Running a robotic head neuro-controller on neuromorphic hardware

PRACTICAL COURSE

submitted by
Michael Stenzel

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universitat Munchen

Supervisor:	Marcello Mulas, PhD
Final Submission:	07.07.2015

Abstract

The task for this practical course was to run the neuro-controller for a six degrees of freedom robot head on neuromorphic hardware. For this, the neuro-controller had to be ported to the latest version of the neural simulator Nengo. This version supports the use of neuromorphic hardware like the SpiNNaker board, which was used for this project.

Contents

Chapter 1

Introduction

The goal for this practical course was to run the neuro-controller for a six degrees of freedom robot head in real time on the neuromorphic computer SpiNNaker. The existing neuro-controller was written in Nengo 1.4, a neural simulator used to model cognitive capabilities of the human brain. In order to run the controller on the SpiNNaker board, the software had to be ported to a newer version of Nengo.

1.1 Project Plan

First of all to a project plan was created to get an overview about what has to be done to achieve the final goal. The plan can be seen in Figure 1.1.
First of all it was important to get familiar with the concepts of Nengo and also with the differences between the new and the old version of it. After that came the understanding of the existing code of the neuro-controller. Parallel to this the first parts of the code were already translated to the new version. Subsequent to this it had to be tested how the SpiNNaker board works in combination with Nengo. The final step was then to evaluate and test the whole system in terms of performance and functionality.

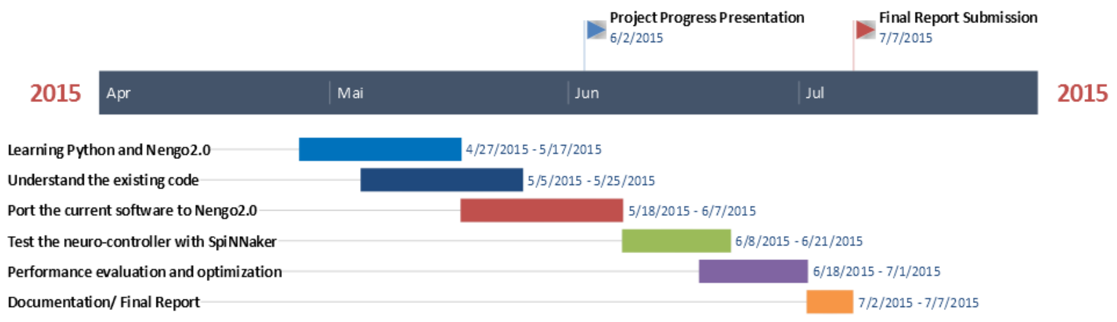

Figure 1.1: Project plan

1.2 Neuro-Controller

The existing software for the neuro-controller was written in Nengo1.4. It was created by Manxiu Zhan as part of his master's thesis [Zha14]. The software contains different biologically plausible control algorithms to control the eyes of the robot head in a human like way. There are five major movement types which will be explained in the following:

Saccades are very fast movements of the eye to direct it towards a new point of interest. These motions are ballistic which means that once the motion has started the target can't be changed. The target of a saccadic movement is determined by a saliency map. This map has also a memory effect. This prevents the eye from visiting known salient spots multiple times and to focus on a salient region for a to long time.

Smooth Pursuit allows the eyes to track a moving object. The speed of the eyes is matched to the speed of the object in order to keep its image centred on the fovea. These movements of the eye cannot be executed voluntarily without a moving stimulus.

Mircosaccades are small involuntary movements of the eye which are executed if the retina does not receive any new information, for example when looking at static scene. The movements of the eye due to the microsaccades will cause the scene to move and the retina will produce new information.

Vergence is the movement of both eyes to fixate on a common point. The goal of this motion is to minimize the disparity of the images received from the left and right eye.

Vestibulo-ocular reflex is a stabilization mechanism to compensate motions of the head. These motions are registered by the vestibular system and are compensated by moving the eyes in the opposite direction.

Chapter 2

Hardware

In this chapter the hardware used in the practical course is briefly explained.

