Ruhr-Universität Bochum

Research Group
Autonomous Robotics

Schriftliche Prüfungsarbeit
für die Master-Prüfung
des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

Autonomous Learning of Sensory-Motor Transformations Involved in Looking Behavior

Christian Josef Bell

Ruhr-Universität Bochum

Research Group
Autonomous Robotics

Schriftliche Prüfungsarbeit
für die Master-Prüfung
des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

Autonomous Learning of Sensory-Motor Transformations Involved in Looking Behavior

vorgelegt von: Christian Josef Bell
Matrikelnummer: XXX XXX XXXXXX

Abgabedatum: 26.03.2013
1. Prüfer: Prof. Dr. Gregor Schöner
2. Prüferin: Dr. Yulia Sandamirskaya

Abstract

This work introduces an architecture for saccadic eye movements implemented in a C++ framework called cedar and based on methods from the dynamical field theory. The used methods are presented and it is explained how they are implemented. The system differs between horizontal and vertical saccades and combines them for oblique saccades. The timing of the saccades is as important as the interplay between the saccade system and the fixation system. Learning dynamics are used for saccadic adaptation. Results, which were measured with the architecture, like saccade trajectories, gain adaptation, etc. are shown. Also some thoughts about further research are mentioned.

Contents

1 Introduction 13

2 Methods 17
- 2.1 Dynamic Field Theory (DFT) . . . 17
- 2.2 Multi-Dimensional DNFs . . . 19
- 2.3 Projections . . . 20
  - 2.3.1 Space code to rate code . . . 20
- 2.4 Hopf Oscillator . . . 22
- 2.5 Learning Dynamics . . . 23

3 Installation and system 25
- 3.1 cedar . . . 25
- 3.2 Test environment . . . 25
- 3.3 Installation . . . 26
  - 3.3.1 Dependencies . . . 26
  - 3.3.2 cedar . . . 27
  - 3.3.3 Architecture . . . 28
  - 3.3.4 Configuration . . . 28
- 3.4 Execution . . . 28

4 Architecture and implementation 31
- 4.1 Robot Simulator . . . 32
- 4.2 Image Processing . . . 33
  - 4.2.1 Camera . . . 33
  - 4.2.2 Resize image . . . 33
  - 4.2.3 Color Conversion from RGB to HSV . . . 35
  - 4.2.4 Splitting channels . . . 36
  - 4.2.5 Thresholding hue . . . 36
  - 4.2.6 Normalization . . . 37

4.2.7 Convertion of matrix type . . . 37
4.3 Perceptual field . . . 38
4.4 Fixation System . . . 38
4.4.1 Center of visual field . . . 40
4.4.2 Projection . . . 40
4.4.3 Transformation from space code to rate code . . . 41
4.4.4 Fixation step . . . 41
4.5 Saccade System . . . 42
4.5.1 Target field . . . 42
4.5.2 Projection . . . 45
4.5.3 Transformation from space code to rate code . . . 45
4.5.4 Hopf oscillator . . . 45
4.5.5 Interneuron . . . 47
4.6 Learning System . . . 48
4.6.1 Gaussian input . . . 49
4.6.2 Interneuron . . . 49
4.6.3 Perceptual error field . . . 49
4.6.4 Projection . . . 51
4.6.5 Transformation from space code to rate code . . . 51
4.6.6 Too far or too close . . . 51
4.6.7 Learning Dynamics . . . 52
4.6.8 Select gain . . . 53
4.7 The system overview . . . 53
4.8 Auxiliary classes . . . 53
4.8.1 Buffer to file . . . 53
4.8.2 Distance error . . . 55
4.8.3 Plugin . . . 55

5 Results **57**
5.1 Saccade trajectories . . . 57
5.2 Gains accross visual field . . . 57
5.2.1 Gains from different horizontal positions . . . 57
5.2.2 Gains from different vertical positions . . . 60
5.3 Alternation between saccades and pause phases . . . 60
5.4 Adaptation experiments . . . 60
5.4.1 Error . . . 62

