Implementation of a closed–loop homeostasis on the BrainScaleS system

Felix Schneider – Internship report

March 27, 2018

Chapter 1

Introduction

Learning experiments are in general very time-consuming, due to its parallel architecture neuromorphic hardware has a huge advantage compared to usual van-Neumann architecture. The *BrainScaleS* wafer-scale hardware system in Heidelberg emulates networks of simplified neuron models and enables highly accelerated operations (x 10^4).
A full wafer module implements around 200 000 neurons which can be interconnected via synapses. This allows to build up individual configured neuronal networks and analyse the interaction of neurons with each other. The platform offers different types of operation modes — we wil use a closed – loop operation mode which enables us to actively react and influence the spiking behavior during emulation.
The goal during this internship was to implement a spike based homeostasis in this mentioned mode. Applied on neurons, homeostasis can be used to set and maintain a target activity – this is especially interesting if the neuron is disturbed by noise input from other spikes sources. At the beginning of this report we will introduce the basic ideas behind homeostasis and closed–loop. Afterwards we start with a simple implementation of a homeostasis using NEST, a simulator for spiking neuronal networks. Therefore the PyNN API was used which is a programming language to define neuronal networks at a high-level of abstraction. These defined networks can then be simulated with NEST and later emulated on the *BrainScaleS* system.
A possible application for this mechanism is a *Spike Based Expectation Maximation* experiment (SEM). This method can find maximum likelihood solutions in an unsupervised setup. An introduction into this topic can be found in [1] and is beyond the scope of this report.

Chapter 2

Theorertical background

2.1 AdEx neurons

On the NM-PM1 (BrainScaleS system) we use deterministic neurons, namely *adaptive exponential integrate and fire (AdEx)* neurons. The behavior of those neurons can be described by the following differential equation

$$C_m \frac{du_k}{dt} = -g_l(u_k - E_L) + g_L \Delta_T exp\left(\frac{u_k - V_T}{\Delta_T}\right) - w(t) + I(t) \tag{2.1}$$

$$\tau_w \frac{dw}{dt} = a(u_k(t) - E_L) - w \tag{2.2}$$

u_k describes the membrane potential, C_m the membrane capacitance, E_L the leak reversal potential, g_L the leakage conductance, w the adaption current and V_T the threshold. If u_k reaches this threshold, we detect a spike. Afterwards the membrane potential is reset to V_{reset} and stays there for a time period τ_{refrac}. We say the neuron is in a refractory period and is unable to elicit a spike. If we take the limit $\Delta_T \to 0$ and drop the adaptation current w, we receive the leaky integrate-and-fire model.
To build up neuronal networks, we connect neurons via synapses. These synapses can have different connection strength also called synaptic weight, which corresponds to the influence a spike from one neuron has to the connected neuron.
Furthermore, the biological process of *spike timing dependent plasticity* (STDP) is implemented on the hardware. This process adjustes the connection strength between neurons based on the relative timing of a neurons input and output. It could be shown that STDP can be used as a learning algorithm for artificial neuronal networks [7]. Therefore this is a important feature of the BrainScaleS system.

2.2 Homeostasis

The biological term “homeostasis“ is a combination of the words “homeo“, standing for “similar“ and “stasis“, defining a period or state of equilibrium. It was coinded by Bernard and Cannon referring to their concept of internal environment in which cells live and explains the effect of self regularization. Examples for variables maintained by a homeostatic process are the human body heat or the sugar level within the body.
In general one could compare homeostasis with a feedback control system, A simple non biological example for such a feedback control system is a thermostat, it performs an homeostasis mechanism by switching the heating in response to a temperature sensor. In Figure 2.1 we see the principle workflow of these processes. The current value of the variable which should be kept constant can differ from the target value due to some disturbances. This will be detected by a receptor which acts like a sensor and transfers this information to a controller, in case of the human body the brain. This controller reacts, based on the given informations, by activating an effector which will counteract the current change of the variable – this is called a negative feedback loop.

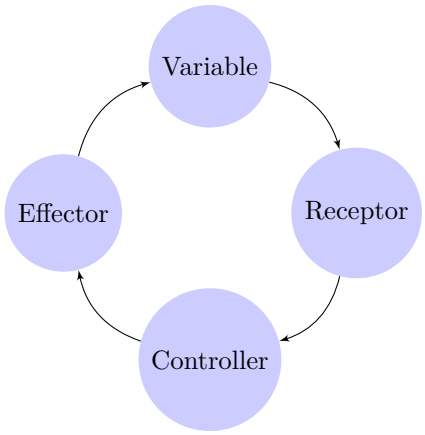


Figure 2.1: Schematic view of the underlying process of homeostasis. A receptor acts as a sensor and detects if the value of the variable which should be maintained differs from the target value. This information will be transfered to a controller which activates an effector that controls the variable.

2.3 Closed–loop

In conventional simulations on the BrainScaleS system, the data flow – from user input to measurement results – is a one–way process. Figure 2.2 shows how the user input is evaluated in the different layers of the system from translating, mapping and hardware configuration to the emulated results which could be for example a membrane potential over time or a spike train of a neuron. The user input in form of a PyNN script is evaluated by the PyNN API PyHMF. The following mapping process is done by the marocco mapping tool. Marocco uses also calibration and blacklisting informations for this process. The last layer, called HALbe, performs the conversion of software configruation data to the hardware–specific format. An overview of these different software modules can be found in [2].