2.1 Robot Head

The robot head, seen in Figure 2.1, consists of two eDVS boards and six servos. The eDVS boards act as eyes and can be tilted and panned independently. The complete head can also be tilted and panned. Hence the head has altogether six degrees of freedom. For each axis DYNAMIXEL MX-28 servos were used. The servos have a serial interface, which is used to send position data from the computer to the specific servo to control it.

2.2 eDVS

The eDVS board - DVS short for Dynamic Vision Sensor - is a visual sensor with a resolution of 128×128 pixels. Other than normal cameras this sensor does not provide images in form of frames. It only measures local pixel-level changes caused by movements in the scene and sends this changes in form of events to the host. This results in a much lower amount of data which has to be processed. [NST]

2.3 Gyroscope

The gyroscope is needed for the Vestibulo-ocluar reflex. It measures the rotation of the robot head. For reasons of simplicity the gyroscope integrated in the eDVS board is used for this task.

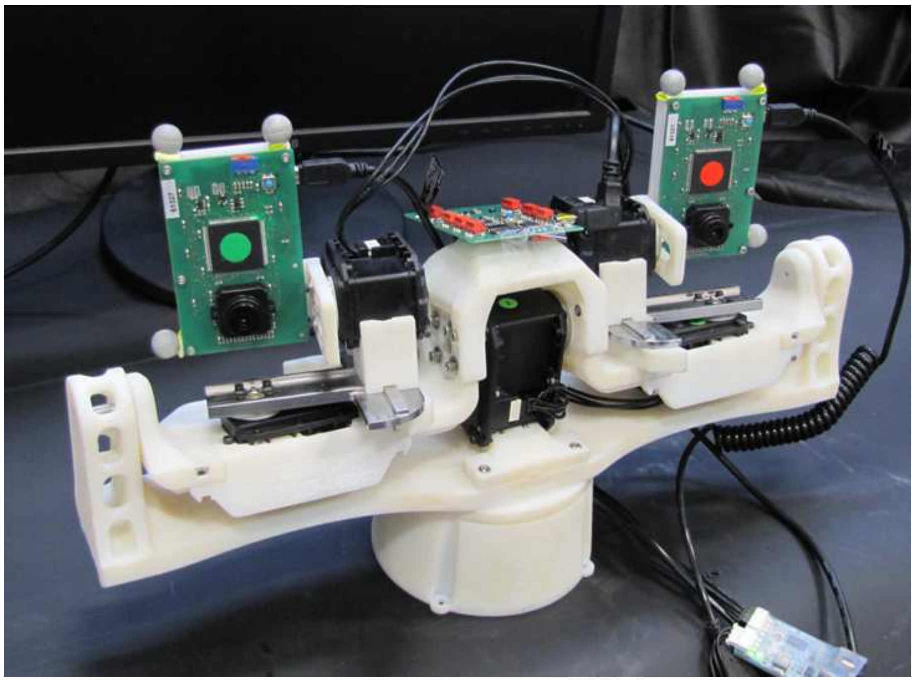

Figure 2.1: The Robot Head with the two eDVS sensors.

2.4 SpiNNaker

The SpiNNaker board seen in Figure 2.2 is a computer designed with a massively-parallel computing architecture, which is inspired by the fundamental structure and function of the human brain. This special architecture allows it to simulate huge neural networks in real time.
For this work the SpiNNaker 103 machine was used. This board has 48 nodes which provide 864 ARM processor cores. The board is supplied with 12V power supply and controlled via a 100Mbps Ethernet connection.

