer by plastic synapses ($1024 * 48 + 48 * 2 = 49248$ synapses in total). Neurons within a given layer share the same LIF parameters and lateral inhibition between these neurons is complete.

The spikes of the middle layer in response to the AER data spikes are passed as input for the output layer, and as there are few spikes per trajectory (between 9 and 14, see Figure 5.11 in Results chapter), the simulation can be sped up to a factor 10 without losing any spikes. Not only does this acceleration allow the simulations to be faster, but also restrictions on the LTP and LTD time-constants are not (as) problematic anymore and synaptic weights can converge properly, unlike during the training of the simple unsupervised model with no middle layer and small output layer.

In the final three-layer model, both the middle and output layers consist of LIF neurons, however their parameters are different (but the same within a layer). Neuron parameters of the middle layer remain unchanged as compared to previous neural network in the *Unsupervised model with larger second layer* (Table 4.3), and neurons of the output layer share the parameters summarized in Table 4.5. STDP parameters between the middle and output layers are summarized in Table 4.6.

Note that the LIF parameters of the output neurons need to be modified when running tests: as the whole network is tested, the simulation cannot be sped up. Therefore all parameters that are time-related are multiplied by 10 (τ_m, $\tau_{refract}$, $\tau_{syn,E}$ and $\tau_{syn,I}$), and so is C_m to maintain the same membrane resistance R_m.

Parameter	Value	Unit	Description
C_m	3	nF	Capacitance of the LIF neuron
τ_m	5	ms	Membrane time-constant
$\tau_{refract}$	5	ms	Refractory period - membrane potential is clamped to V_{reset}
V_{reset}	-70	mV	Reset potential after spike
V_{rest}	-65	mV	Membrane rest potential
V_{thresh}	-62	mV	Threshold voltage at which spike is emitted
$\tau_{syn,E}$	10	ms	Excitatory input current decay time-constant
$\tau_{syn,I}$	10	ms	Inhibitory input current decay time-constant
i_{offset}	0	nA	Offset current: base input current to add at each time-step

Table 4.5: **Parameter values of the LIF neurons in the output layer of the three-layer model.**
As the time is sped up by a factor 10, the time-constants τ_m and $\tau_{refract}$ are decreased. Now that output neurons can receive inputs from at most 48 neurons (instead of 1024), their membrane potential varies less and the threshold must be decreased.

Parameter	Value	Unit	Description
delay	1	ms	Synaptic delay
τ_+	5	ms	Long-Term Potentiation time-constant
τ_-	30	ms	Long-Term Depression time-constant
A_+	0.5	nA	Maximum positive weight change
A_-	0.5	nA	Maximum negative weight change
w_{min}	0	nA	Minimum synaptic weight
w_{max}	3	nA	Maximum synaptic weight

Table 4.6: **Parameter values of the STDP function in the three-layer model.**
The maximum weight is decreased so that it corresponds to the difference between V_{rest} and $V_{threshold}$, as in the STDP function between the input and middle layer (Table 4.4).

Chapter 5

Results

5.1 Simple supervised model

Individual trajectory recognition

Four out of the 15 trajectories recorded (four out of the 45 recorded files) are used to train the neural network, starting from positions 2 and 4 and ending in positions 1 and 3 (trajectories known as 2-1, 2-3, 4-1, 4-3). Each trajectory is first trained individually, meaning that only one trajectory is presented to the neural network 18 times to observe the effect of the STDP learning as function of time (iteration after iteration). Teaching neurons emitting a supervisory signal are used to direct the learning towards the expected firing pattern.

Before the training (Figure 5.1), output neurons are not selective and respond in a similar way to the input. Remember that initial weights come from a random uniform distribution thus this is expected. During the training, the external stimulation drives the learning towards the desired output: the weights are gradually updated, strengthening synapses with neuron corresponding to end-position 1 and weakening synapses with neuron corresponding to end-position 3. The network learns to recognize one trajectory in less than 10 presentations (Figure 5.2).

After the training, the same trajectory is presented again to the two-layer network, with no supervisory signal, STDP, nor lateral inhibition. As expected, only the output neuron which has learned the trajectory fires (Figure 5.3): the neural network is capable of recognizing individual trajectories, even with other records of the same trajectory (Figure 5.4).

Figures are shown for the training using the third recording of the trajectory starting at position 2 and ending at position 1 (known as trajectory 2-1). As Python indexes begin at 0, the output neuron index does not correspond to the end-position index. With two end-positions, neuron 0 corresponds to end-position 1 while neuron 1 corresponds to end-position 3.

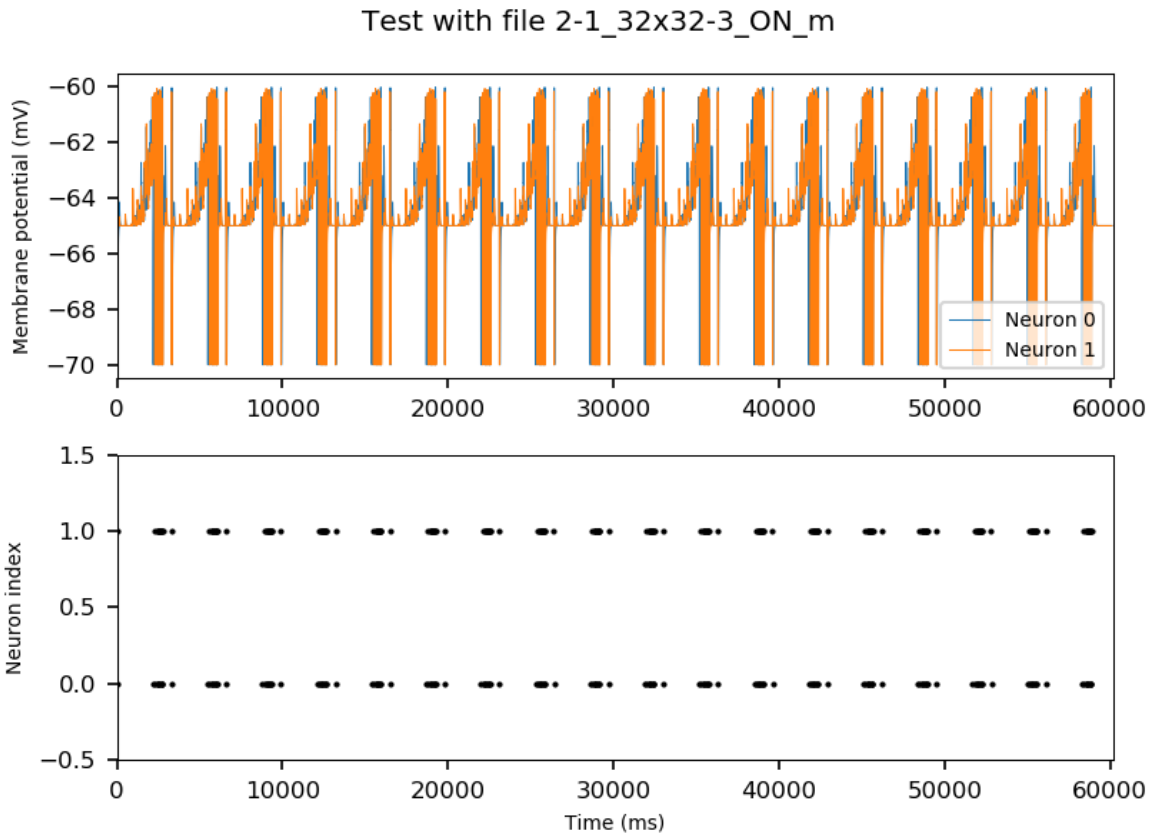


Figure 5.1: **Membrane potential and raster plot of the output layer, before training and upon presentation of trajectory 2-1.**
Recording 3 of trajectory 2-1 is presented 18 times. As this is a test, there is no learning (no STDP), no supervisory signal, and no lateral inhibition. We can see that none of the output neurons is selective to the trajectory presented and firing does not change at each presentation of the same trajectory, showing that indeed no learning occurs. Note that with the code used, SpiNNaker cannot take files of more than around 1 minute (60000 ms), otherwise packets are lost. We can thus present the 3-second trajectories at most 18 times with 300 ms between each sample ($18 * 3 + 17 * 0.3 = 59.1s$).

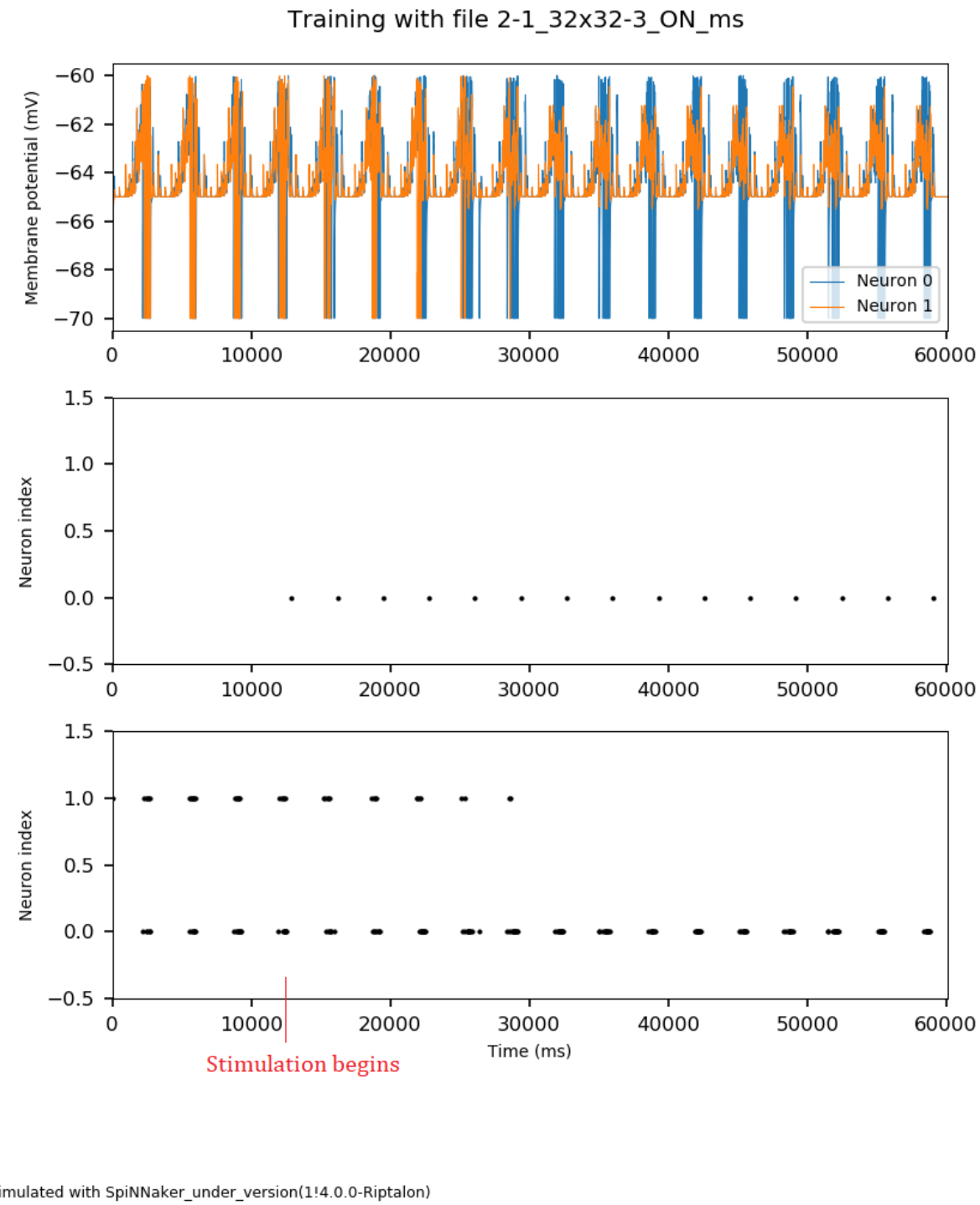


Figure 5.2: **Membrane potential of the output layer and raster plots of the stimulation and output layers, during training with trajectory 2-1.**
During the first three presentations of the trajectory, no supervisory signal (middle subplot) drives the learning to show the original spiking pattern before learning. The output is close to the raster plot of Figure 5.1 as neurons are not selective, except this time lateral inhibition is enabled. At the fourth presentation of trajectory 2-1, stimulation by the teaching neuron 0 is added to supervise the learning towards the output neuron 0 (corresponding to end-position 1). At each time-step, the weights between the input and output layers are updated. The spiking activity of neuron 1 in the output layer (corresponding to end-position 3) decreases progressively until it completely stops.