5.4.2 Generalization to neighboring locations 64

6 Conclusion and further research **65**

Bibliography **67**

7 Appendix **71**

7.1 DVD content 71

7.2 Source code 71

7.2.1 BufferToFile.h 71

7.2.2 BufferToThief.cpp 73

7.2.3 CenterOfVisField.h 75

7.2.4 CenterOfVisField.cpp 77

7.2.5 ColorConversionModified.h 79

7.2.6 ColorConversionModified.cpp 82

7.2.7 DistanceError.h 91

7.2.8 DistanceError.cpp 93

7.2.9 Fixation.h 95

7.2.10 Fixation.cpp 97

7.2.11 HopOscillator.h 102

7.2.12 HopfOscillator.cpp 104

7.2.13 InterStep.h 114

7.2.14 InterStep.cpp 116

7.2.15 LearningDynamic.h 119

7.2.16 LearningDynamic.cpp 121

7.2.17 PeakDetector.h 127

7.2.18 PeakDetector.cpp 128

7.2.19 Plugin.h 130

7.2.20 Plugin.cpp 131

7.2.21 RobotSimulator.h 133

7.2.22 RobotSimulator.cpp 135

7.2.23 SelectGain.h 156

7.2.24 SelectGain.cpp 158

7.2.25 ThresholdingHue.h 162

7.2.26 ThresholdingHue.cpp 164

7.2.27 TooFarOrTooClose.h 167

7.2.28 TooFarOrTooClose.cpp 169

List of Figures

2.1 General example of an activation field (extracted from [Schöner, 2008]) 17
2.2 Example of a dynamical neuronal field with gaussian input 18
2.3 Example of a multidimensional neuronal field with gaussian input 19
2.4 Illustration of the projection equation 20
2.5 Projection (x-direction) example of a gaussian input 21
2.6 Example of a horizontal projected Gaussian input transformed from space code to rate code 21
2.7 Convergence of a rate code 22
2.8 Example of a Hopf oscillation 23

3.1 Screenshot of cedars processing framework 26

4.1 Main parts of the architecture 31
4.2 Snapshot of a robot simulator environment 32
4.3 Sequence of the image processing 34
4.4 Fixation system 38
4.5 Sequence of the saccade system 43
4.6 Sequence within the Hopf oscillator step 46
4.7 Learning System 48
4.8 The complete system in detail 54

5.1 Saccade trajectories with a noise gain of 0.1 58
5.2 Saccade trajectories with a noise gain of 0.5 58
5.3 An oblique saccade with a curved trajectory 59
5.4 Horizontal saccades from different horizontal locations 59
5.5 Horizontal saccades from different vertical locations 60
5.6 Alternation between saccades and pause phases (a): $\tau_{pause} = 500$, (b): $\tau_{pause} = 1000$ and (c) $\tau_{pause} = 2000$ 61
5.7 Error in pixels of horizontal saccade (gain 0.7, tau-learning 10000) 62
5.8 Error in pixels of horizontal saccade (gain 0.5, tau-learning 10000) 63

5.9 Error in pixels of horizontal saccade (gain 0.5, tau-learning 10000) 63
5.10 Example of a learned gain after one vertical saccade 64

List of Tables

4.1 Parameter settings for the *robot simulator* step on the example of the yellow cylinder from figure 4.2 (a) and (b) and additional input noise gain 33
4.2 Parameter settings for the *resize image* step 35
4.3 Parameter settings of the *color conversion* step 36
4.4 Parameter settings of the *thresholding hue* step 37
4.5 Parameter settings of the *normalization* step 37
4.6 Parameter settings of the *convertion of matrix type* step 38
4.7 Parameter settings for the *perceptual field* step 39
4.8 Parameter settings of the *center of visual field* step 40
4.9 Parameter settings of the horizontal (left) and vertical (right) *projection* steps . 41
4.10 Parameter settings for both *space code to rate code* steps in the fixation system 41
4.11 Parameter settings of the *fixation* step 42
4.12 Parameter settings for the *target field* step 44
4.13 Parameter settings of the *peak detector* step connected to the target field 45
4.14 Parameter settings of the *space code to rate code* step in the saccade system . . 45
4.15 Parameter settings of the *Hopf oscillator* steps for both horizontal and vertical version 47
4.16 Parameter settings of the *interneuron* step of the saccade system 48
4.17 Parameter settings for the *gaussian input* step 49
4.18 Parameter settings of the *interneuron* step of the learning system 49
4.19 Parameter settings for the *perceptual error field* step 50
4.20 Parameter settings of the *peak detector* step connected to the perceptual error field 51
4.21 Parameter settings for the *learning dynamics* step 52
4.22 Parameter settings of the *select gain* step 53