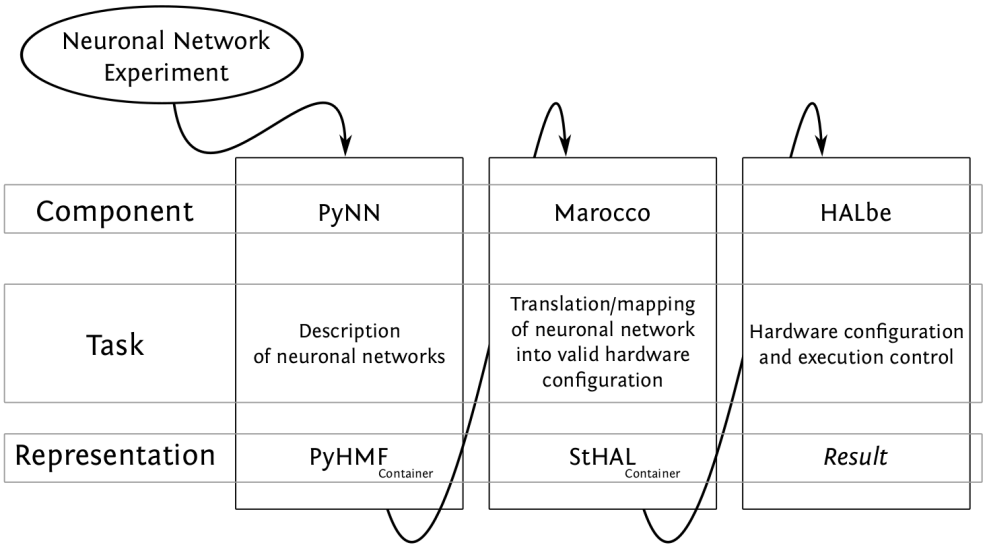


Figure 2.2: Schematic of conventional experimental procedure. After the neuronal network is described in PyNN, the network will be translated and mapped to the hardware where the emulation will take place. After the emulation we receive the results of the described network but we have no influence on the neurons during emulation. (Figure taken from [2])

Another approach is to use a *closed–loop* setup where data does not flow just in one way. The neuromorphic system interacts with a regular computer – forming a hybrid system. A typical example is the interactive control of a robotic system. The output of the neuromorphic part will controll the movement of the robot and will influence the sensory data the robot receives. The sensory information will then be send back to the neuromorphic part.

We will emulate a neuronal network on the BrainScaleS system, where the control host will read-out spike information from the neuromorphic part. Based on this information the host will influence the behavior of the neuromorphic part by sending spikes back. Figure 2.3 shows this closed–loop concept.
For highly accelerated hardware like the BrainScaleS system, this task is very challenging due to timing constraints. The computation on the host and the communication between the two systems has to keep up with the accelerated neuromorphic system.

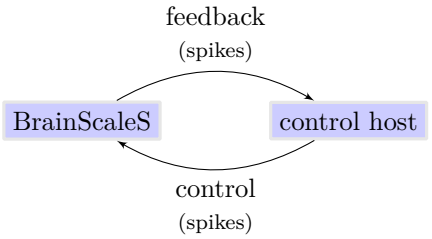


Figure 2.3: Schematic view of the closed–loop mode of the BrainScaleS system. By combining the neuromorphic system and the conventional computer which acts as a controller, we form a hybrid system. The host will receive a feedback in form of spike information from the neuronal network. The host can send spikes back to the neuromorphic system and actively influence the emulation.

Because the BrainScaleS system operates in continuous time, it can not be paused during emulation. Therefore we do not have synchronization points, where data can be exchanged between the systems. A fast computability and precise timing control is therefore an important aspect of the software implementation.

2.4 Closed–loop homeostasis

The goal of this internship was to apply the homeostasis mechanism (see Chapter 2.2) to the spike frequncy of neurons on the BrainScaleS system. We used the closed–loop mode (see Chapter **??**) to influence the spiking behavior of a neuron. This could be achieved by sending spikes from the host to the network in a negative feedback-loop. Figure 2.4 shows how this could be accomplished using a proportional controller.

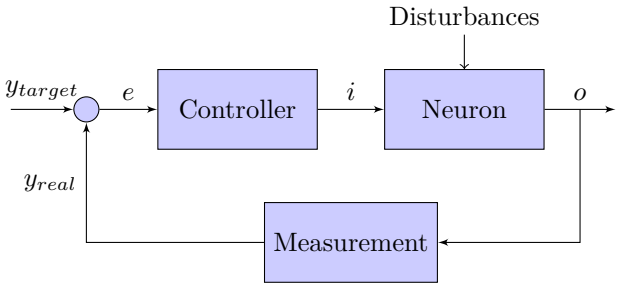


Figure 2.4: Schematic view of homeostasis, y_{target} is the target frequency of the neuron and $y_{target} - y_{real}$ the error (e), the controller adapts the input (i) to the neuron in a way that the output is closer to the target frequency y_{target}. The output o will then be measured and compared with the target frequency y_{target} where the loop starts again.