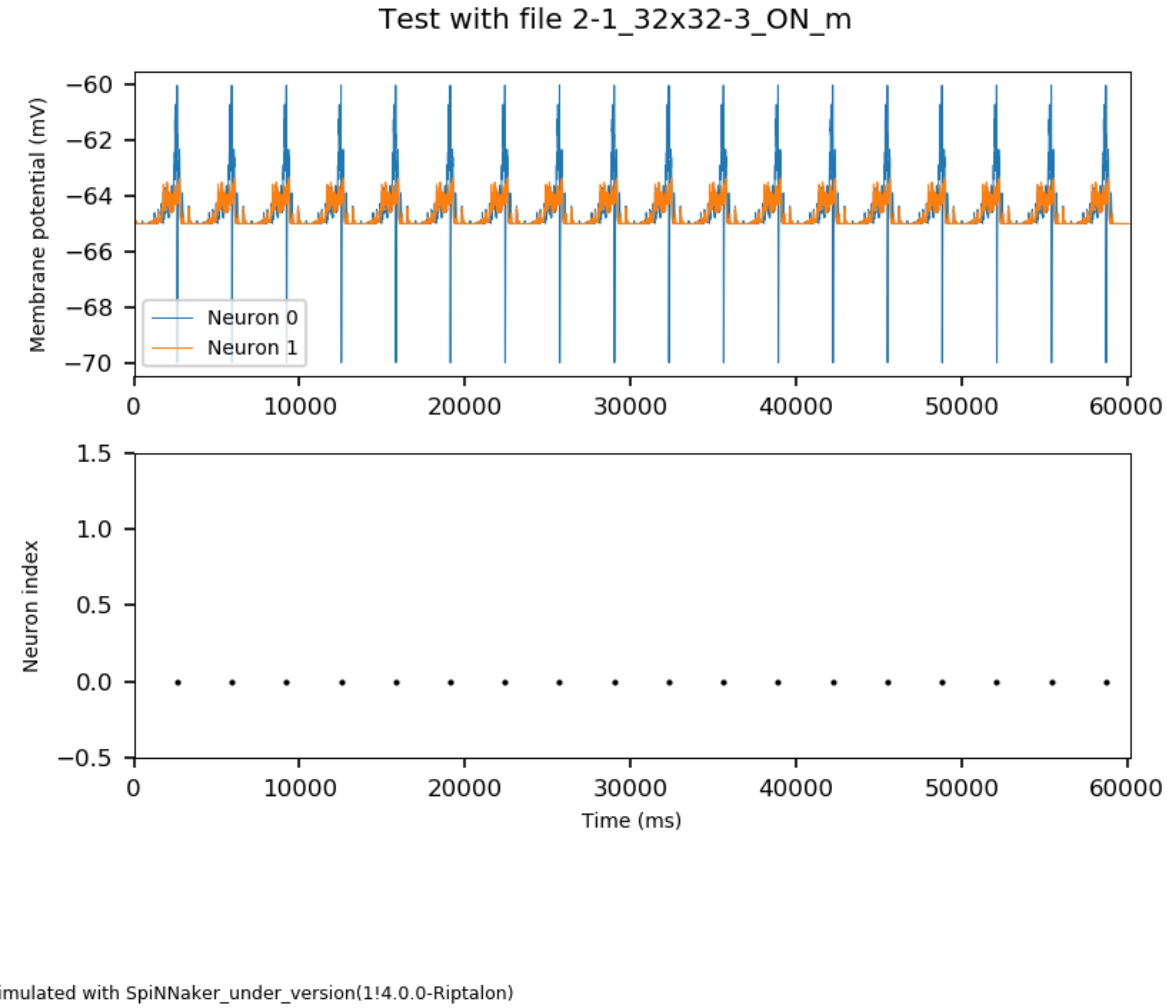


Figure 5.3: **Membrane potential and raster plot of the output layer, after training with trajectory 2-1 and upon presentation of trajectory 2-1 (same recording).** To check that the trajectory has been learned, it is presented again (18 times) to the network in which the supervisory signal, STDP learning and lateral inhibition are removed. The synaptic weights between the input and output layers are the final weights of the training. As observed on the figure, neuron 0 spikes at the end of each trajectory presentation, meaning that learning was effective and the neural network is able to recognize this specific trajectory.

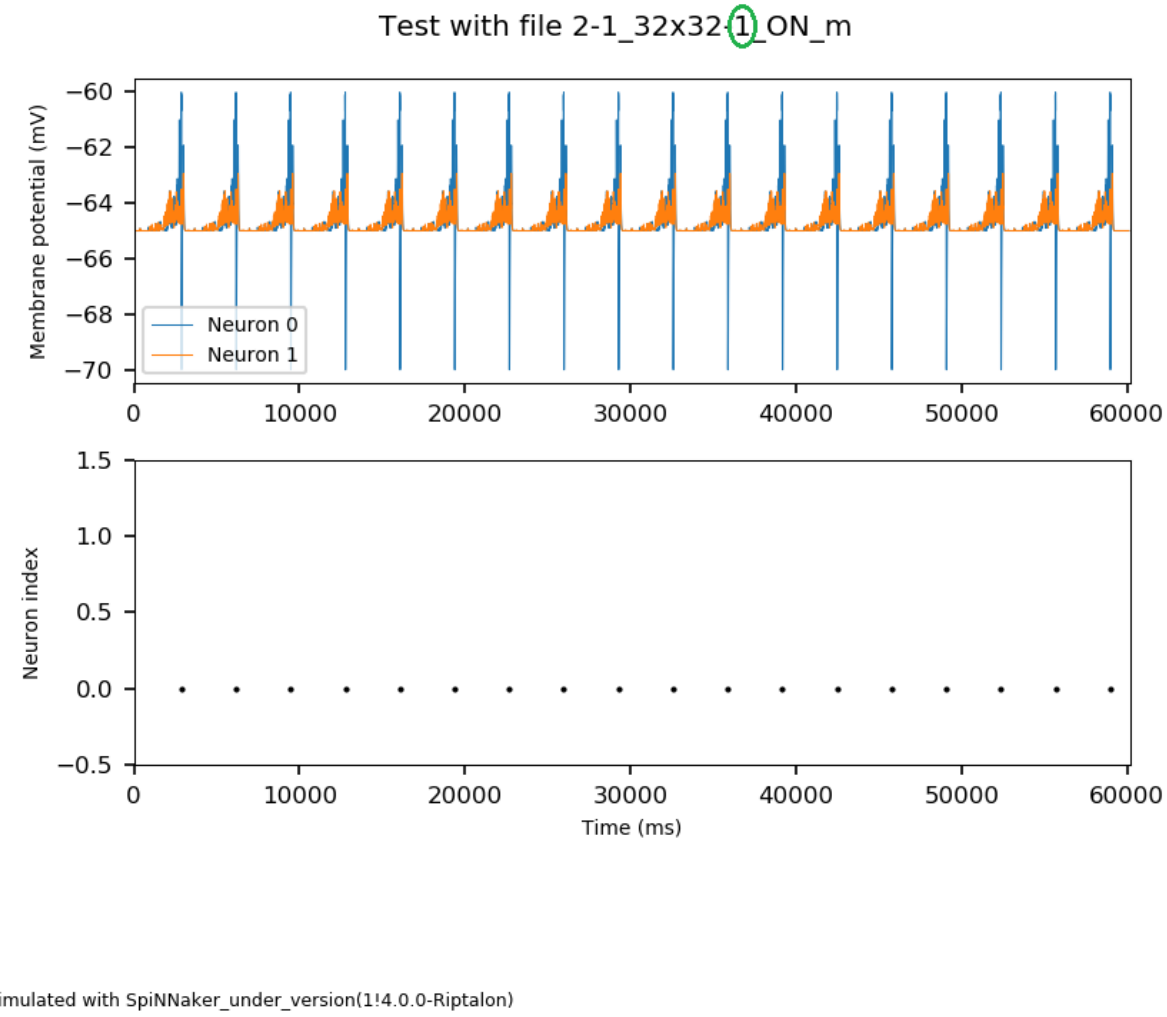


Figure 5.4: **Membrane potential and raster plot of the output layer, after training with trajectory 2-1 and upon presentation of a new recording of trajectory 2-1.** Same as in Figure 5.3 except that the recording 1 of trajectory 2-1 is used. This figure shows that learning is still effective although the recording is slightly different. This proves that the neural network is robust to some variance in the data.

Learning of multiple trajectories

The next step after seeing that the neural network is able to learn each trajectory separately is to find out if it is able to recognize several trajectories together. To do so, the four trajectories are each presented 45 times during the training. Training is done with a stimulation as previously (Figure 5.2), which is again turned off for the tests, as are STDP learning and lateral inhibition. Since lateral inhibition is disabled during the tests, the threshold voltage of the membrane potential V_{thresh} is increased to -50.

Before the training, the two neurons of the output layer respond to the input in a similar way, meaning that they are not selective (Figure 5.5). After the training, the neural network is able to discriminate the end-positions correctly (Figure 5.6). In the next tests with shifted and partial trajectories, the trained model used is the one presented in this subsection.

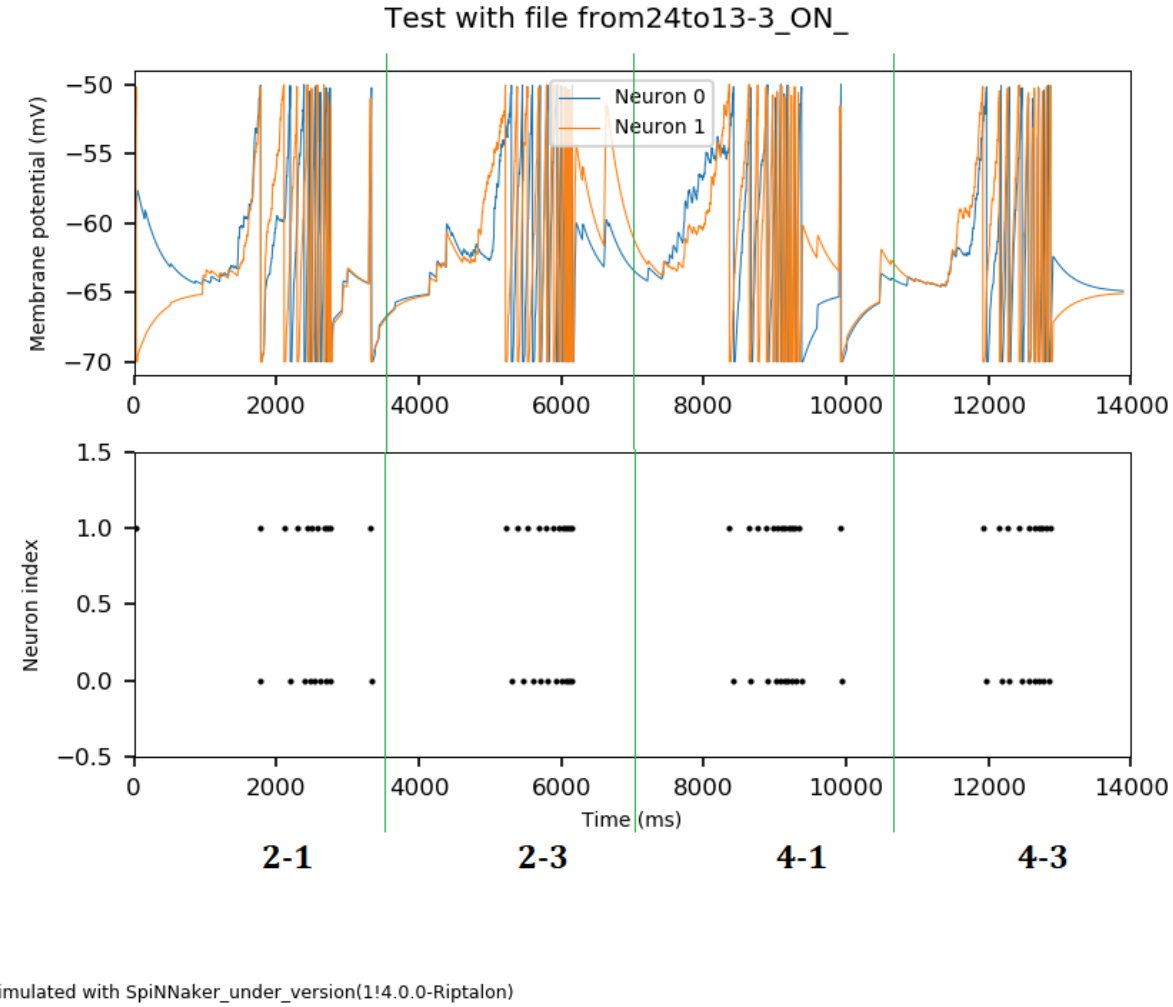


Figure 5.5: **Membrane potential and raster plot of the output layer, before training, and upon presentation of the four trajectories later used for training.**
The four trajectories are presented to the untrained network (in the order 2-1, 2-3, 4-1, 4-3). The two neurons of the output layer respond to the input in a similar way, meaning that they are not selective.