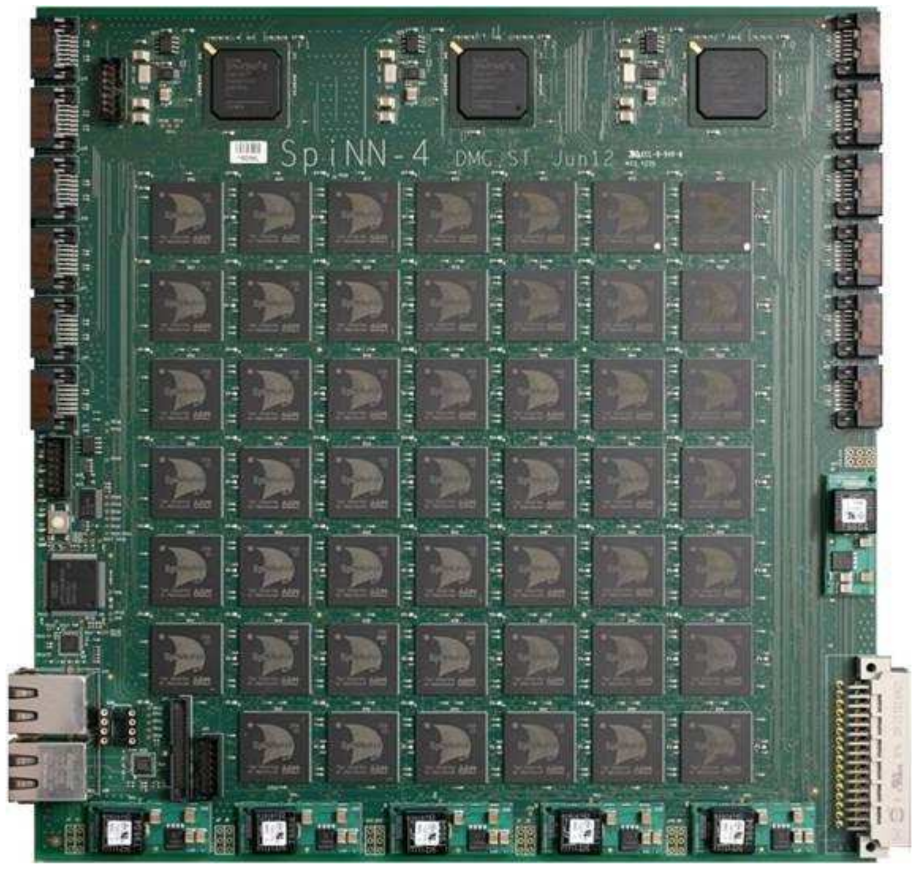

Figure 2.2: The 48-node SpiNNaker board

Chapter 3

Software Implementation

This chapter will provide a deeper insight into the funtional principle of Nengo and the rewritten neuro-controller.

3.1 Nengo

Nengo is a Python based framework for creating and simulating large-scale neural networks. It uses the Neural Engineering Framework (NEF) [CNR] to solve for appropriate synaptic connection weights to achieve the desired computation of a network. There were major changes between version 1.x and 2.x. In fact the whole framework was rewritten for version 2.x and a complete new syntax was introduced. As a result code written for Nengo1.4 can not be used with the latest version. This website [Resa] shows the detailed differences between the new and old API.
Neural networks in Nengo are build using a few basic object. The most important object are explained in the following:

Network: A network in Nengo is a model containing all other objects belonging to the neural network.

Ensemble: An ensemble is a group of neurons that represents information in form of a real valued number. Multiple ensembles can be combined into an ensemble array, for example to represent a whole set of data like the pixels of a camera image. Ensembles have, next to the number of neurons, two important parameters, the distribution of the encoders and the distribution of the intercepts. The encoders define the transform between neuron space and representational space for each neuron. The intercepts define the activity of the neurons.

Node: Nodes are used to generate inputs for ensembles. A node can provide different types of information. In the simplest case this can be a constant value.

But it can also generate mathematical functions like a sine or represent the data gathered by some sensor.

Connection: In order to carry informations from one object to the other, both elements have to be connected using the connection object. Connections can be used to connect nodes with ensembles or ensembles with other ensembles or even to create recurrent connections. It is also possible to let the connection apply mathematical functions to the transported information or use certain transformations to transmit the data. Another feature is the post-synaptic time constant of a connection. When this constant is set the connection acts as a low-pass filter.

Probe: Probes are used to obtain data from a network. By probing an object the data it generates is stored and can be visualized or used for other purposes.