We set a target activity y_{target} and after each frequency measurement we can compare the target activity and the current activity y_{real} and define an error $e(t)$. The adjustments $u(t)$ will then be proportional to this defined error with some constant K_P.

$$e(t) = y_{target} - y_{real}(t) \tag{2.3}$$

$$u(t) = K_P \cdot e(t) \tag{2.4}$$

The proportionality constant K_p will define how strong the adjustments at every adaptation are and can have a strong influence on the quality of the result. If K_p is to large, the frequency can

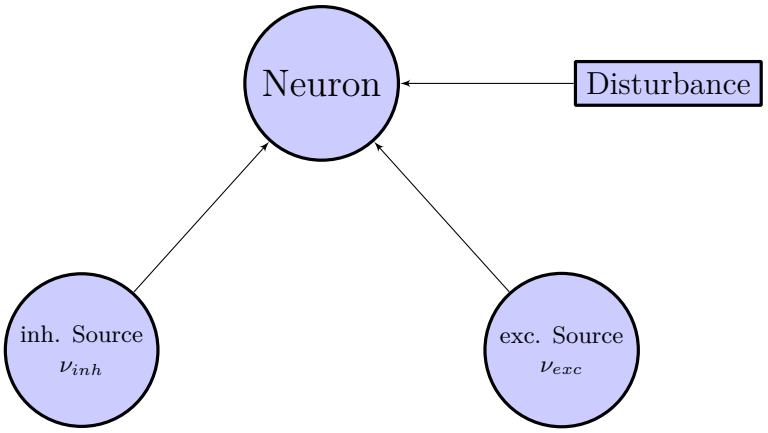


Figure 2.5: Schematic topology of the homeostatic setup. The neuron we want to apply homeostasis on receives disturbing input in form of spikes from other neurons. The target frequency can be reached by adapting the frequency of the connected inhibitory and excitatory spike sources correspondingly.

oscillate around the target frequency, if it is to small, the adjustments will be very slow or the target frequncy will never be reached – a steady state error $e_{ss} = \lim_{t\to\infty} e(t)$. These are typical proplems of a proportional controller and will be discussed later in this report.
Figure 2.5 shows schematically how input in form of spikes is sent from the control host to the neuron to regulate the neurons frequency. The control host acts as a controller in this setup. It reads out the current frequency and adjusts the frequencies ν_{inh} and ν_{exc} of the inhibitory and excitatory spike source. These excitatory and inhibitory spike soure are the effector in this homeostasis implementation. ν_{inh} and ν_{exc} is adjusted in a way that they counteract the behaviour of the neuron when the frequency starts to differ from the target frequency.

2.5 Sea-of-noise network

If we apply homeostasis to a neuron without input from other neurons, ν_{exc} will be increased till the target frequency is reached. But we also want to investigate how homeostasis behaves with additional disturbing input from other neurons. A method for a background spike source is a *Sea-of-noise network* (SoN). We connect a population of neurons inhibitory with each other randomly. The threshold of each neuron is set below the resting potential and all neurons spike permanently. Some of these neurons in the SoN–network can then be connected to the neuron. The spike frequency of this neuron depends on the amount of connections to the SoN–network, the synaptic weigh, the amount of neurons in the SoN network and their refractory period τ_{ref}. This kind of background source was choosen because it will be necessary in a *Spike based Expectation Maximation* experiment which is the goal after this internship.

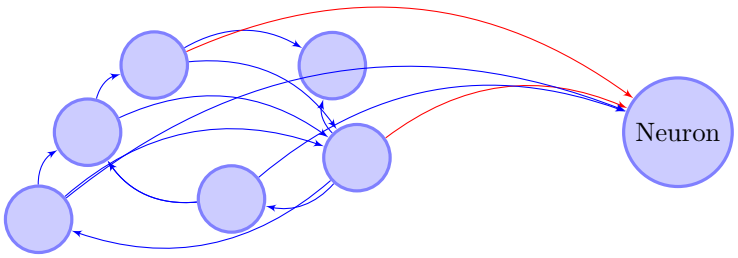


Figure 2.6: For neurons in a *SoN*-network the threshold v_{thresh} is set below the resting potential v_{rest}. This leads to a bursting behaviour of the neurons. These neurons have a random inhibitory (blue) connection whith each other. Some of these neurons from the SoN population are excitatory (red) or inhibitory connected to the external neuron.

Chapter 3

NEST – Simulation

This NEST implementation was carried out to understand the principles of homeostasis and to get used to work with PyNN which will also be used on the hardware implementation. The spiking neuronal network simulator NEST can be used to simulate the dynamics of neuronal systems. Important to note is that this simulation was not done in real–time. After a simulation time of 1000 ms the simulation is paused and the corresponding parameters are adapted based on the past simulation step before the simulation continues.
In Figure 3.1 we can see the topology for the implemented model, which is slightly different compared to Figure 2.5. In this case we can adapt the frequency of the inhibitory and excitatory source as well as the synaptic weights which connect the sources and the neuron. In the emulation on the BrainScaleS system in chapter 4 we will only adjust the frequencys of the spike sources but a weight adaptation should be implemented later. Because the weight resolution on the hardware is limited to 4 bits, increasing the weight by one will have a strong influence on the neuron. This weight adaptation can only be used for a rought adaptation of the neurons frequency. The rate adaptation of the spike sources will be used for fine–tuning. However, this is only valid for the hardware emulation and not for the NEST – simulation. The weight resolution in the NEST simulation is not limited and we could use it for fine–tuning of the neurons frequency as well.