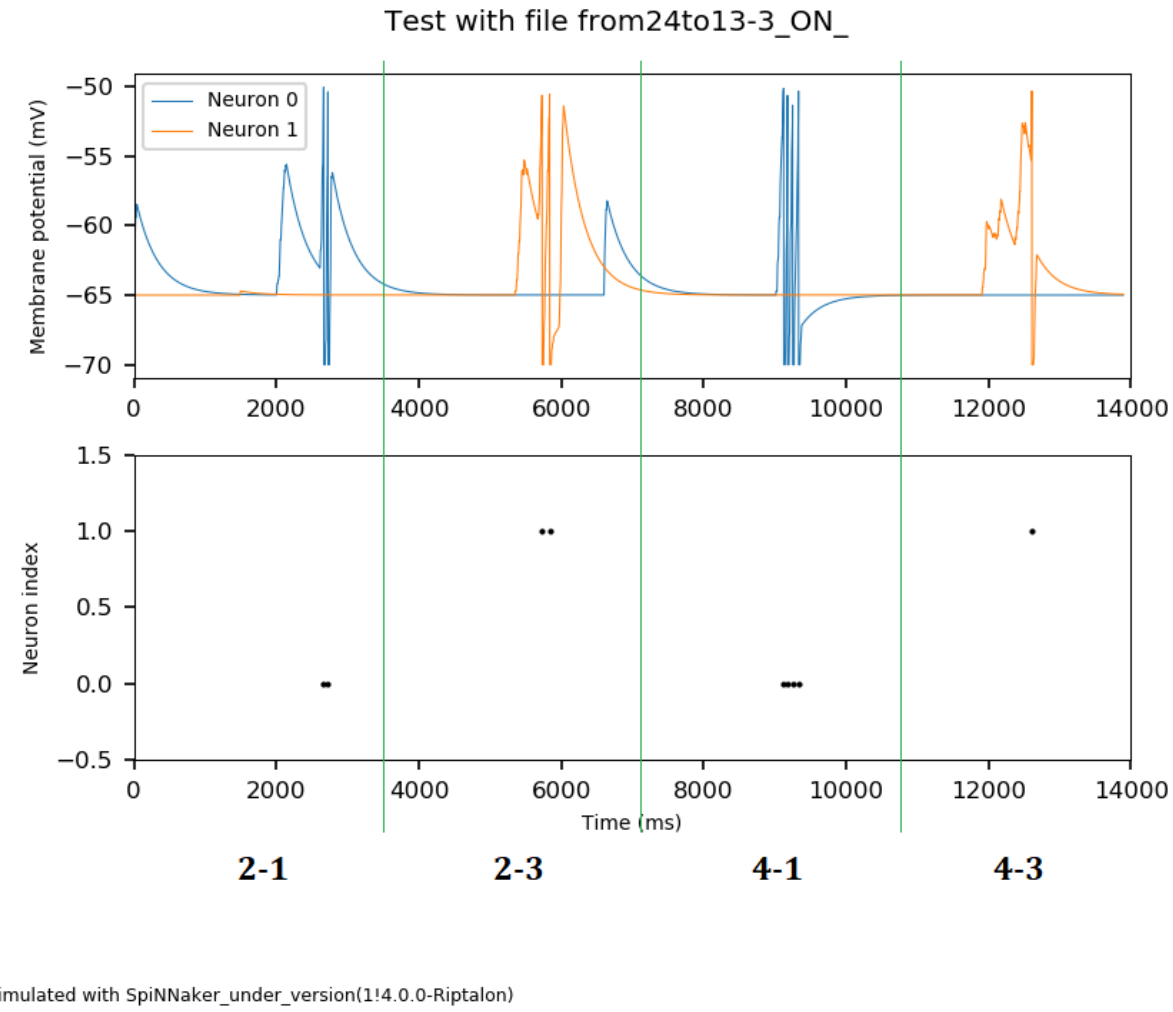


Figure 5.6: **Membrane potential and raster plot of the output layer, after training with four trajectories and upon presentation of the four trajectories used for training.**
The neural network is trained by presenting each of the four trajectories around 45 times. After the training, the four trajectories are presented to the trained network which is able to recognize the correct end-positions for all four trajectories.

Tests with unknown trajectories

Tests are performed on the neural network trained with four trajectories using the 45 trajectories recorded. These include other recordings of the same trajectories as used for the training as well as trajectories with shifted start-positions (start-positions 1, 3, and 5) and trajectories finishing at end-position 2 (Figure 5.7).

The neural network is able to recognize all five trajectories ending in position 1 and three of the five trajectories ending in position 3 (80% accuracy). For trajectories 1-3 and 5-3, the membrane potential of the corresponding output neuron increases but does not reach the threshold. The neural network being able to recognize end-positions in trajectories that were not used for the training shows that it is not overtrained and does not only recognize these specific trajectories.

When trajectories finishing in end-position 2 are presented to the network, in four cases none of the output neurons fire (as expected), but trajectory 5-2 is misclassified and recognized as ending in position 3 (neuron 1 fires). This probably comes from the fact that trajectories 4-3 and 5-2 cross each other and share part of the trajectory.

Since the ball appears larger when it approaches the end-positions, more input neurons spike at the end of a trajectory and this firing is similar across trajectories with same end-positions. The aim is not that the network recognizes the end-position of the ball when it is already reached, but to recognize it in advance. Thus, to see if the network can really adapt and predict end-positions, next tests are performed using only parts of the trajectories.

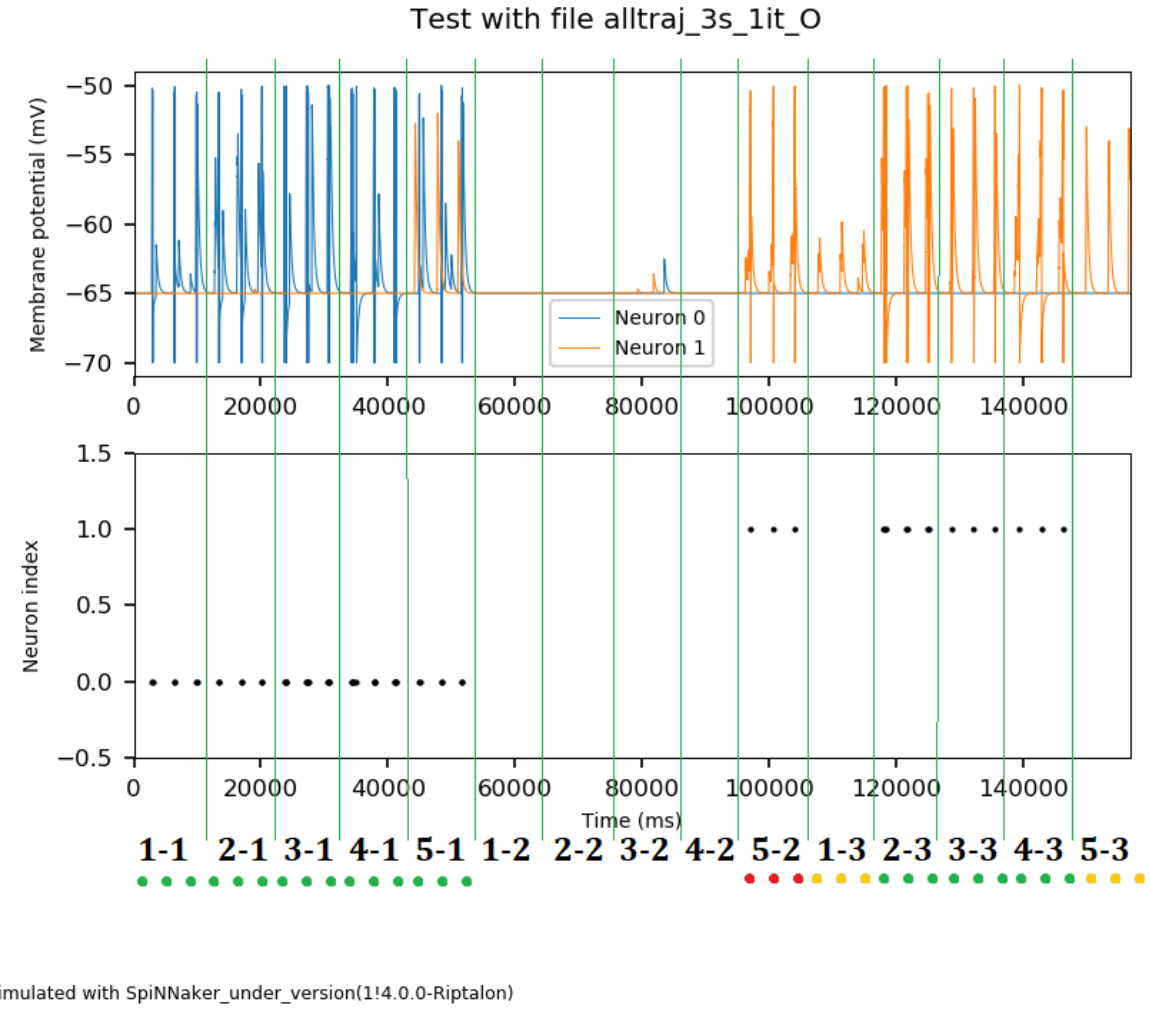


Figure 5.7: **Membrane potential and raster plot of the output layer upon presentation of all recorded trajectories.**
The neural network is able to recognize the end-position in all trajectories finishing at end-position 1, no matter the start-position, although training is done only with trajectories starting in positions 2 and 4. Recognition of end-position 3 is not as good as for end-position 1. The color dots below the trajectory ID represents the correctness of the output: green means that the expected output neuron fired, orange corresponds to a missing output, and red that the wrong neuron fired (or in this case, that a neuron fired whereas it should not). As the network is not trained to recognize end-position 2, when none of the output neurons fires upon presentation of trajectories ending in this position, it could be considered as a correct response of the network.

Tests with partial trajectories

Finally, tests are performed using the first half (1500 ms) of the 30 recordings of trajectories from start-positions 1 to 5 to end-positions 1 and 3 (three recordings each). Synaptic weights used are again the weights resulting from the training with four trajectories.

Out of the 30 trajectories presented, none of the trajectories are misclassified, meaning that the wrong output neuron never fires. 24 recorded trajectories are correctly predicted (80% accuracy), including all trajectories starting at positions 1 to 4. Upon presentation of the three trajectories 5-3, no output neuron fires. This might come from the fact that the synaptic weights of input neurons corresponding to the extreme of the field of view have not been strengthened. Upon presentation of the trajectories 5-1, both output neurons spike. Neuron 1 (corresponding to end-position 3) probably fires because the first half of trajectory 5-1 takes place in the right side of the visual field of the DVS camera, where start-positions 4 and 5 and end-position 3 are located.