List of Abbreviations

Abbreviation	Meaning
cedar	Framework for cognition, embodiment, dynamics, and autonomy in robotics
DFT	Dynamic Field Theory
DNF	Dynamical Neuronal Field
HSV	Hue, Saturation, Value
RGB	Red, Green, Blue
ROI	Region Of Interest

1 Introduction

This masterthesis was written at the Institut für Neuroinformatik, Bochum. Its goal was to build an architecture for eye movements - especially in this case, the focus was on saccades - based on the dynamical field theory (section 2.1) and implemented in the framework cedar (section 3.1). Additionally, a learning method for saccadic adaptation should be implemented. This chapter gives an introduction to this work and a classification to other works.

[Enderle, 1994] defines five types of eye movements[1]:

- vestibular ocular movements
- optokinetic eye movements
- vergence eye movements
- smooth pursuit eye movements
- saccadic eye movements

The vestibular ocular movements hold a visual target on the center of the visual field during head movements by moving the eyes in the opposite direction of the head movement.

The optokinetic eye movements occur, when a target moves through the visual field. The eyes follow this target until the target leaves the visual field. Then the eyes jump back to the starting position.

Vergence eye movements hold a visual target on the center of the retinas of both eyes. If the target moves closer, the eyes converge, and if the target moves far away, the eyes diverge.

Smooth pursuit eye movements happen, when the eyes follow a slowly moving target.

Saccadic eye movements are the eye movements, which were important for this masterthesis. Therefore they will be discussed in more detail. Saccadic eye movements or so-called saccades are rapid eye movements, which occur, when a target appears somewhere in the visual field and the eyes jump to that target.

[Findlay and Walker, 2012] list four categories of saccades:

[1] Hint: All these five types of eye movements are nicely demonstrated in short animations from Mark Bolding at http://www.youtube.com/user/mbolding

- Reflexive and voluntary saccades
- Delayed saccades and memory-guided saccades
- Anti-saccades
- Microsaccades

Reflexive and voluntary saccades are saccades which are executed, when a new target appears on the visual field (reflexive) or when a decision was made to do a saccade (voluntary), for example by getting the instruction “look to the right”.

> “In a strict sense, however, all saccades are essentially voluntary in nature as an observer can always decide not to move the eyes.” [Findlay and Walker, 2012]

Delayed saccades have a bigger delay, because of the ability to suppress a response to a visual stimulation. In the case of memory-guided saccades there is a very briefly stimulation, which leads to a saccadic movement to a remembered location.

Anti-saccades are saccades in the opposite direction of the visual target.

Microsaccades occur during fixation. They are very short saccades (<0.5°) and necessary to prevent fading.

Saccades rarely take longer than 80 ms, which is an extremely brief execution, so visual feedback during the saccade does not play a role in controlling the saccade [Guthrie et al., 1983, Ethier et al., 2008].[2]

In this work horizontal and vertical saccades are treated separately, which is consistent with other scientific work:

> “Clinical, anatomical, and physiological evidence all point to separate brain-stem substrates for horizontal and vertical saccades, at least at the immediate prenuclear level.” [Grossman and Robinson, 1988]

If an oblique saccade should be executed, a horizontal and a vertical saccade is performed separately. They only start at the same time. In a cartesian coordinate system (as it is used in this work) the resulting oblique saccade has a curved trajectory [Grossman and Robinson, 1988].

Normally, saccades alternate with fixation. In this work this alternation is implemented as a pause phase (see subsection 4.5.4). During the pause phase the fixation is active. [Wilimzig et al., 2006] use an initiation field for the interaction between fixation and saccades.