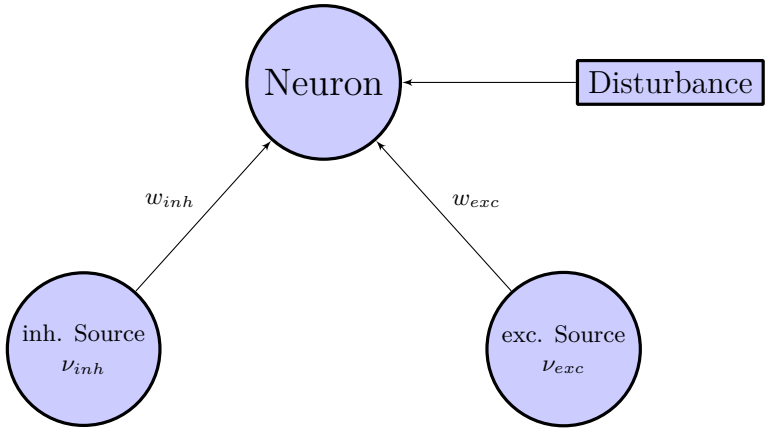


Figure 3.1: The figure shows the topology for this NEST–simulation. The neurons frequency will be maintained by adjusting the rates of the inhibitory and excitatory source and the synaptic weights which connect the spike sources and the neuron. The adjustmends will be determined through a negative feedback–loop as described in chapter 2.2.

Figure 3.2 shows how homeostasis is applied to the neuorns frequency. Plot **A** shows the rate of the neuron over time steps. The red dashed line shows the target rate, the gray line how the neuron would behave if we would not appy homeostasis. We can see that between time step 25 and 50 and from time step 100 till the end the neuron receives disturbing input. Plot **B** and **C** show how the weights and rates of the spike sources is adapted over time. At the beginning the neuron receives no input from other neurons and the target frequency is reached by increasing excitatory weights and the excitatory rate. After we introduce some external input as disturbance, the weights and rates are adapted correspondingly to counteract these distrubances till the target frequency is

reached. The goal is now to use the same mechanism on neurons on the BrainScaleS system. As described before, we will start with adaptation of the spike source rates, weight adaptation will not be discussed in this report.

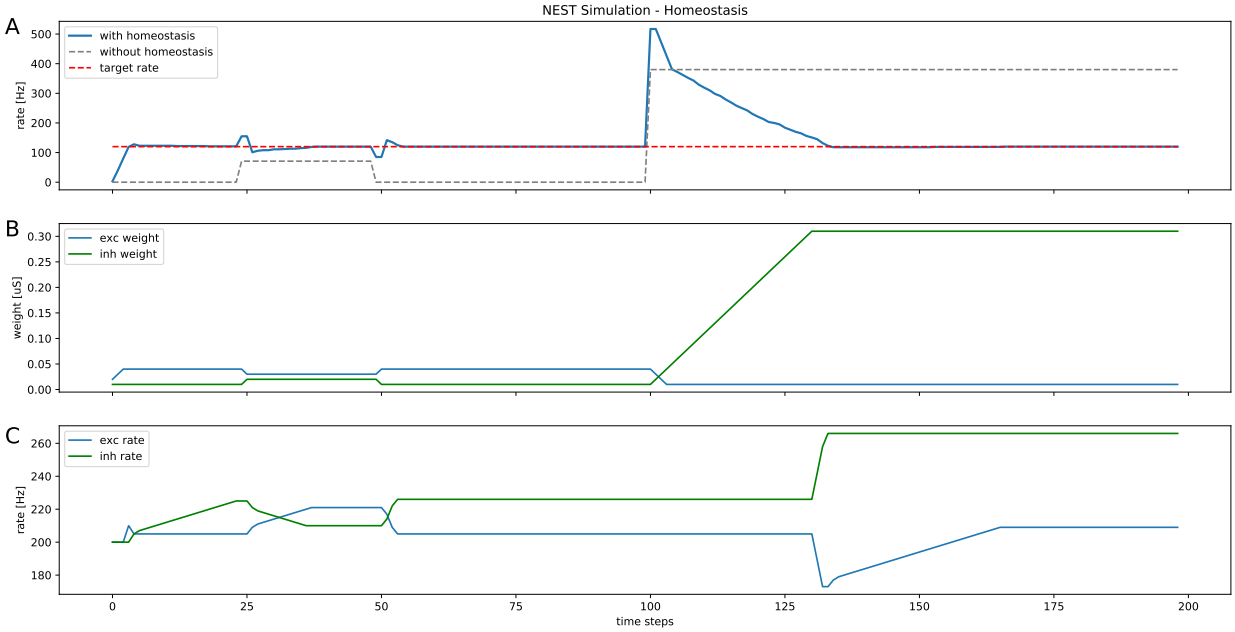


Figure 3.2: The X-axis is given in time-steps because this simulation was not carried out in real-time. After a simulation time of 1000 ms the simulation was paused, the current reight of the neuron computed and weights and rate adapted correspondingly. This plot shows this computed rates and the adjustments on the parameters. In the upper plot we can see how the frequency of the neuron changes, the blue graph shows the applied homeostasis, the gray plot shows how the neuron would behave without homeostasis. The plots below show how the weights and the frequency of the two poisson sources are adapted. At the beginning there is no external signal connected to the neuron and the frequency will be regulated towards the target frequency of 120 Hz. Between time-step 25 and 50 there is an DC current with an amplitude of 0.5 nA connected to the neuron to show how the homeostatis deals with such external disturbances. After time step 100 there is an external poisson source with a rate of 500 Hz connected (weight = 0.05 uS) to the neuron to simulate another form of disturbance.