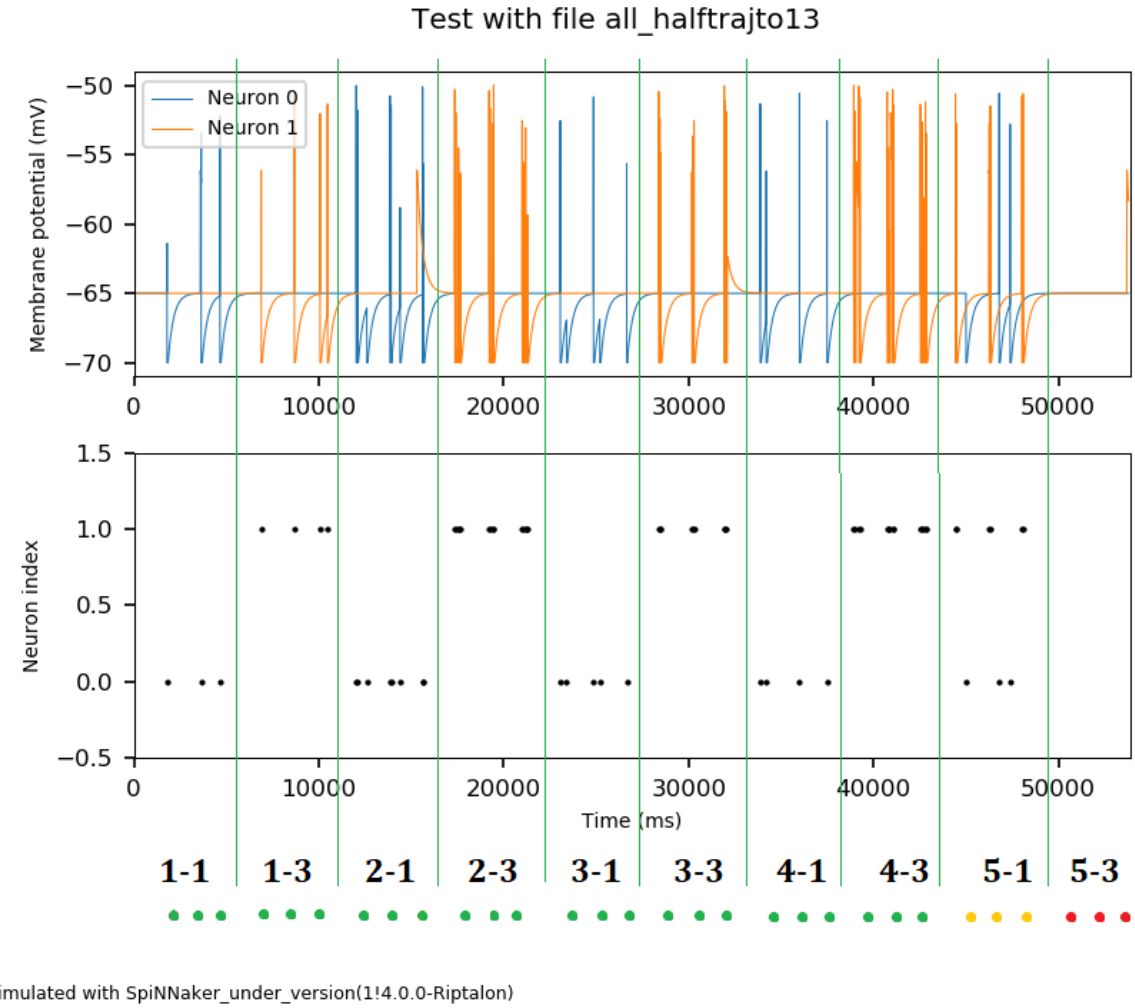


Figure 5.8: **Membrane potential and raster plot of the output layer, upon presentation of half-trajectories.**
After training the neural network with four whole trajectories, half of each of the 30 recorded trajectories (ending in positions 1 and 3) are presented to the network. The color points below the trajectory ID represents the correctness of the output: green means that the expected output neuron fired, orange that both output neurons fired, and red that no neuron fired (notice that the wrong neuron never fires alone). 24 trajectories are correctly predicted. Three trajectories do not trigger any spike, and three are not correct because both neurons spike. These six trajectories that are not correctly identified all correspond to a start-position shifted at position 5.

5.2 Simple unsupervised model

The neural network of the simple unsupervised model is basically the same neural network as in the simple supervised model, except the stimulating population is removed. Without the help of the teaching neurons, the output neurons are not able to learn to recognize different trajectories properly (Figure 5.9, same parameters as in the supervised model). They have no indication on which pattern they should be learning, and the STDP weight update is somewhat random. As the trajectory length is much longer than the STDP learning window, the synaptic weights do not converge in the expected manner. A large range of different combinations of STDP parameters was tried (one example in Figure 5.10, with $\tau_+ = 5$ ms and $w_{max,i} = 2$ nA), none of which has been proven to provide efficient learning. This shows that one neuron is not capable of learning a complete trajectory, thus the next step is to increase the output layer size in order to allow multiple neurons to learn different part of a trajectory.

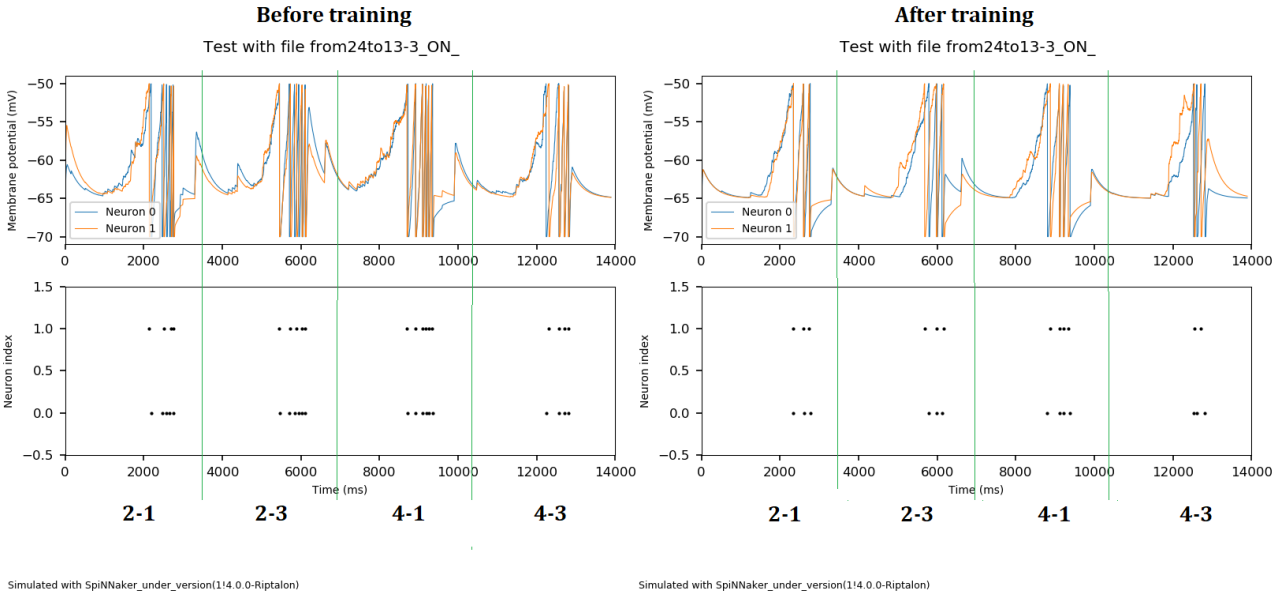


Figure 5.9: **Membrane potential and raster plot of the output layer of the simple unsupervised model.**
After training the neural network with four different trajectories (same as previously: 2-1, 2-3, 4-1, and 4-3, presented around 45 times each) but without any supervisory signal, the network is not able to recognize the end-positions upon new presentation of these trajectories (tests with STDP and lateral inhibition disabled). The effect of LTD is not strong enough and uncorrelated synapses do not get depressed enough, even when the initial weights and LTP time-constant τ_+ are reduced (see Figure 5.10).

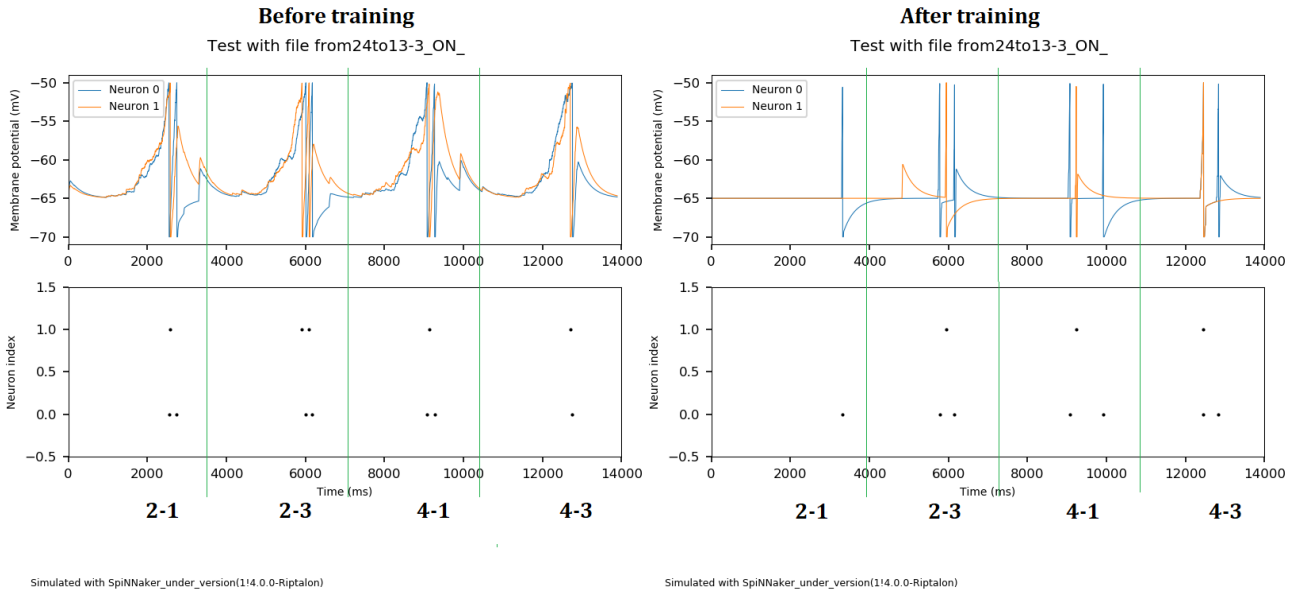


Figure 5.10: **Membrane potential and raster plot of the output layer of the simple unsupervised model with decreased initial weights and LTP time-constant.**
After training the neural network with four different trajectories, no supervisory signal, initial weights limited to $w_{max,i} = 2$ nA and LTP time-constant equal to $\tau_+ = 5$ ms (instead of 10 nA and 30 ms respectively), the network is not able to recognize the end-positions upon new presentation of these trajectories (tests with STDP and lateral inhibition disabled). The effect of LTD is not strong enough.

5.3 Unsupervised model with larger second layer

When the size of the second layer is increased and the neural network trained with the same four trajectories as previously, different patterns are observed for different trajectories, with six or seven neurons being specific to each end-position (Figure 5.11). The output layer now consists of 48 neurons, of which 29 fire at least for one trajectory after training the neural network. Before the training, all 48 neurons spike for all trajectories, up to twice, especially for trajectories 2-3 and 4-1. These are diagonal (so longer), therefore neurons are expected to have a stronger activity. After the training, some neurons get selective and spike for trajectories with the same end-position (six neurons for end-position 1 and seven neurons for end-position 3). There remain neurons that are specific to a single trajectory and spiking only when this particular trajectory is presented to the network. There are also several neurons spiking for trajectories with different end-positions, which is not surprising as trajectories are not completely uncorrelated (they can have the same start-position or cross each other). There also remains one neuron spiking for all four trajectories.