[2] Because of observation reasons for the user of the architecture the duration of saccades in this work is set to somewhat longer to see what is happening (see subsection 4.5.4).

If there are several visual targets, the system has to select, which target should be the saccadic target. If two targets are very close to each other, the saccadic landing position is in the middle of both targets [Wilimzig et al., 2006].

During lifetime of a human being the physiological conditions change, because of aging and development. This obviously has an impact on the oculomotor system. So "[...] a continuous recalibration of the oculomotor system by visuo-motor adaptation mechanisms [...]" [Pelisson et al., 2010] is needed.

> "If a saccade is inaccurate, a visual error teaching signal adjusts the adaptive weights to reduce the saccadic error." [Gancarz and Grossberg, 1998]

How accurately a saccade is, is defined by: "[...] gain, the ratio between the distance travelled by the eyes (saccade amplitude) and the distance of the target from the initial eye position (desired saccade amplitude)." [Pelisson et al., 2010]

This gain learning process takes place in the cerebellum [Grossberg, 1969, Grossberg and Kuperstein, 1986, Gancarz and Grossberg, 1998], by iteratively updating the relationship between the location of a visual target and the motor commands, which are necessary to look at this target [Pelisson et al., 2010].

Chapter 2 introduces the methods used for the architecture, chapter 3 gives a short description of how to install and execute the system, chapter 4 describes the architecture modules and their implementations by a chronological order, chapter 5 shows some important results and finally chapter 6 summarizes the work and gives a short view into possible further research. The appendix 7 lists the dvd content and the source code. The enclosed DVD contains among other things a colored PDF of this thesis.

2 Methods

This chapter gives a short introduction into the methods used in this thesis. Subsection 2.1 describes the Dynamic Field Theory, subsection 2.2 addresses multi-dimensional dynamical neuronal fields, subsection 2.3 explains projections, subsection 2.4 gives information about the Hopf oscillator and finally subsection 2.5 describes learning dynamics.

2.1 Dynamic Field Theory (DFT)

Figure 2.1 (extracted from [Schöner, 2008]) shows a two-dimensional activation field with a metric content on the x-axis and an activation on the y-axis. For example, the activation could be the visual input coming from a camera and the metric content could be the horizontal location of this visual input. Assuming that the camera gets an input of a yellow cylinder (like in this thesis in subsection 4.1), that the hue value is the extracted information of the camera input and this information is the input of the field, then there is an activation in the field and the field gets a peak at the horizontal position of this input. If the yellow cylinder disappears than the peak decays.

Equation 2.1 is based on [Amari, 1977] and [Schöner, 2008] and it describes the neuronal field equation extended by a noise kernel:

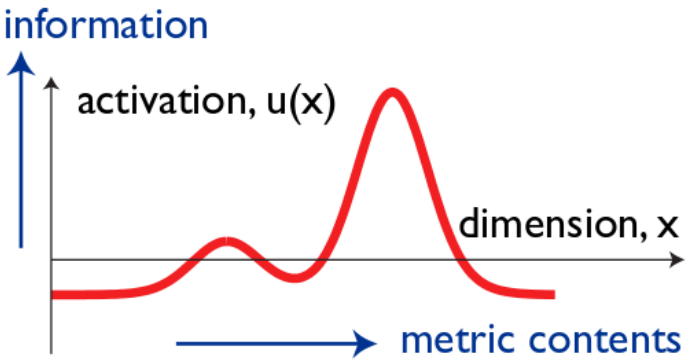

Figure 2.1: General example of an activation field (extracted from [Schöner, 2008])