Chapter 4

Hardware – emulation

All units in this chapter are given in hardware time. For biological time one has to take the speed–up factor of 10^4 into account. We will use the closed–loop operation mode as described in figure 2.3 for this implementation. We will use two different threads during the emulation on the control host. The first thread will read–out spike information of the neuron from the corresponding FPGA. Combined with the elapsed time we can compute the current frequency of the neuron. The second thread will react on this frequency information and adjust the amount of excitatory or inhibitory spikes which are sent to the neuron. This is similar to the previous NEST simulation but without weight adaptation. There are two more differences to the NEST simulation we have to deal with. We can either send excitatory or inhibitory spikes but not both simultaneously as we did in the NEST simulation. The second point is that we can not define a rate ν_{exc} or ν_{inh} for the spikes we are sending from the host. We can only call a `send_spike` function to send a single spike to the neuron. We will use this function within a while–loop to send spikes continuously.
With a sleep–time within this loop after every sent spike, we can control the amount of spikes we are sending. We use variables $sleep_{exc}$ and $sleep_{inh}$ which define the sleep–time between every sent excitatory or inhibitory spike. These variables can be adapted during emulation and can be used to control the amount of spikes we are sending from the host.

4.1 Timing – control

If we set this sleep time to zero, which means sending spikes to the neuron as fast as possible, this `send`–process should always take the same time period. We measured the time period between sending a spike and sending the next spike. This can be seen as an interspike interval distribution, but it is important to note that we measure only the time when we send a spike from the host and not when the neuron spikes. We will refer to this as ISI_{send}. This measurement is important to quantify the latency caused by the software.
Figure 4.1 shows this measruement where we sent one million spikes and plotted this distribution in a histogram. The x-axis shows the different time durations which occured during the measurement, the y-axis shows the relative occurence logarithmically scaled. All four plots show the same measurement but different cut–outs. We want to adjust the amount of spikes we are sending to the neuron by adjusting the sleep between each spike. Therefore this distribution should become as narrow as possible. Otherwise outliers wich are several orders of magnitude away from the main peak would disturb the rate adaptation of the neuron.
To avoid outliers and to get a sharper peak, we did the following adjustments. The simplest and most straightforward modification was to pre–allocate all necessary memory before the emulation. The next step was to use the linux `schedtool` which allows us to controll the CPU scheduling behavior, because sending spikes and receiving information about the spikes of the neuron are two functions running simultaneously in two different threads. The scheduler decides on the basis of the used scheduling policies and the priority of each thread which thread will be executed next. A real time scheduling policie is `SCHED_FIFO` which stands for First in - first out scheduling.
`SCHED_FIFO` uses lists of runnable threads for every possible priority (1 (low) -99 (high)) to decide which thread will be executed next. If a thread with higher priority than the current thread becomes runnable, the current thread will be paused and added to the head of the corresponding priority – waiting list. If we want to put a thread to the end of the list, we cann call `sched_yield`

and the next thread will be executed. This is useful in our context because we have long waiting times in the send and read–out thread where nothing happens and other processes can be executed.

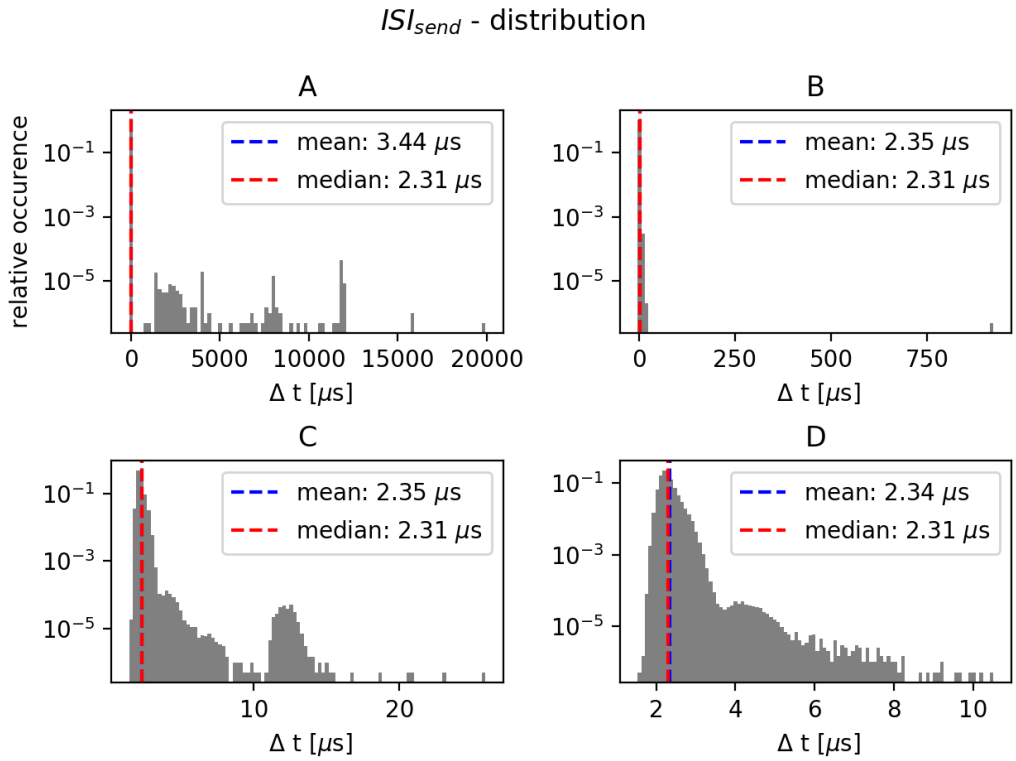


Figure 4.1: This four histograms show different ranges of the same distribution. On th x-axis are the different occured time periods given, on the y-axis the relative occurence of these time periods. We sent spikes to the neuron as fast as possible and measured the time period between sending a spike and sending the next spike. Due to disturbances like other processes which are running on the host or memory allocating we receive such a widespreaded distribution. The goal is to get this distribution as narrow as possible for a precise timing control.

