Towards Closed-Loop Experiments on Neuromorphic Hardware

Visual Feedback Experiment Implementation

Nils Fischer

Electronic Visions, Kirchhoff Institut für Physik, Universität Heidelberg

Projektpraktikum im Sommersemester 2014

This experiment is designed to probe the capabilities of the *Hybrid Multiscale Facility (HMF)* neuromorphic hardware system in Heidelberg to run in a closed-loop environment, communicating with a cluster computer in realtime. The work presented here lays the foundation to implement such an architecture by providing a working implementation of the experimental setup running on a neuronal simulator (NEST) with both continuous and incremental updates and on the HMF in DNC loopback mode. Additionally, a preliminary hardware neuron calibration is explored to find suitable neurons for this experiment.

Contents

1 The Visual Feedback Experiment

Figure 1 gives an overview of the experimental setup. A cluster computer simulates an object in the *Feedback Environment*, given by its position on an axis. The neuromorphic hardware models a number of neurons that represent a simple retina, the *Retinotopic Response*. In every timestep, the Feedback Environment generates a number of spikes for each of the Retinotopic Response's neurons that follow a gaussian curve centered around the object position. Upon receiving the spikes, the neurons mirror the incoming spikes, thus effectivly *seeing* the object. The Feedback Environment then assumes the object to be at the position given by the mirrored spikes and applies a force

$$F = k \cdot x \tag{1}$$

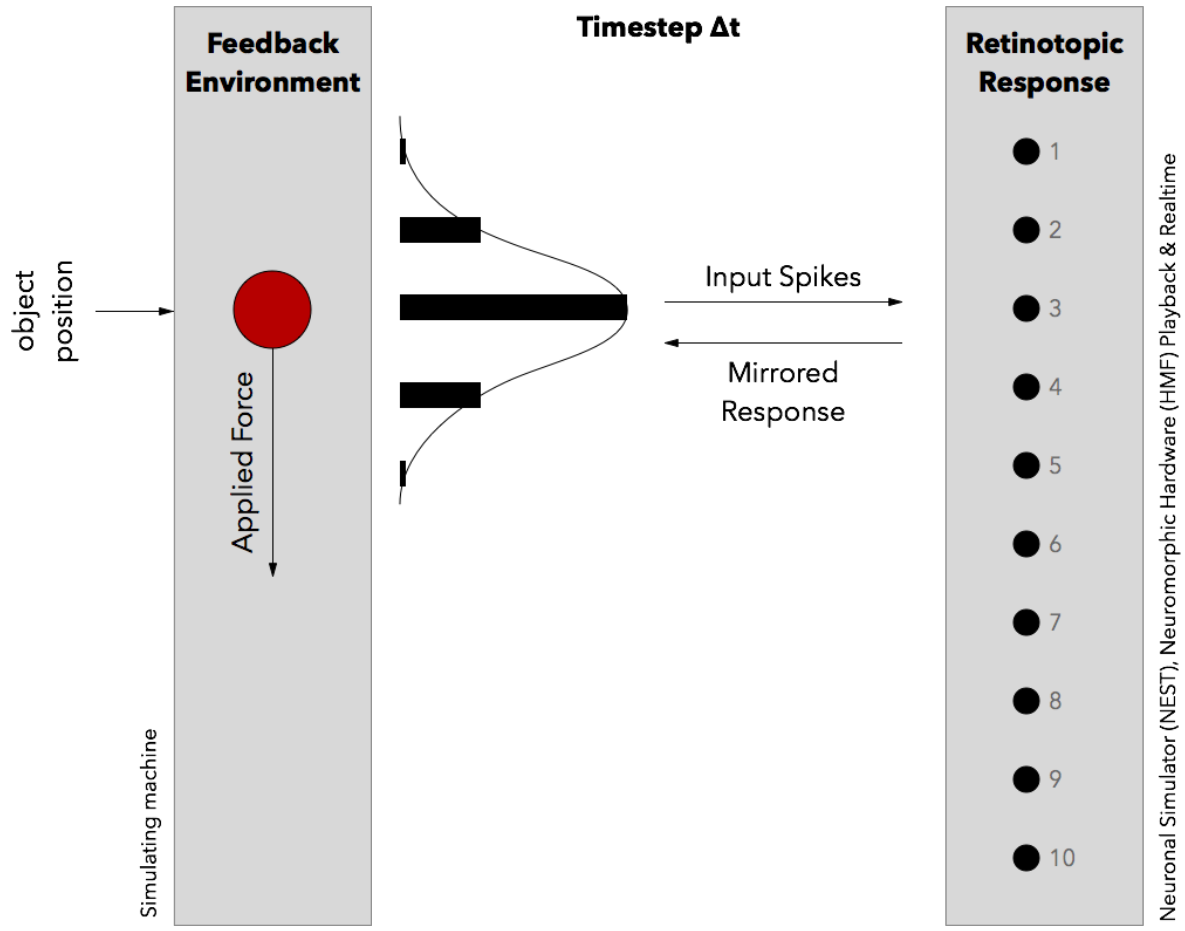

Figure 1: Setup of the Visual Feedback Experiment: an virtual environment (left) interacts with a neuronal 'network' (right).

to it, with the object's position x and a feedback constant k, moving it along the axis.