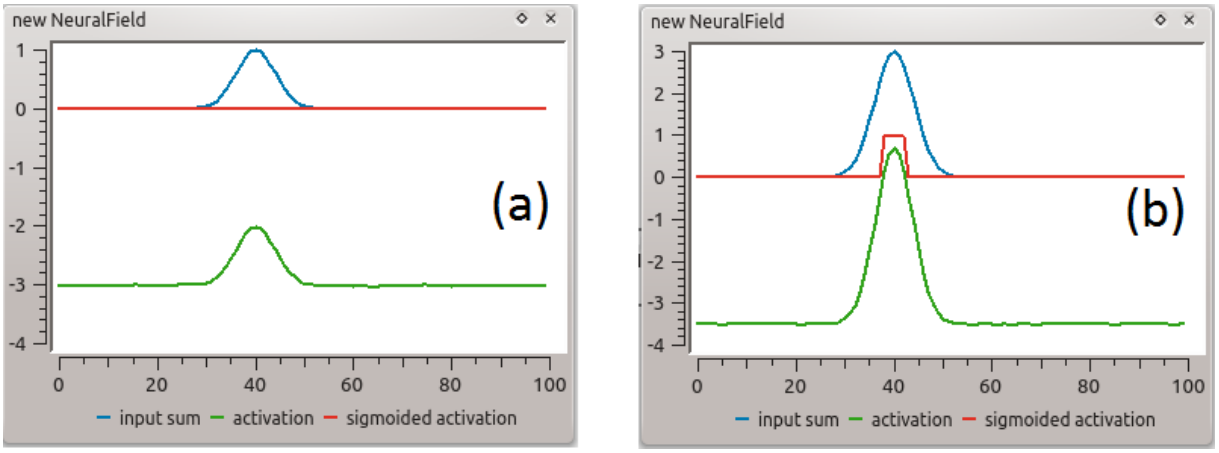

Figure 2.2: Example of a dynamical neuronal field with gaussian input

$$\tau \dot{u}(x,t) = -u(x,t) + h + \int w(x-x')\sigma(u(x',t))dx' + S(x,t) + c_\eta \eta \tag{2.1}$$

where

- τ is the time scale parameter
- $u(x,t)$ is the activation field
- $h < 0$ is the constant resting level
- $w(\triangle x)$ is the interaction kernel
- $\sigma(u)$ is the sigmoidal nonlinear threshold funtion of equation 2.3
- $S(x,t)$ is the input function
- c_η is the constant noise gain
- η is the noise

The equation 2.1 will be explained on the basis of figure 2.2 (referring to [Richter, 2011]). The green line is the activation $u(x,t)$, the blue line is the input $S(x,t)$ and the red line is the sigmoidal nonlinear threshold function $\sigma(u)$. In figure 2.2 (a) there is a small bump of activation at $x = 40$ due to the gaussian input also at $x = 40$, but the input is not strong enough to exceed the threshold. Also, it can be seen that the activation level is resting at $h = -3$. In figure 2.2 (b) the gaussian input is somewhat raised. The input now is strong enough to pass the threshold resulting in a so called supra-threshold peak of activation. Furthermore, the global inhibition