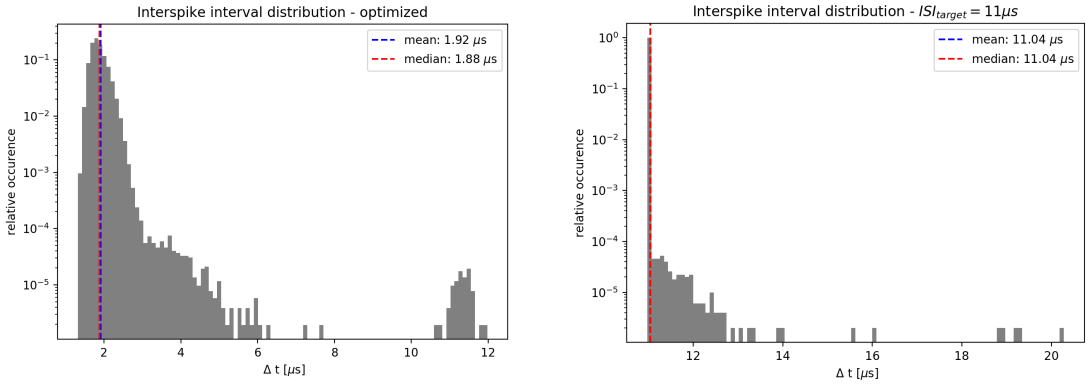
In general such a scheduled thread runs until it is blocked, the thread calls `sched_yield()` or a thread with higher priority becomes runnable.
The host will distribute all runnable processes on all available CPUs in the most efficient way. Switching the process on different CPUs could also cause latencies, therefore we fixed this process on an appointed CPU to avoid this.
In Figure 4.2a we can see how these adjustments affect the distribution. This Figure shows the whole measurment and is not a cut–out as before. The histogram has become more contracted and the biggest outlier is around 11 μs, unfortunately we do not know where these outliers are comming from, yet. The main peak is around 2 μs which would be a frequency of 500 kHz. Due to the speed–up this represents 50 Hz in biological time.

```
1 last = gettime();
2 send_spike();
3 while(True){
4  if((gettime() - last) > sleep)
5     break;
6  sched_yield()
7 }
```

Listing 4.1: This code snippet shows how the sleep process is implemented. `last` gives the sended spike a time-stamp and the while loop will not be leaved until the sleep period is elapsed. Until then the while loop will call `sched_yield` and the other thread will continue.

During the emulation we set the sleep–time to some value and the next spike will be sent if this

sleep–time has elapsed. Figure 4.2b shows how the distribution looks like if we set sleep–time to 11 μs. This means that we try to reach an interspike interval of $ISI_{target} = 11\ \mu s$. Here we used again the ISI_{send} definition described above. The main peak contains now 99.95 % of the entries in the histogram (in total one million) and has a width of 0.1 μs. Furthermore, outliers several orders of magnitude away from the main peak could be avoided. The code snippet in listing 4.1 shows how the wait process is implemented. `last` gives the sent spike a time stamp and the while loop will be exited if the sleep–time `sleep` has elapsed. Otherwise, `sched_yield()` will be called, the current thread will be added to the waiting list by the scheduler and other processes can be executed.



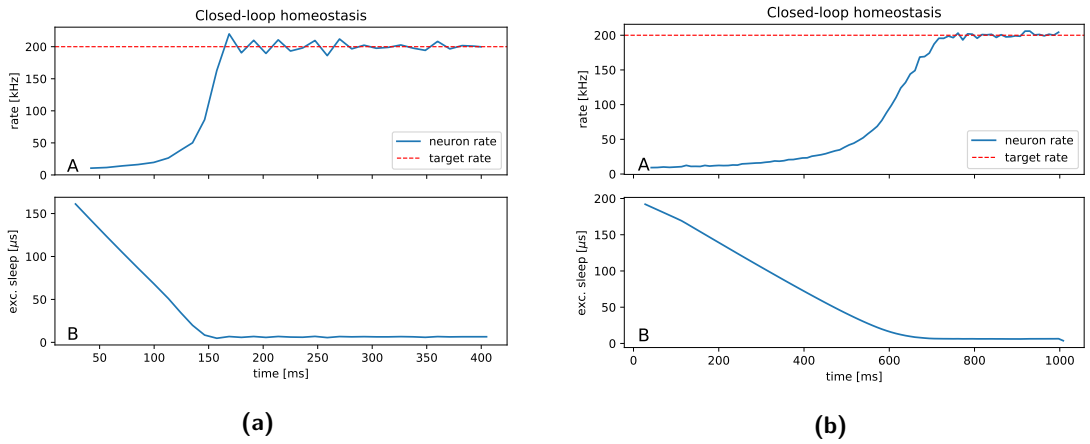
(a) Sending spikes with maximum speed

(b) Sending spikes with a set ISI_{target} of 11 μs

Figure 4.2: These two plots show the ISI_{send} distribution after the described optimization. The x-axis shows the occured time durations and the y-axis the relative occurence. Plot a) shows how the distribution looks like if we send spikes to the neuron as fast as possible, b) shows a distribution with a sleep (compare Listing 4.1) of 11 μs.