2 Software Architecture

The Visual Feedback Experiment is mostly implemented in Python, with crucial realtime components running in C++.

The `retinotopic_model` Python module provides the `Model`, `Update` and `ModelEnvironment` classes that implement the communication between the experiment's components. The `ModelEnvironment` distributes spikes between any number of `Model` objects it was provided with in the form of `Update` objects that store the input spikes and can be applied to a `Model`.

A `Model` applies updates either in *sequential* or *incremental* mode. Figure 2 shows the difference between them. In sequential mode, every update's output spikes are computed by applying the update to the given situation, whereas in incremental mode, for every update the model is reset and all previous updates are combined to one.

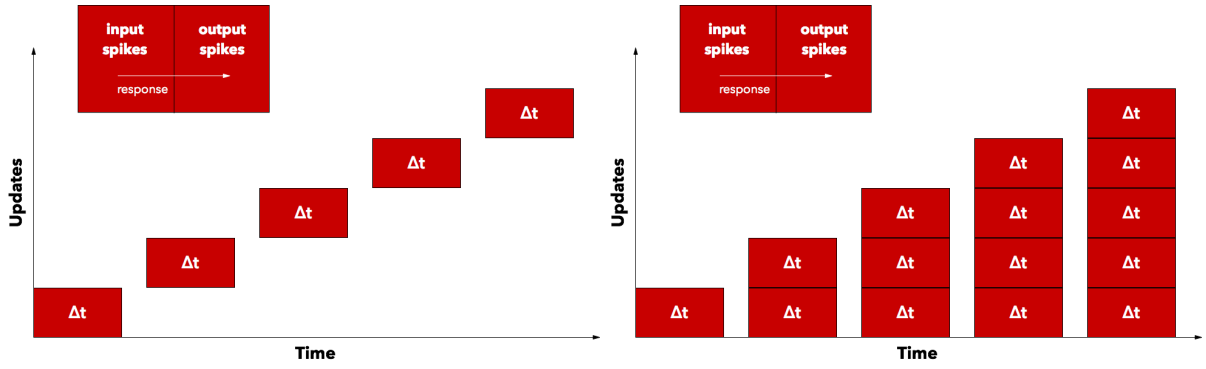

Figure 2: Experiment execution modes: the stepwise or *sequential* mode is shown on the left, the 'incremental' mode is illustrated using ever-growing simulation times (right).

There is a `Model` subclass `VisualFeedback` for the Feedback Environment, as well as for every architecture used in this experiment, namely `NeuronalSimulatorModel` for the neuronal simulator *NEST* and `NeuromorphicHardwareModel` for the *Hybrid Multiscale Facility (HMF)* neuromorphic hardware. They expose the same public interface, but differ in their implementation.

3 Neuronal Simulator Implementation

To verify the experiment's functionality and aquire data for comparison purposes, the first step of this project involved simulating the Retinotopic Response with the *NEST* neuronal simulator. Using the PyNN language, an input population and a same-size population of `IF_curr_exp` neurons are created and connected one-to-one. For the sequential update mode, the input spikes are simply queued and the simulator is run for the timestep Δt. In incremental update mode, the simulator is reset and all timesteps' input spikes up to the given time are queued. Both methods should producing the same mirrored output spikes.

Figure 3 shows the simulated time series of the object's position with different feedback constants and a time delay to produce an oscillating behaviour. Because the resulting time series is the same for both update modes, the figures don't differentiate between them.