Simulator: The simulator brings a network to life. Once a model with all its nodes, ensembles and probes was created it can be passed to the simulator. It will build the network and perform all necessary calculations. The standard simulator of Nengo uses the CPU of the host computer for a calculations. This is not convenient for larger networks since it is very slow. The standard simulator also has no real time capabilities. But at this point a huge advantage of Nengo comes into effect. It is possible to use the same network model with different simulators. So instead of using the standard simulator it is possible to use a simulator which uses the GPU for computation or in case of this work the SpiNNaker board.

This was just a short introduction to the concepts of Nengo. More detailed informations and examples can be found on the Nengo homepage [Resb].

3.2 Nengo SpiNNaker

In order to use the SpiNNaker board to simulate neural networks created with Nengo a special simulator is needed. This simulator is called nengo_ spinnaker and can be found here [nenc]. With this piece of software it is possible to simulate even large networks in real-time on the SpiNNaker board. Since the simulator is still under active development not all planned features are available yet. Unfortunately that means that at this point of time it is not possible to feed the simulator with new data during a running simulation. So instead all sensor data has to be recorded before running the simulation. This data is loaded as whole dataset to the SpiNNaker board and is then used for the simulation. Hopefully by the end of the year the development process will come to a point where it is possible to use real-time input data for the simulation.

3.3 The new neuro-controller

The new neuro-controller for Nengo2.1 has basically the same functionality as the old version. The networks are connected structurally in the same way as in the old controller and as described in Zhan's master's thesis [Zha14]. But there are other factors which have to be adapted. These factor are the post-synaptic time constants of the connections, the connection weights and the distribution of encoders and intercepts of the ensembles. These factors were also present in the old controller, but just taking the same values for the new controller was not sufficient. Since there are no strict rules for choosing these parameters have to be found experimentally.
A feature of the new software is that each of the movement types can be tested separately and exclusively. This makes testing and adapting new parameters for a single movement type much more convenient.
Unfortunately due to the disability of the nengo_ spinnaker simulator to run online simulations with real-time data it was only possible to verify the functionality of two of the five eye movement types. These are the Vetibulo-ocular reflex and the Microsaccades. Testing both functions showed that they basically work but that it is hard to find the correct parameters for the network. The other three movement types can not be tested, since for this functions the eDVS input, the neuro-controller and the servo motion build a control loop which only works in real-time.

Chapter 4

How to use the software

In this chapter it will be explained how to set up Nengo and nengo_ spinnaker and how to use the neuro-controller in combination with the robot head and the SpiNNaker board.
All Python files which are explicitly mentioned in this chapter can be found on the enclosed DVD in the folder 'software'.

4.1 Setting up Nengo and nengo_spinnaker

First of all the Nengo framework has to be installed. Because nengo_ spinnaker is still under development, also the development version of Nengo should be installed. For this just follow the steps under the section Developer install on this GitHub page [nenb].
After Nengo was successfully installed, nengo_ spinnaker can be installed. This can be easily done using pip. For further information regarding the setup and the configuration of the SpiNNaker board refer to the GitHub page [nenc].
Another helpful piece of software is the Nengo GUI. This tool is an interactive visualizer for Nengo which really helps the user to understand what is going on inside a network. The GUI can be found here [nena].
After everything was installed the file `sin_test.py` can be run to test if everything is working properly. This script simulates a sine wave using Nengo. When you open the file you will find a flag called 'SPINNAKER', if you set this flag the nengo_ spinnaker simulator will be used instead of the standard simulator.

4.2 Recording the sensor data

Since the SpiNNaker simulator is not able to load input data in real-time, the required data has to be recorded first. For this purpose there are two Python scripts. One is called `gyro_recorder.py`. As the name implies this script will record the gyroscope data of the eDVS board. The script contains two important variables

called `timespan` and `timestep`. `timespan` defines for how long the data is recorded and `timestep` sets the frequency at which new data is collected. At the end of the script the data is saved into a `*.pkl` file and also a plot of the recorded data will appear. It can happen that a serial error is raised when the script is started. Usually the reason for this is that the gyro client uses the wrong address to connect to the eDVS board. The address can be changed in the gyro_client.py' file.
Recording the visual data from the two eDVS boards works similarly. Running the `DVS_recoder.py` script will save the downscaled pixel data of both sensors into a single `*.pkl` file. The script also contains the variables `timestep` and `timespan`, they have the same functions as explained above. The addresses of the eDVS boards can be directly changed in the `DVS_recoder.py` script.
It is important to perform both recordings with the same values for `timespan` and `timestep` and also to use this values later for the simulation.