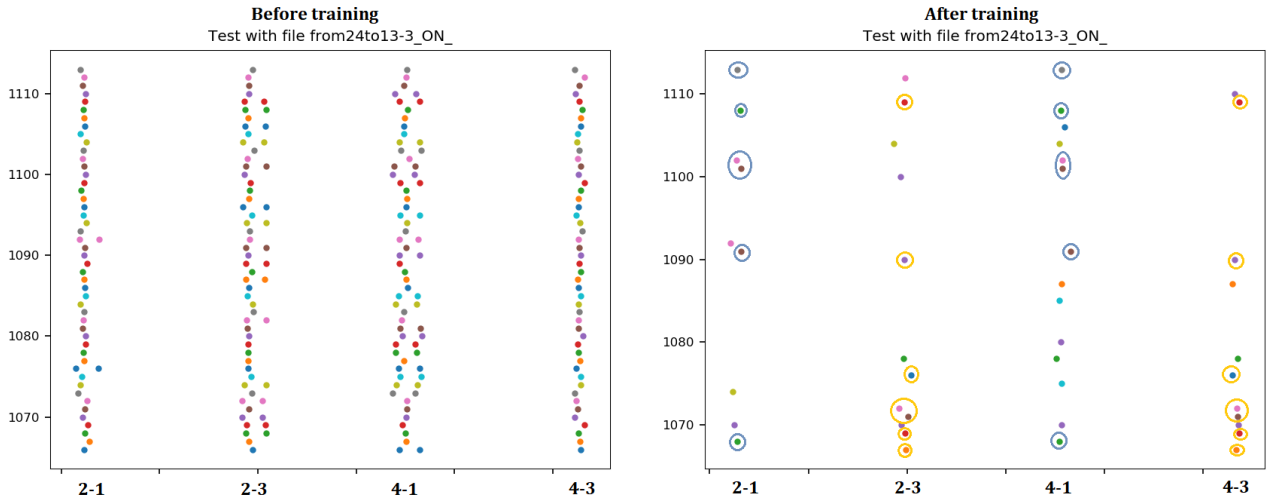


Figure 5.11: **Raster plot of the second layer of the unsupervised model with larger second layer.**
This figure is plotted with *matplotlib.pyplot* instead of *pyNN.utility.plotting* to show different neurons in different colors in order to better distinguish neurons common to several trajectories. The x-axis represents the time, each trajectory being presented one after the other (same order as other test figures: 2-1, 2-3, 4-1, and 4-3). The y-axis shows the 48 neurons, with neuron 1066 of the figure corresponding to output neuron 0 and neuron 1113 to output neuron 47. There are six neurons spiking both for trajectories 2-1 and 4-1 (blue circles) and seven neurons spiking both for trajectories 2-3 and 4-3 (yellow circles). These neurons can be considered as specific to their respective end-position. Additionally, one neuron is not selective at all and spikes for all four trajectories (neuron 1070 on the figure). Also, for each trajectory, there are between one and four neurons spiking only for this given trajectory, and between one and four neurons spiking for at least another trajectory with different end-position.

5.4 Unsupervised model with three layers

Once the 48-neuron layer has shown to respond similarly to trajectories with the same end-positions, a third layer is added to the network and trained in turn with STDP (as in the "layer-by-layer" strategy of Bichler et al.). In the network with a trained middle layer but untrained output layer, both output neurons fire in a similar pattern in response to the four trajectories (Figure 5.12). After the training of the output layer of the neural network, only the neuron corresponding to the end-position of the trajectory presented fires (Figure 5.13).

To see if the network is able to recognize end-positions of trajectories beginning at different start-positions and how it reacts for trajectories finishing at a different end-position, all 45 recorded trajectories are presented to the network (Figure 5.14). Remember that these 45 recordings correspond to 15 different trajectories (three recordings each), starting at five distinct start-positions and ending at three distinct end-positions. If we do not take into account trajectories finishing at end-position 2 (the neural network was not trained for it), 25/30 trajectories are correctly identified (83% accuracy). For the other five trajectories, both neurons fire; if lateral inhibition is maintained during the test, this is not the case and 100% of the end-positions are correctly identified (see Figure 5.15). We can also notice that the incorrect output neuron never fires alone, and that no neuron fires for end-position 2, which can be considered as a correct response.

Finally, first half of the trajectories finishing in end-positions 1 and 3 are presented to the network to evaluate its ability to predict and not only recognize end-positions. Overall, 16 out of the 30 trajectories presented correctly predicts the end-position (53% accuracy). Two trajectories trigger spikes in both output neurons and for the last 12 trajectories, signal is not strong enough to cause any spike, although we can see that for eights of those 12 trajectories, the membrane potential rises but does not reach the threshold (if the threshold was decreased, accuracy could be raised to 80%!).

Figures 5.17 and 5.18 show the heatmap of the weights of the output neurons respectively before and after the training of both layers. It is reconstructed from the weighted sum of synapses of the middle layer with the corresponding weights for the neuron of the second layer. Before the training, weights are random and uniformly distributed. After the complete training, weights of each neuron are stronger near the corresponding end-position of the ball. Also, for neuron 1 corresponding to end-position 3, synaptic weights are stronger on average and more weights are not equal to zero. This might explain why position 3 is better recognized and why sometimes neuron 1 fires for trajectories ending in position 1 although it should not. Also notice that more than 95% of the weights are equal to zero after the complete training, because most of them are set to zero as they do not undergo depression due to the short LTD time-constant. Thus, only around 30 non-zero weights are sufficient for the end-position to be recognized.

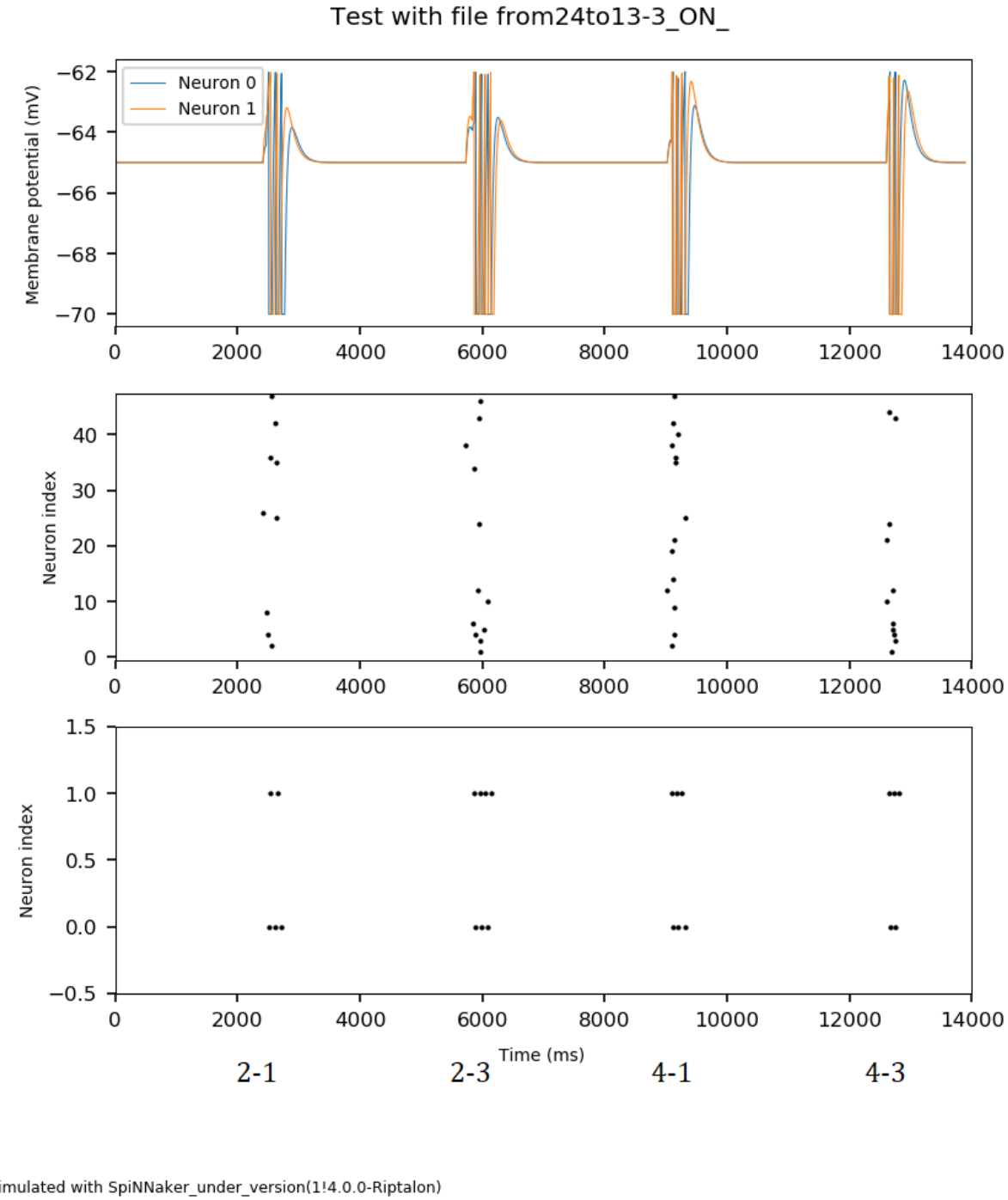


Figure 5.12: **Membrane potential of output layer and raster plots of middle and output layers of the three-layer unsupervised model, before training of the third layer.**
The four trajectories 2-1, 2-3, 4-1, and 4-3 are presented to the neural network after the training of the middle layer but before the training of the output layer. The two neurons of the output layer respond to the input in a similar way, meaning that they are not selective.

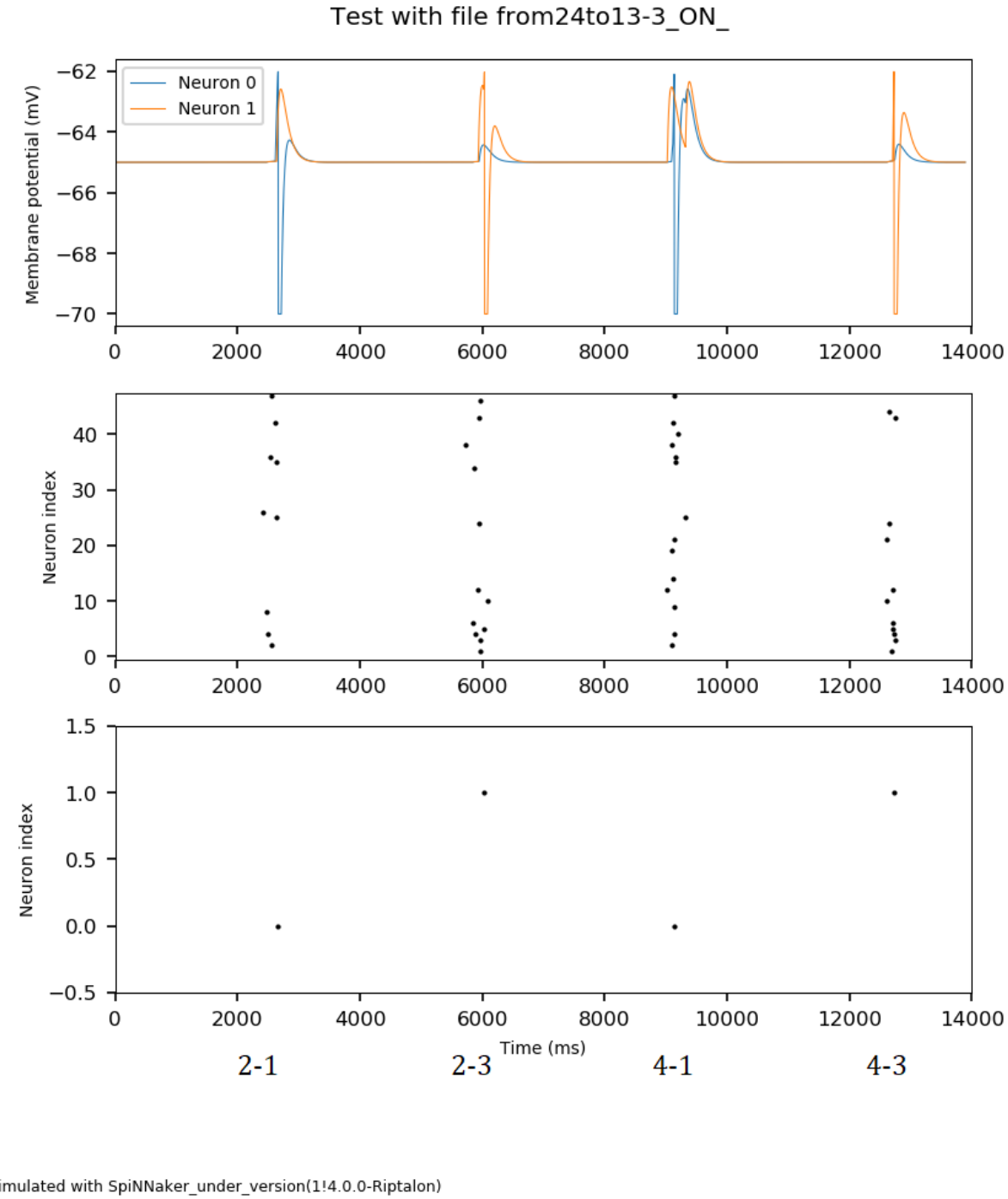


Figure 5.13: **Membrane potential of output layer and raster plots of middle and output layers of the three-layer unsupervised model, after complete training.** When both layer have been trained, the four trajectories used for training are presented to the trained network, which is able to recognize the correct end-positions for all four trajectories.