4.2 Homeostasis emulation

Now we can use the variables $sleep_{exc}$ / $sleep_{inh}$ to control the amount of spikes we are sending to the neuron. At the beginning we will not use any background spike source and the neuron will only receive spikes which we send from the host. In Figure 4.3a it is shown how the neurons frequency reaches the target frequency (red dashed line) after approximately 200 ms. A typical problem for a proportional controller is the oscillation around the target value. We can reduce this oscillation by decreasing the proportionality constant K_P. But this will lead to a longer period of time to reach the target value (see Figure 4.3b). A compromise between accuracy/oscillation and speed has to be made. A discussion how this can be improved can be found in the last chapter. The proportionality constant K_P for Figure 4.3a is five times as big as in Figure 4.3b. In Figure 4.3a the target value is reached after around 160 ms but with strong oscillations around the target value. In Figure 4.3b, the target value is reached with smaller oscillations around the mean but only after around 790 ms.
In later experiments the neuron will receive input from other neurons and homeostasis applied to this neuron should keep the neuron on a set target activity. Therefore we use a Sea-of-noise network (see chapter 2.5) as source for the neuron. Figure 4.4 shows how this looks like. The plot **A** shows the frequency of the neuron over time and the target rate (red dashed line). Plot **B** and **C** show how the corresponding sleeps are adjusted. This plot can be separated into four parts. The first part is from 0 ms to 500 ms. The neuron receives only input from the SoN network and its frequency is around 100 Hz. After 500 ms the homeostatic process starts. We start sending excitatory spikes from the host to the neuron to reach the target frequency of 200 Hz. Because the waiting time between two spikes is at the beginning at 200 μs very high, the impact on the neurons frequency is very low. When we start to decrease $sleep_{exc}$, we send more spikes to the neuron and its frequency rises. Around 750 ms the target frequency is reached and $sleep_{exc}$ stays constant. After 1000 ms the target frequency switches to 50 Hz and $sleep_{exc}$ increases till it reaches again 200 μs and has almost no effect on the neurons frequency. Because the current frequency of the neuron is still above the target frequency, we start sending inhibitory spikes around 1300 ms. After 1900 ms the target frequency is reached and $sleep_{inh}$ stays constant.



(a) **(b)**

Figure 4.3: These plots show how the frequency of a neuron can be adjusted. The neuron receives no other input than the spikes we are sending from the host. The left plot uses a five times higher proportionality constant K_P than the right plot. The upper plots show the frequency of the neuron (blue line) and the target frequency (red dashed line). The plots below show how the sleep between each sent spike is adjusted. We can see that a trade–off between accuracy and speed of the adjustments has to be made. A faster adaptation of the frequency, lead to stronger oscillations.

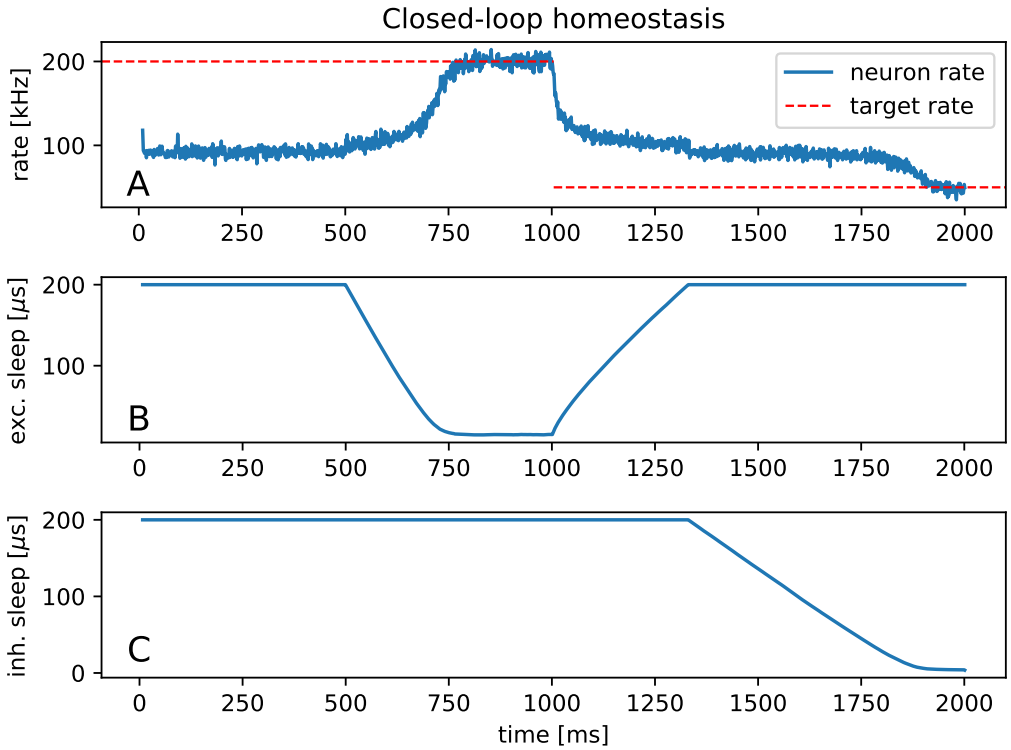


Figure 4.4: **A** shows the frequency of the neuron over time and the target frequency. **B** and **C** show how $sleep_{exc}$ and $sleep_{inh}$ evolves over time. This plot can be subdivided into four parts. 1) 0 ms-500 ms, no spikes from the host, input only from SoN network. 2) 500 ms - 1000 ms, sending excitatory spikes from the host to reach the target frequency of 200 Hz. 3) 1000 ms - 1300 ms, increasing $sleep_{exc}$ till 200 μs. 4) 1300 ms - 2000 ms start sending inhibitory spikes and decrease $sleep_{inh}$ till the target frequency of 50 Hz is reached.