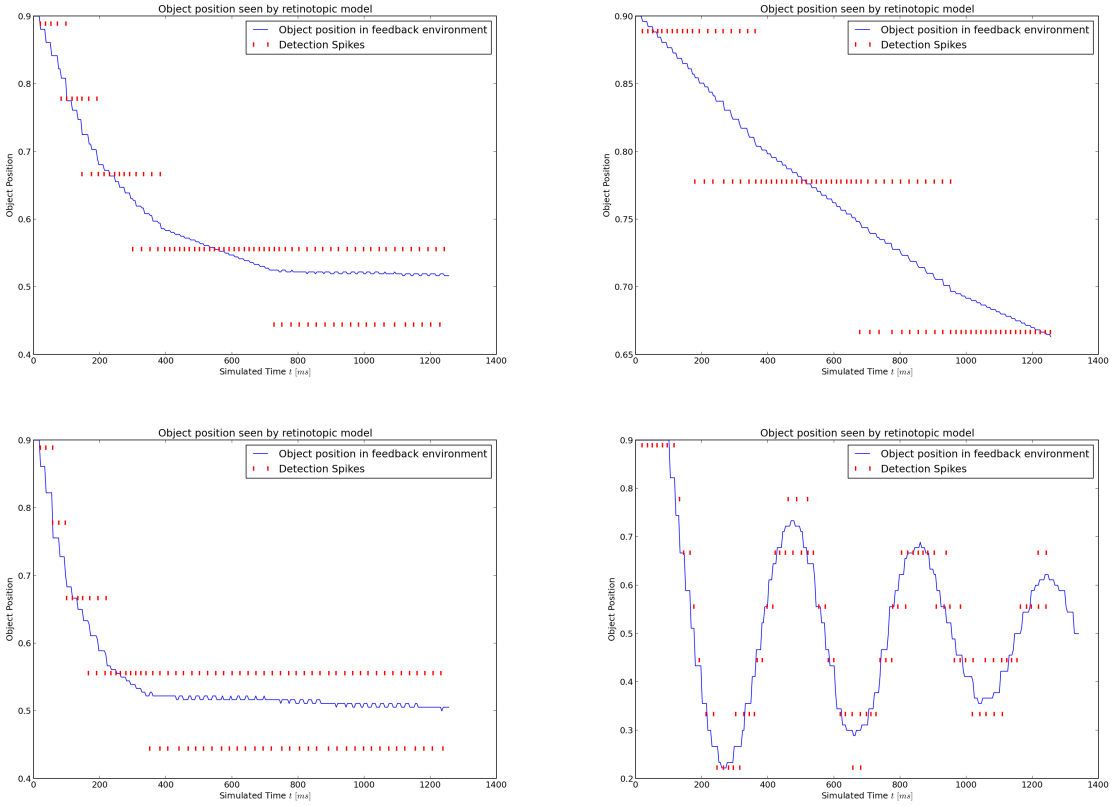

Figure 3: Time series of the object's position with different parameters (i.e. feedback constant and time delay) using a NEST simulator-based neuronal network implementation.

4 Neuromorphic Hardware Implementation

Running the experiment on the HMF neuromorphic hardware in realtime is challenging, because the simulating machine has to "keep up" with the hardware's speed. A first implementation of the Visual Feedback experiment on the hardware is therefore performed in the conventional "Playback" mode, where spikes are queued incrementally and the hardware is reset between timesteps. This also provides a testing environment for non-realtime hardware experiments.