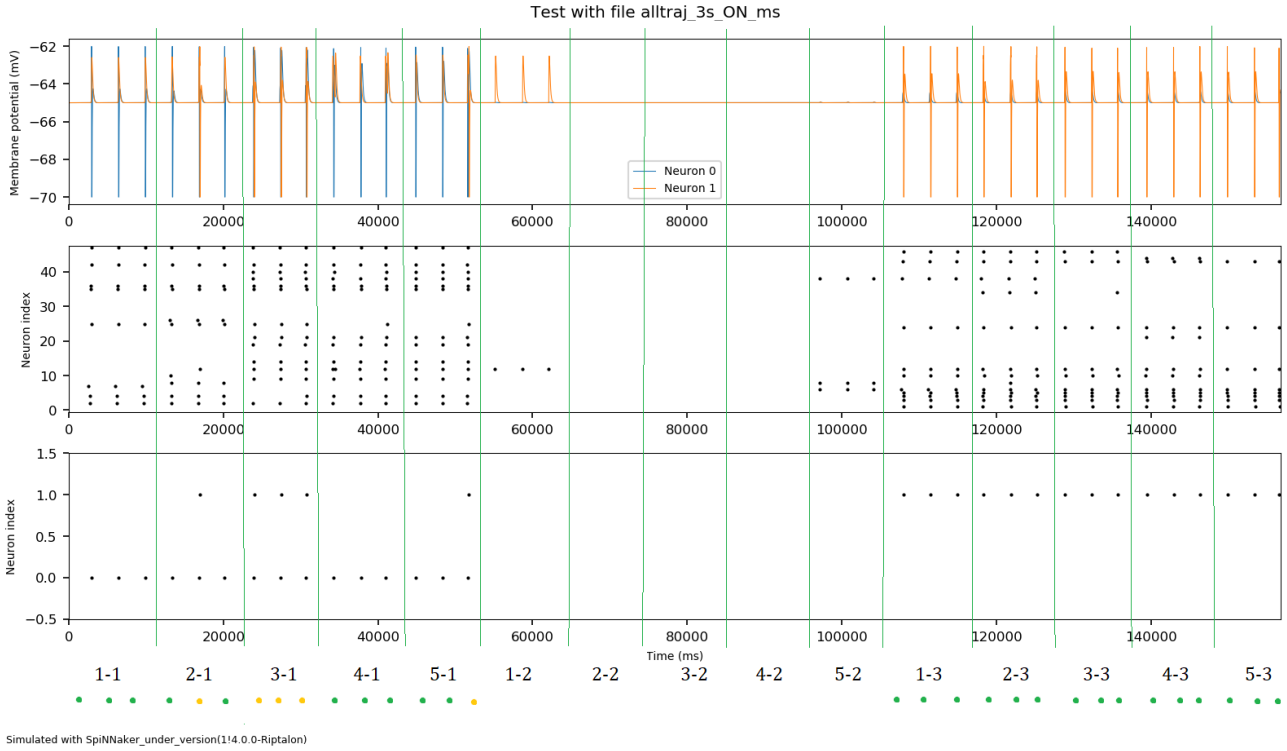


Figure 5.14: **Membrane potential of output layer and raster plots of middle and output layers of the three-layer unsupervised model, after complete training with four trajectories and upon presentation of the 45 recorded trajectories.**
After training the neural network with four trajectories (four recordings), all 45 files recorded for the project are presented to the network. These represent 15 different trajectories recorded three times each, from five start-positions to three end-positions. The color points below the trajectory ID represents the correctness of the output: green means that the expected output neuron fired, orange that both output neurons fired, and red that the wrong neuron fired (not the case). As the network is not trained to recognize end-position 2, none of the output neurons fires upon presentation of trajectories ending in this position, which could be considered as a correct response of the network.

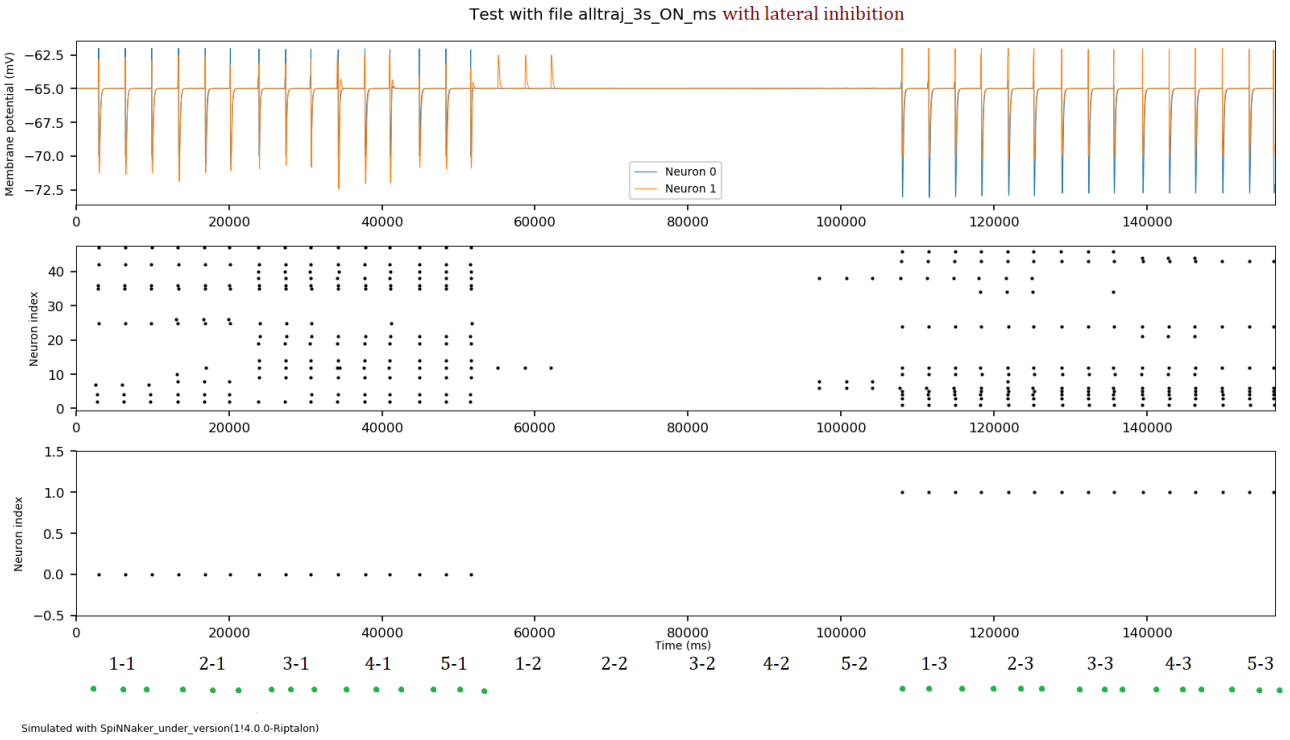


Figure 5.15: **Membrane potential of output layer and raster plots of middle and output layers of the three-layer unsupervised model, after complete training with four trajectories and upon presentation of the 45 recorded trajectories, while keeping lateral inhibition enabled.**
After training the neural network with four trajectories (four files), all 45 files recorded at the beginning of the project are presented to the network. These represent 15 different trajectories recorded three times each, from five start-positions to three end-positions. The color points below the trajectory ID represents the correctness of the output: green means that the expected output neuron fired, orange that both output neurons fired (not the case), and red that the wrong neuron fired (not the case). As the network is not trained to recognize end-position 2, none of the output neurons fires upon presentation of trajectories ending in this position, which could be considered as a correct response of the network. In this test in which lateral inhibition is not disabled, all correct end-positions are recognized by the neural network.

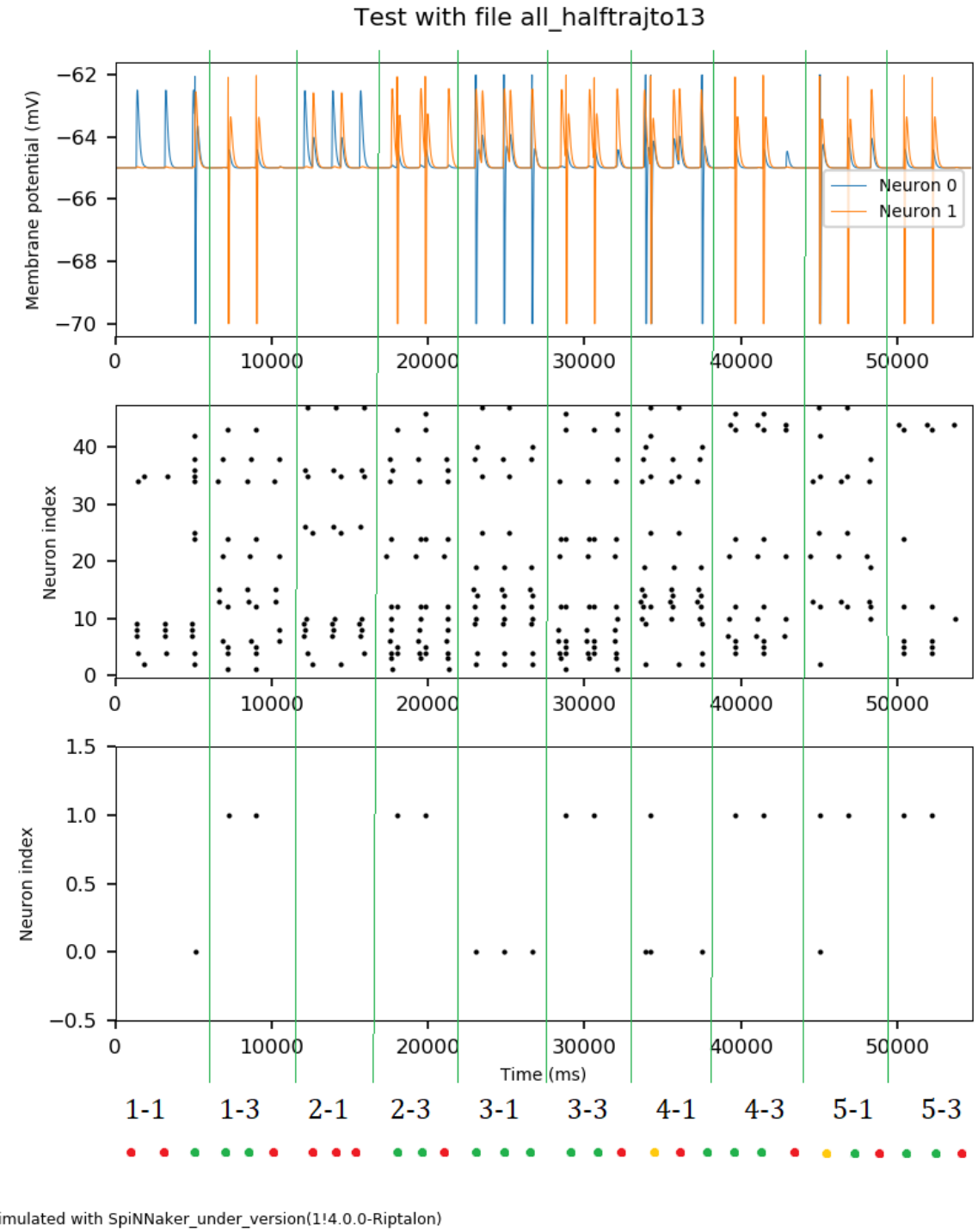


Figure 5.16: **Membrane potential of output layer and raster plots of middle and output layers of the three-layer unsupervised model, after complete training with four trajectories and upon presentation of half-trajectories.**
After training the neural network with four trajectories (four files), the 30 files corresponding to trajectories ending in positions 1 and 3 are presented. The color points below the trajectory ID represents the correctness of the output: green means that the expected output neuron fired, orange that both output neurons fired, and red that no neuron fired. The neural network is not completely able to recognize half-trajectories.