Chapter 5

Conclusion and Outlook

The goal of this internship was to implement a closed–loop homeostasis on the BrainScaleS system. This goal could be fulfilled, but there are still some possibilities for improvement which we want to discuss here.
To perform homeostasis in real-time, we had to make sure that the period of time for sending a spike takes always the same period of time. This could be achieved by using a scheduling tool, pre–allocating the necessary memory and pinning the process to a CPU. From the software side, the maximum speed with which spikes from the host can be sent defines an upper limit for the maximum frequency. This upper limit is around 500 kHz (50 Hz biological). A possibility to push this limit upwards could be to send several spikes during one iteration of the while–loop. It would be useful to improve this even further to reach higher frequencies.
In general, a benchmark seems meaningful to examine which frequencies can be reached and to what accuracy the frequency can be maintained.
The next point is that currently either excitatory or inhibitory spikes are sent to the neuron. To get a smooth transition between sending inhibitory and excitatory spikes, we have to increase $sleep_{exc}$ / $sleep_{inh}$ till the sent spikes have a neglectable impact on the neurons frequency. Afterwards we can start to decrease $sleep_{inh}$ / $sleep_{exc}$ correspondingly. By sending excitatory and inhibitory spikes and adapting the corresponding sleeps simultaneously, this transition can be done smoother and faster. This can be done by using different threads for excitatory and inhibitory spikes.
As mentioned before, with a proportional controller problems like oscillation around the target value or a steady-state error can occur. Therefore in Figure 4.4 we choosed a proportionality constant in the range of the constant from Figure 4.3a. This leads to a fast adaptation of the frequency and the oscillation can be neglected compared to the variance of the background source. This proportionality constant is choosen by hand and should be computed automatically and adapted by hand only if necessary.
There exist rules of thumb like the "Ziegler–Nichols" method how to set the proportionality constant. The idea of this method is to find a critical constant $K_{P,crit.}$ which leads to constant oscillation. The corresponding constants can then be looked up in tables. For a simple proportional controller this is $K_P = \frac{1}{2}K_{P,crit.}$. The problem is that we can not determine $K_{P,crit.}$ for every setup the homeostasis is applied on.
Another possibility to improve the behaviour of this controller is by adding a derivative or integral term. A derivative term adapts the adjustment $u(t)$ proportional to the derivative of $e(t)$.

$$u(t) = K_D \cdot \frac{d\,e(t)}{d\,t} \tag{5.1}$$

A strong adaptation leads to overshooting which is the reason for an oscillating behavior of the frequency. The derivative term can decrease overshooting because the influence of this term decreases if the error does not change.
The third possible term is an integral term, it accumulates the errors from the past and can eliminate the steady-state error which can occur with a simple proportional controller.

$$u(t) = K_I \cdot \int_0^t e(\tau)d\tau \tag{5.2}$$

The disadvantage of this term is that it increases the oscillatory behavior of the controlled variable. Therefore it is important to choose the right proportionality constants to determine the influence

of these three terms on the frequency. These three terms are added up and the final adjustment for every update will be calculated as follows.

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(\tau)d\tau + K_D \cdot \frac{d\,e(t)}{d\,t} \tag{5.3}$$

A realization of these points and the usage in a spike based expectation maximation experiment could be done as part of a bachelor thesis. First ideas for such an experiment werde developed, were we would use three neurons for the learning process.
It can be shown (see [4]) that a group of neurons receiving structured input and using spike timing dependent plasticity perform expectation maximation. In a regular SEM experiment patterns need to be normalized, otherwise some patterns would have an intrinsic advantage compared to other patterns. This normalization could be avoided by applying homeostasis to the activity of the neurons which receive the patterns as structured input in form of Poisson spike trains – the cause layer neurons. A detailed introduction into this topic can be found in [1].

Bibliography

[1] Oliver Breitwieser, *Towards a Neuromorphic Implementation of Spike-Based Expectation Maximization*, Master Thesis, University Heidelberg, 2015.

[2] Eric Müller, *Novel Operation Modes of Accelerated Neuromorphic Hardware*, PhD Thesis, University Heidelberg, 2014.

[3] Andrew P. Davison, Eric Müller, Sebastian Schmitt, Bernhard Vogginger, David Lester, Thomas Pfeil, *HBP Neuromorphic Computing Platform Guidebook* , 09.03.2018

[4] Nessler, B., Pfeiffer, M., Buesing, L., and Maass, W., *Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity.* PLoS Computa- tional Biology, 9(4):e1003037., 2013

[5] Ziegler, Nichols, *Optimum settings for automatic controllers*, Trans. ASME, 64 (1942), pp. 759-768

[6] man7 – Linux Programmer's Manual, *http://man7.org/linux/man-pages/man7/sched.7.html*, 21.03.18

[7] O'Connor P, Neil D, Liu SC, Delbruck T and Pfeiffer M (2013) *Real-time classification and sensor fusion with a spiking deep belief network. Front. Neurosci. 7:178.* doi: 10.3389/fnins.2013.00178