4.3 Running the Neuro-Controller

The complete neuro-controller can be found in the file `main.py`. This script contains several important flags at the beginning. First of all there is the `OFFLINE` flag which defines if offline data is used or if the data is acquired in real-time. Since real-time does not work yet it should be set to `True`.
There are five more flags, one for each movement type, they are called `VERGENCE`, `VESTIBULO_OCULAR_REFLEX`, `SACCADES`, `MICRO_SACCADES` and `SMOOTH_PURSUIT`. Setting one of these flags to `True` will activate the corresponding function.
One flag worth noting is left. This is the `SPINNAKER` flag. The state of this flag decides whether or not the SpiNNaker board is used for the simulation. It is recommended to use SpiNNaker since otherwise the simulation will take a very long time to finish.
Before running the simulation on the SpiNNaker board it should be made sure that it is powered up and connected to the same network as the host computer. The network connection can be tested py pinging the IP-address of the board.
It is also worth noting that the neuro-controller will only work on the 48-node SpiNNaker board. The 4-node board is by far to small to simulate the number of ensembles used for the controller.

4.4 Controlling the robot head

After the simulation of the neuro-controller the generated positon data for the servos will be saved to a `*.pkl` file. In order to use this data to control the servos of the robot head and see what the simulation actually did you have to run the `motor_playback.py` script. Make sure to set the correct timestep inside the file. The address of the servos can be changed in the `motor_client.py` file.

Chapter 5

Conclusion

At the end of this practical course it can be said that the goal was only partially achieved. The porting of the old code of the neuro-controller to the latest version of Nengo was successful. It could be also shown that it is possible to run neural-networks created with Nengo on the SpiNNaker board.
The part that is missing is the ability to run the simulation of the neuro-controller in real-time. This feature will hopefully be implemented by the developers of nengo_spinnaker in the future.

List of Figures

Bibliography

[CNR] CNRGlab@UWaterloo. Neural engineering framework. http://compneuro.uwaterloo.ca/research/nef.html. [Online; accessed 07-Juli-2015].

[Con00] Conradt, J., Tevatia, G., Vijayakumar, S., & Schaal, S. On-line Learning for Humanoid Robot Systems, International Conference on Machine Learning (ICML2000), Stanford, USA (2000).

[Con02] Conradt, J., Simon, P., Pescatore, M., and Verschure, PFMJ. Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, Int. Conference on Artificial Neural Networks (ICANN2002), p. 795-800, Madrid, Spain (2002).

[Den13] Denk C., Llobet-Blandino F., Galluppi F., Plana LA., Furber S., and Conradt, J. Real-Time Interface Board for Closed-Loop Robotic Tasks on the SpiNNaker Neural Computing System, International Conf. on Artificial Neural Networks (ICANN), p. 467-74, Sofia, Bulgaria (2013).

[nena] Nengo gui. https://github.com/nengo/nengo gui. [Online; accessed 07-Juli-2015].

[nenb] Nengo on github. https://github.com/nengo/nengo. [Online; accessed 07-Juli-2015].

[nenc] nengo_spinnaker. https://github.com/project-rig/nengo spinnaker. [Online; accessed 07-Juli-2015].

[NST] TUM Neuroscientific System Theory. edvs: Event based dynamic vision system. https://www.nst.ei.tum.de/projekte/edvs/. [Online; accessed 07-Juli-2015].

[Resa] Applied Brain Research. Converting from nengo 1.4 to nengo 2.0. https://pythonhosted.org/nengo/converting.html. [Online; accessed 07-Juli-2015].

[Resb] Applied Brain Research. Nengo. https://pythonhosted.org/nengo/index.html. [Online; accessed 07-Juli-2015].

[Zha14] Manxiu Zhan. Neural-controlled Stereoscopic Vision System. 2014.