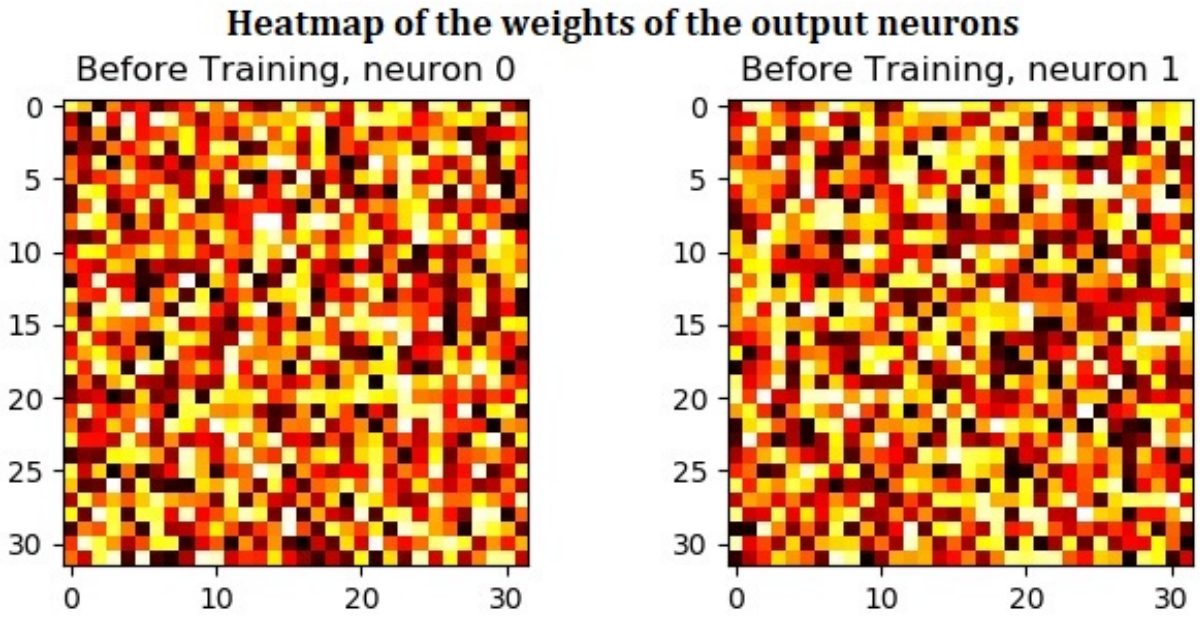


Figure 5.17: **Heatmap of the weights of the output neurons before training.** Before the training of the two layers, the heatmap of the weights of output neurons is reconstructed from the weighted sum of synapses of the middle layer with the corresponding weights for the neuron of the second layer; all weights initially random. For both neurons, the resulting weights are random and uniformly distributed.

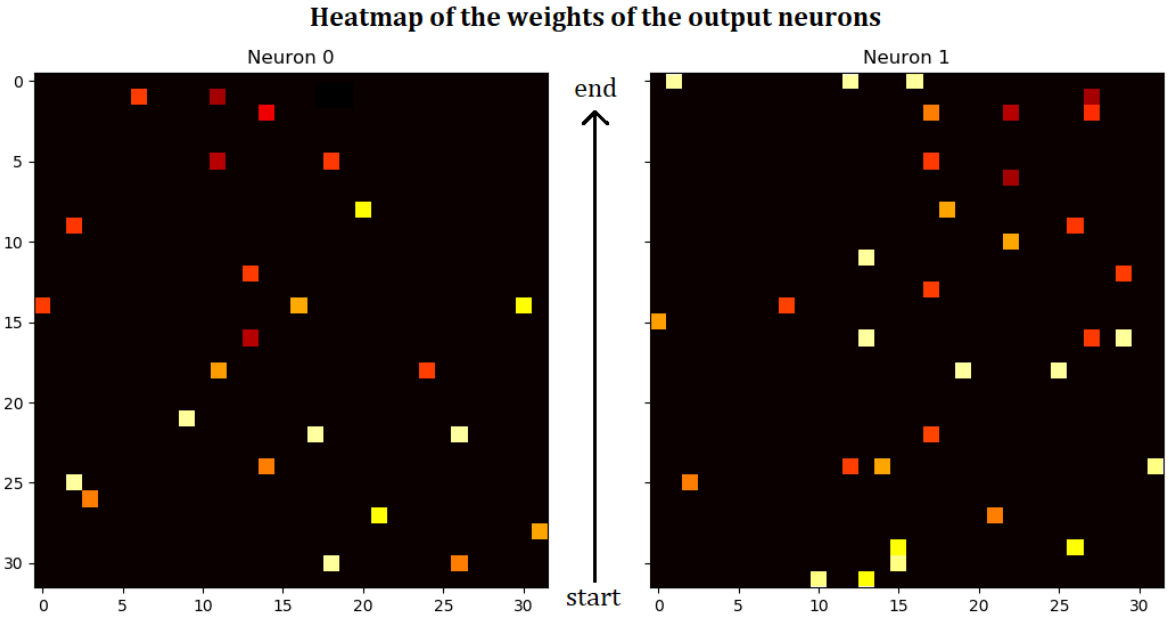


Figure 5.18: **Heatmap of the weights of the output neurons after training.** After the training of the two layers, the heatmap of the weights of output neurons is reconstructed from the weighted sum of synapses of the middle layer with the corresponding weights for the neuron of the second layer. For both neurons, weights are stronger (red) near the corresponding end-position of the ball. Also, for neuron 1 corresponding to end-position 3, weights are stronger on average and more weights are non-zero (not black). This might explain why end-position 3 is better recognized and why sometimes neuron 1 fires for trajectories ending in position 1.

Chapter 6

Discussion

In this project, implementing a neural network able to learn and recognize end-positions of different trajectories of a moving ball was achieved, with Spike-Timing Dependent Plasticity (STDP) learning and Leaky-Integrate-and-Fire (LIF) neurons. Using a supervised model - in which trajectories with labeled end-positions are fed into the network for training with a supervisory signal from teaching neurons - proved to be very efficient at recognizing end-positions (80% accuracy) as well as predicting them from partial trajectories (80% accuracy), including test trajectories for which the start-positions were shifted compared to the trajectories used for training.

As the final objective is to build a robot goalie able to learn from live data, an unsupervised algorithm with a three-layer neural network was also implemented and yielded good results too. It is slightly better than the *Simple supervised model* at the recognition of end-positions (83% accuracy), and is able to predict the end-positions decently (53% accuracy). Although 53% might not seem exceptional, the prediction task is not a classification task, and random weights (untrained weights) yield 0% correct response. Thus, this result remains satisfying as it is far from random. Finally, the best professional goalkeeper of 2017-2018 English Premier League, David de Gea (Manchester United and Spain national team), managed to stop 81% shots on goal (116 shots out of 144) so aiming for an accuracy of 80% for a robot goalkeeper is a reasonable objective.

	Supervised 2-layer model	**Unsupervised 3-layer model**
End-position recognition	80%	83%
End-position prediction	80%	53%

Table 6.1: **Accuracy summary of the Spiking Neural Networks response.**

Multiple reasons could explain why the neural network does not recognize every trajectory and end-position perfectly. First, the number of parameters to be optimized is relatively large (16 per layer). Training the second layer takes at least 15 minutes if four trajectories of 3 seconds are presented around 45 times each with 300 ms in between, also accounting for the SpiNNaker hardware to set-up and end the simulations every minute (which takes about 30 seconds each time). Trying a large range of parameters therefore is very time-consuming, and although a great variety of combinations have be tried out, it is possible that a more optimal combination has been missed.

An additional difficulty were the limits imposed by the SpiNNaker hardware and the PyNN software. One problem came from the pre-sensitive scheme (Figure 2.9 in Section 2.3 of Background chapter) of the SpyNNaker STDP implementation: if a neuron never fires, its synaptic weights remain constant. This means that randomly high initial weights can remain high, despite the input neuron not being involved in the trajectory recognition, and Long-Term Depression (LTD) is not strong enough to allow neurons to become selective. I managed to partially overcome this limitation by setting all unmodified synaptic weights to zero after all trajectories had been presented once. Then, only synapses involved in at least one of the trajectory were continued to be strengthened (Long-Term Potentiation, LTP) or weakened (LTD). Also, setting small initial random synaptic weights for the training of the second layer allowed to prevent too much neurons from firing initially (remember that they receive inputs from 1024 neurons). After the learning, less input neurons influence the second layer neurons, and weights need to be larger to trigger post-synaptic spikes.

Another problem arising from SpyNNaker were the strict constraints imposed on the STDP learning parameters: trajectory recordings were long compared to the range of values allowed for the time-constants. In particular, the LTD time-constant τ_- was limited to approximately 30 ms. In the paper by Bichler et al. [28], a fully local and very loosely constrained STDP learning rule is implemented, in which all synapses that are not strengthened are depressed, not only those falling within a specific τ_- time-window. The authors explain that "This [neurons becoming selective] only works if LTD is systematically applied to synapses not undergoing an LTP, even those not receiving a pre-synaptic spike. Therefore, classical STDP mechanisms modeled by the equation $\Delta w = f(t_{post} - t_{pre})$ fail at this task, because it is not possible with this simple rule to depress synapses whose activation time is precisely not correlated with the post-synaptic spike" [28]. In order to reproduce a similar update rule, I tried to set A_- to zero (and τ_- to 0.1 as it cannot be zero), so that synapses were never depressed, and only strong LTP changes occurred. Together with setting all unmodified weights to zero, I was hoping that depressing uncorrelated synapses in this manner would allow a high selectivity of neurons. However this was not very efficient as too many synaptic weights were set to zero (LTD effect a lot stronger than LTP). Therefore, this idea was abandoned and it was not implemented in the final model. As there are many input neurons spiking during the training of the second layer, setting small initial weights for the synapses and updating all unmodified weights to zero was sufficiently effective. Neurons could become selective enough and succeed at their task of recognizing parts of trajectories, although synapses whose activation time was not precisely correlated with the post-synaptic spike were not depressed by STDP (but set to zero if never modified).

Finally, to train the third layer of the neural network, the simulation could be sped up to a factor 10 and a value of 30 ms for τ_- was not too restrictive. In the end, the three-layer neural network was trained using a fully-unsupervised algorithm, in a "layer-by-layer" strategy. The LIF neurons and STDP learning parameters were different from a layer to another but consistent within a layer.

Conclusion

In this project, a Spiking Neural Network model for the visual input processing of a humanoid goalkeeper robot was implemented. Visual animations of balls rolling in a 3D virtual world were simulated using Simulink and recorded with a DVS camera. Using the inputs from this spiking silicon retina, the neural network was simulated in real-time on a SpiNNaker board, using STDP learning and LIF neuron model. Developing the Spiking Neural Network model was a challenging task since the weight updates - especially for LTD - were limited by the built-in STDP mechanism of sPyNNaker.

In the end, the neural network is able to learn, recognize and predict the end-positions of different recorded trajectories of the moving ball. It is remarkable to note that the neural network is able to recognize end-positions of trajectories with different start-positions that have never been presented before, and to some extend, predict end-positions with partial trajectories. Nevertheless, before using this algorithm in a physical robot, several improvements need to be made.