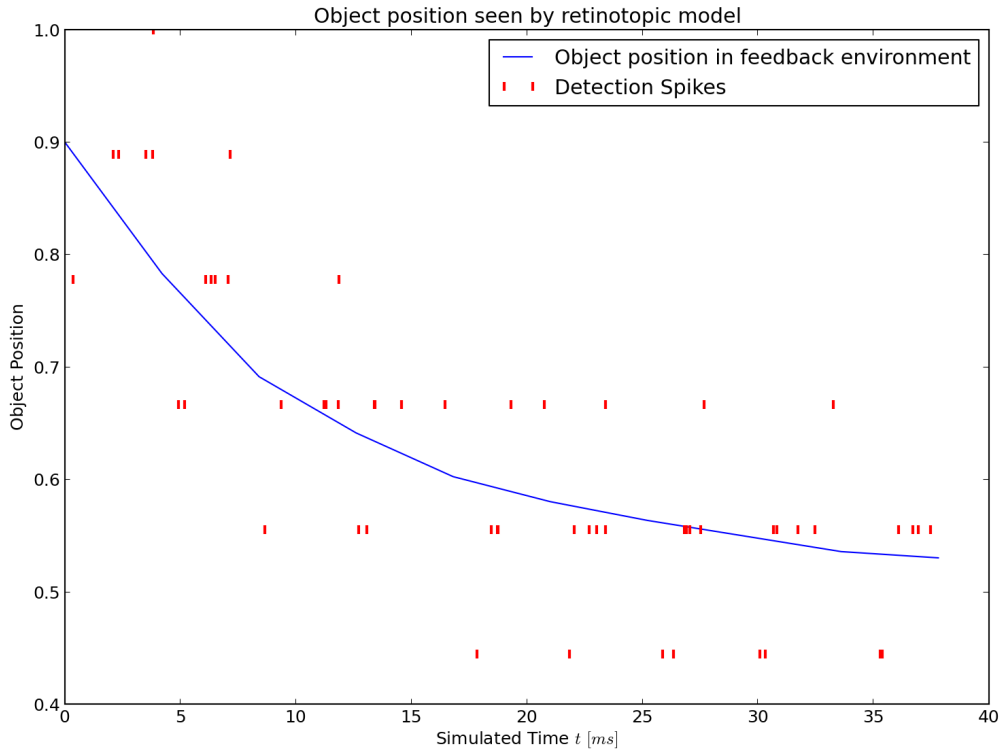

Figure 4: Object position and detection spikes for hardware in Playback mode with DNC loopback

Before using hardware neurons, the experiment was conducted with the DNCs in loopback mode. This way, the experimental setup could be tested without relying on hardware neurons to be calibrated. Instead, input spikes are simply mirrored by the DNCs. Figure 4 shows a sample run in this configuration that shows the desired behaviour similar to the outcome of the simulation.

5 Hardware Neuron Calibration

To now use hardware neurons for the experiment, we have to select a set of neurons that are calibrated to mirror incoming spikes. Because at the time of the experiment, we could not rely on a hardware calibration algorithm, we performed a test to probe a number of neurons for their reactivity with the goal to find a sufficient number with the desired behaviour to conduct the experiment.

In particular, we performed a parameter sweep for the V_t parameter, quantifying each neuron's "reactivity quality" for a given value of V_t by computing the ratio between incoming and outgoing fire rate, with a ratio of 1 being desirable. Figure 5 shows the results of this test for a number of selected neurons for illustration purposes. Note, that other parameters were set to default values (e.g. $E_l = 400$) but could easily be tested with this method as well. This could be particularly useful to put a hardware calibration algorithm to the test.

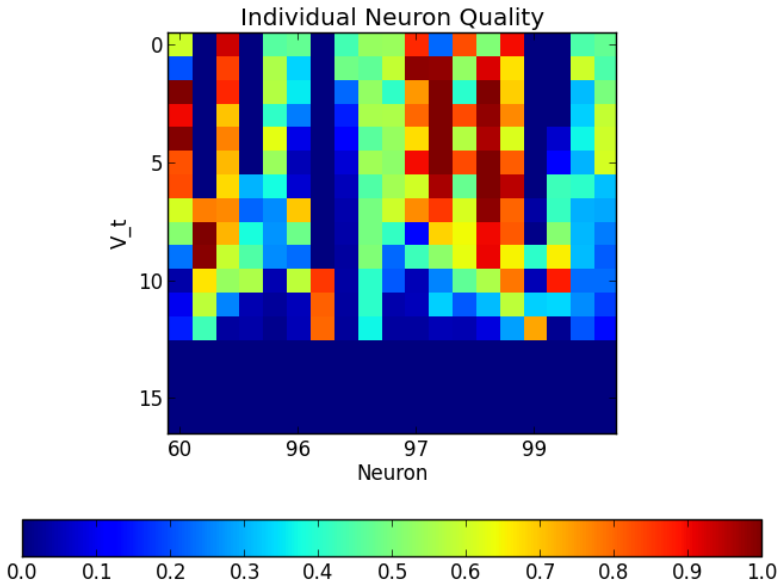

Figure 5: Neuron quality quantified by the ration between incoming and outgoing fire rate for $400 \leq V_t \leq 520$

Additionally, a linearity test was performed for neurons that showed the desired mirroring behaviour in the parameter sweep. Here, the input fire rate ν_{BEP} was varied and compared to the output fire rate ν_N to make sure they behave linearly.

These tests produced a sufficient number of neurons with the desired predictable behaviour to be used in the experiment.

6 Next Steps

The work presented here lays the foundation to conduct further experiments, some of which were already outlined above.

First, the hardware neurons found using the preliminary calibration described here should be used instead of the DNC loopback for the hardware implementation of the Visual Feedback experiment to put the hardware's conventional "Playback" mode to a test.

Then, realtime experiments can be conducted using the same neurons. The *SpiNNaker* interface already implemented in the software stack can be used for realtime communication.

Also, a hardware calibration algorithm can be tested by running the parameter sweep and linearity test on the calibrated hardware. A well-functioning calibration should drastically improve the individual neuron quality and their linearity described above. Eventually, such a calibration should be used to calibrate the hardware instead of using specific neurons for the experiment, so that arbitrary neurons can be selected for the Retinotopic Response.