Future work

The most important difficulty to overcome are the time limitations. To solve the time-constant problem as well as to reduce the overall simulation time, shorter recordings of the moving ball should be made. Since recording data was a time-consuming task and that this project had strict deadlines, I did not have the opportunity to make new recordings of trajectories but I would recommend that the next student continuing the project increases the ball speed in the recordings. Bichler et al. do not provide much information on the motorway data recorded; they do not specify how long on average a car stays in the field of view of the DVS camera, but it could certainly be less than 1 second.

A possible solution to improve the results could be the use of a Convolutional Neural Network (CNN) [69], another class of neural networks inspired by biological process, based on the visual cortex [70]. Neurons of the output layer are mapped and not fully connected to the neurons of the input layer. Instead, they are connected to only a small region of the input layer, known as receptive field, and neurons of the same map share the same synaptic weights. Another amelioration coming from the use of CNNs is that it could allow to keep the original 128x128 resolution, presenting the advantage of not loosing spatial information [71][72] and allowing SpiNNaker to interact directly with the DVS camera once an interface between them is developed.

Another objective is to make the algorithm able to predict more distinct end-positions, with the intention to make the end-position range continuous (as compared to current discrete positions). Therefore, the neural network needs to be trained with more trajectories, requiring the recording of new data. Another amelioration would be to train the network using recordings with various camera angles for a more realistic scenario. We can also remember that in this project, the balls are simulated as rolling on a flat surface. In the pursuit of realism, an improvement would be to add an elevation value to the height parameter of the ball, and the neural network would have to predict an end-position within a square instead of on a line.

Bibliography

[1] J T Trachtenberg, B E Chen, G W Knott, G Feng, J R Sanes, E Welker, and K Svoboda. Long-term in vivo imaging of experience-dependent synaptic plasticity in adult cortex.[see comment]. *Nature*, 420(6917):788–794, 2002.

[2] Min Fu and Yi Zuo. Experience-dependent structural plasticity in the cortex. *Trends in Neurosciences*, 34(4):177–187, 2011.

[3] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. pages 1–27, 2018.

[4] Brian Whitworth and Hokyoung Ryu. A Comparison of Human and Computer Information Processing. *Computer*, pages 1–14, 2007.

[5] Luke Muehlhauser and Louie Helm. Intelligence Explosion and Machine Ethics. *Singularity Hypothesis: A Scientific and Philosopohical Assessment*, 6:1–28, 2012.

[6] Chia-Hao Chang and Chen Yubao. Autonomous intelligent agent and its potential applications. *Computers and Industrial Engineering*, 31(96):409–412, 1996.

[7] N. Lazzeri, D. Mazzei, L. Cominelli, A. Cisternino, and D.E. De Rossi. Designing the mind of a social robot. *Applied Sciences (Switzerland)*, 8(2):1–18, 2018.

[8] H Markram. The human brain project. *Sci Am*, 306(6):50–55, 2012.

[9] Martha J Farah. Progress and Challenges in Probing the Human Brain Progress and Challenges in Probing the Human Brain. 526:371–379, 2015.

[10] Arnaud Messé, David Rudrauf, Habib Benali, and Guillaume Marrelec. Relating Structure and Function in the Human Brain: Relative Contributions of Anatomy, Stationary Dynamics, and Non-stationarities. *PLoS Computational Biology*, 10(3), 2014.

[11] Vidushi Sharma, Sachin Rai, and Anurag Dev. A Comprehensive Study of Artificial Neural Networks. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(10):278–284, 2012.

[12] Magdi Zakaria, Mabrouka Al-shebany, and Shahenda Sarhan. Artificial Neural Network: A Brief Overview. *International Journal of Engineering Research and Applications*, 4(2):7–12, 2014.

[13] Lyle N Long and Guoliang Fang. A Review of Biologically Plausible Neuron Models. *AIAA InfoTech@Aerospace Conference*, (April):1–14, 2010.

[14] Horacio G Rotstein. Neurons and Neural Networks: Computational Models Horacio G. Rotstein Farzan Nadim. *ENCYCLOPEDIA OF LIFE SCIENCES / & 2002 Macmillan Publishers Ltd, Nature Publishing Group / www.els.net*, pages 1–15, 2002.

[15] Edmund T. Rolls, Leonardo Franco, Nikolaos C. Aggelopoulos, and Jose M. Jerez. Information in the first spike, the order of spikes, and the number of spikes provided by neurons in the inferior temporal visual cortex. *Vision Research*, 46(25):4193–4205, 2006.

[16] S Mehdi Aghdaee, Lorella Battelli, John A Assad, and Phil Trans R Soc B. Relative timing : from behaviour to neurons Relative timing : from behaviour to neurons. (January), 2014.

[17] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[18] Ahmed A Abusnaina and Rosni Abdullah. Spiking Neuron Models : A Review. 8(June):14–21, 2014.

[19] Jamil Abou Saleh, Fakhreddine Karray, and Michael Morckos. A qualitative evaluation criterion for human-robot interaction system in achieving collective tasks. *IEEE International Conference on Fuzzy Systems*, (June 2012), 2012.

[20] Sergio Davies. Learning in Spiking Neural Networks. *Thesis & Dissertation*, pages 1–27, 2012.

[21] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(MAY):1–23, 2011.

[22] Sergio Davies, Javier Navaridas, Francesco Galluppi, and Steve Furber. Population-based routing in the SpiNNaker neuromorphic architecture. *Proceedings of the International Joint Conference on Neural Networks*, pages 10–15, 2012.

[23] Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2013.

[24] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 $\times$ 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[25] Sen Song, Kenneth D. Miller, and L. F. Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, 2000.

[26] Taro Toyoizumi and J P Pfister. Spike-timing dependent plasticity and mutual information maximization for a spiking neuron model. *Advances in neural information processing systems*, 17:1409–1416, 2005.

[27] Natalia Caporale and Yang Dan. Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. *Annual Review of Neuroscience*, 31(1):25–46, 2008.

[28] Olivier Bichler, Damien Querlioz, Simon J. Thorpe, Jean Philippe Bourgoin, and Christian Gamrat. Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*, 32:339–348, 2012.

[29] A. J F Siegert. On the first passage time probability problem. *Physical Review*, 81(4):617–623, 1951.

[30] L. F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5-6):303–304, 1999.

[31] Florian Jug, Matthew Cook, and Angelika Steger. Recurrent competitive networks can learn locally excitatory topologies. *Proceedings of the International Joint Conference on Neural Networks*, (June):1–8, 2012.

[32] Andrew P Davison. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2(January):1–10, 2008.

[33] SpiNNaker Manchester. Running PyNN Simulations on SpiNNaker. 2015.

[34] Oliver Birbach, Udo Frese, and Berthold Bäuml. Realtime Perception for Catching a Flying Ball with a Mobile Humanoid. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5955–5962, 2011.

[35] R. Mori, K. Hashimoto, and F. Miyazaki. Tracking and catching of 3D flying target based on GAG strategy. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, 5(April):5189–5194, 2004.

[36] Fumiaki Takagi, Hiroto Sakahara, Tetsu Tabata, Hiroyuki Yamagishi, Takashi Suzuki, and Fumio Miyazaki. Navigation control for tracking and catching a moving target. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 866–871, 2009.

[37] Seville Chapman. Catching a Baseball. *American Journal of Physics*, 36(10):868–870, 1968.

[38] Peter McLeod and Zoltan Dienes. Do Fielders Know Where to Go to Catch the Ball or only How to Get There? *Journal of Experimental Psychology: Human Perception and Performance*, 22(3):531–543, 1996.

[39] J Wesson. The science of soccer. pages 19–29, 2002.

[40] D. P. Pelvig, H. Pakkenberg, A. K. Stark, and B. Pakkenberg. Neocortical glial cell numbers in human brains. *Neurobiology of Aging*, 29(11):1754–1762, 2008.

[41] Eric R. Kandel, James H. (James Harris) Schwartz, and Thomas M. Jessell. Principles of neural science, 1991.

[42] Gordon M. Shepherd. The Synaptic Organization of the Brain, 2004.

[43] R. Williams. The Control Of Neuron Number. *Annual Review of Neuroscience*, 11(1):423–453, 1988.

[44] Rebecca Lewis, Katie E. Asplin, Gareth Bruce, Caroline Dart, Ali Mobasheri, and Richard Barrett-Jolley. The role of the membrane potential in chondrocyte volume regulation. *Journal of Cellular Physiology*, 226(11):2979–2986, 2011.

[45] T I Shaw, P C Caldwell, A L Hodgkin, and R D Keynes. The effects of injecting 'energy-rich' phosphate compounds on the active transport of ions in the giant axons of Loligo. *From the Laboratory of the Marine Biological Association, Plymouth and the Physiological Laboratory , University of Cambridge*, pages 561–590, 1960.

[46] William A Catterall. From Ionic Currents to Molecular Review Mechanisms: The Structure and Function of Voltage-Gated Sodium Channels. *Neuron*, 26:13–25, 2000.

[47] Eugene M. Izhikevich. Polychronization: Computation with Spikes. *Neural Computation*, 18(2):245–282, 2006.

[48] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–33, 2011.

[49] Valentino Braitenberg. Corticonics: Neural Circuits of the Cerebral Cortex, 1992.

[50] Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.

[51] F. Rieke, D. Warland, R. De Ruyter Van Steveninck, and W. Bialek. Spikes: Exploring the Neural Code, 1997.

[52] Wulfram Gerstner and Werner M Kistler. Spiking Neuron Models: Single Neurons, Populations, Plasticity. *Book*, page 494, 2002.

[53] Christof Koch. Biophysics of Computation: Information Processing in Single Neuron, 2004.

[54] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215, 1997.

[55] G Q Bi and M M Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 18(24):10464–10472, 1998.

[56] Per Jesper Sjöström, Gina G. Turrigiano, and Sacha B. Nelson. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32(6):1149–1164, 2001.

[57] P J Sjöström, a Rancz, a Roth, and M Häusser. Dendritic Excitability and Synaptic Plasticity Dendritic Excitability and Synaptic Plasticity. *Physiological Reviews*, 88(November 2009):1–28, 2010.

[58] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner. A neuronal learning rule for sub-millisecond temporal coding, 1996.

[59] Richard Kempter, Wulfram Gerstner, and J Leo van Hemmen. Spike-Based Compared to Rate-Based Hebbian Learning. *Advances in Neural Information Processing Systems (NIPS)*, (Hebb 1949):125–131, 1998.

[60] J.-P. Pfister. Triplets of Spikes in a Model of Spike Timing-Dependent Plasticity. *Journal of Neuroscience*, 26(38):9673–9682, 2006.

[61] Robert C. Froemke. Temporal modulation of spike-timing-dependent plasticity. *Frontiers in Synaptic Neuroscience*, 2(June):1–16, 2010.

[62] Xin Jin, Alexander Rast, Francesco Galluppi, Sergio Davies, and Steve Furber. Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware. *Proceedings of the International Joint Conference on Neural Networks*, (1), 2010.

[63] Nassim Khaled. *V-Realm Builder©. In: Virtual Reality and Animation for MATLAB® and Simulink® Users*. Springer, London, 2012.

[64] Patrick Lichtsteiner, Christoph Posch, Tobi Delbruck, and Senior Member. Temporal Contrast Vision Sensor. *Work*, 43(2):566–576, 2008.

[65] Peter O'Connor, Daniel Neil, Shih Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, 7(7 OCT):1–13, 2013.

[66] Tobi Delbruck and Manuel Lang. Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor, 2013.

[67] Peter U Diehll, Daniel Neill, Jonathan Binas, Matthew Cook, Shih-chii Liu, and Michael Pfeiffer. Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing, 2015.

[68] Andrew D. Brown, Steve B. Furber, Jeffrey S. Reeve, Jim D. Garside, Kier J. Dugan, Luis A. Plana, and Steve Temple. SpiNNaker - Programming model. *IEEE Transactions on Computers*, 64(6):1769–1782, 2015.

[69] P Haffner, Y Lecun, L Bottou, and Y Bengio. Gradient-Based Learning Applied to Document Recognition, 1998.

[70] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[71] LISA Lab. Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation.

[72] Hamed Habibi Aghdam and Elnaz Jahani Heravi. Guide to convolutional neural networks : a practical application to traffic-sign detection and classification.