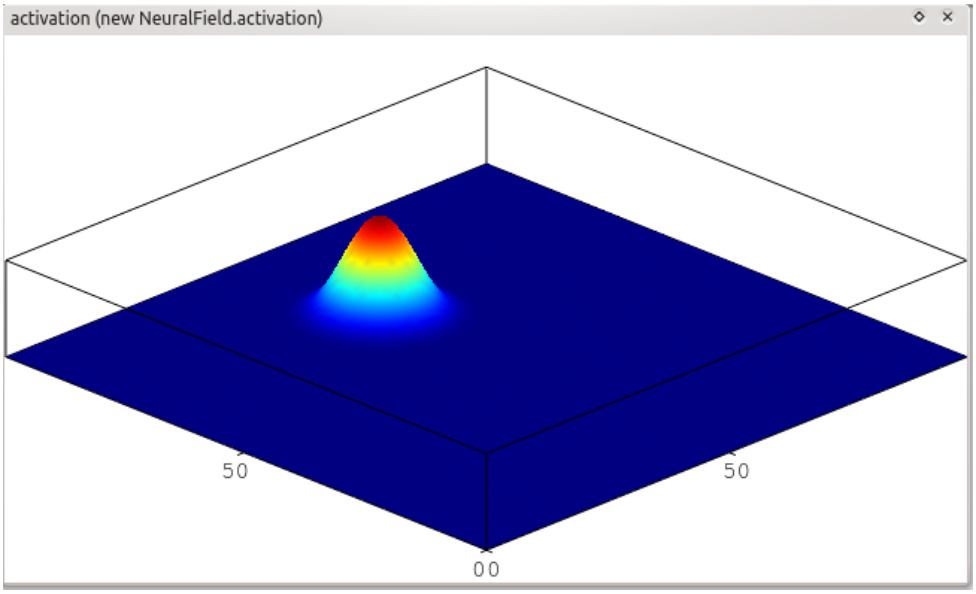

Figure 2.3: Example of a multidimensional neuronal field with gaussian input

decreases the rest of the activation field. The small fluctuations in figure 2.2 (a) and (b) come from the noise.

Equation 2.2 is the interaction kernel with the global inhibition w_{inh} and the local excitation w_{exc}.

$$w(\Delta x) = w(x - x') = w_{inh} + w_{exc} \exp\left(-\frac{(x-x')^2}{2\sigma_i^2}\right) \tag{2.2}$$

Equation 2.3 is an exemplary sigmoidal nonlinear threshold function.

$$\sigma(u) = \frac{1}{1 + \exp\left(-\beta\left(u - u_0\right)\right)} \tag{2.3}$$

2.2 Multi-Dimensional DNFs

In section 2.1 the dynamical neuronal field was just dependent on the metric dimension x and on time t. But of course, an dynamical neuronal field can be multi-dimensional. Figure 2.3 shows an example of a multi-dimensional dynamical neuronal field with gaussian input. Here, there are the two metric dimensions x and y and time t.

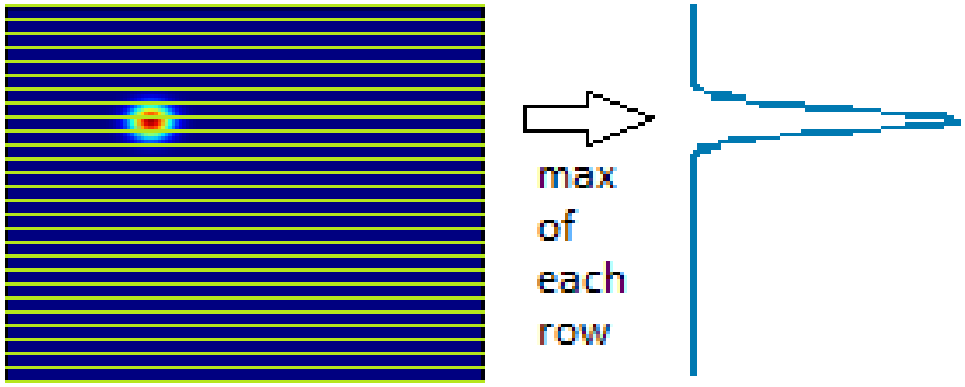

Figure 2.4: Illustration of the projection equation

2.3 Projections

A projection is a transformation from a m-dimensional world to a n-dimensional world, thus $\mathbb{R}^m \rightarrow \mathbb{R}^n$. In this thesis only a field reduction from 2-dimensional fields to 1-dimensional fields is needed. There are several ways to obtain this reduction. One is the maximum value method (used in subsection 4.4.2) as in equation 2.4, which is a 2D to 1D projection that calculates the maximum value for each vector x_i.

$$\mathbf{X} = \begin{pmatrix} max_1(\mathbf{x}_1) \\ \vdots \\ max_n(\mathbf{x}_n) \end{pmatrix}, n \varepsilon \mathbb{N} \tag{2.4}$$

Figure 2.4 is a schematic illustration of formula 2.4 for a 2D to 1D vertical projection. The field is divided into many rows. For each row the maximum is calculated, which then is saved to the corresponding element of the vector. Figure 2.5 shows an example of a reduced neuronal activation field from 2D to 1D (x-direction).

2.3.1 Space code to rate code

A space code to rate code transformation is also a dynamical system with equation 2.5. To receive information about where the maximum of a field is, a space code to rate code transformation can be accomplished.

$$\dot{v} = v + \frac{dt}{\tau} \cdot (-sv + o) \tag{2.5}$$

where

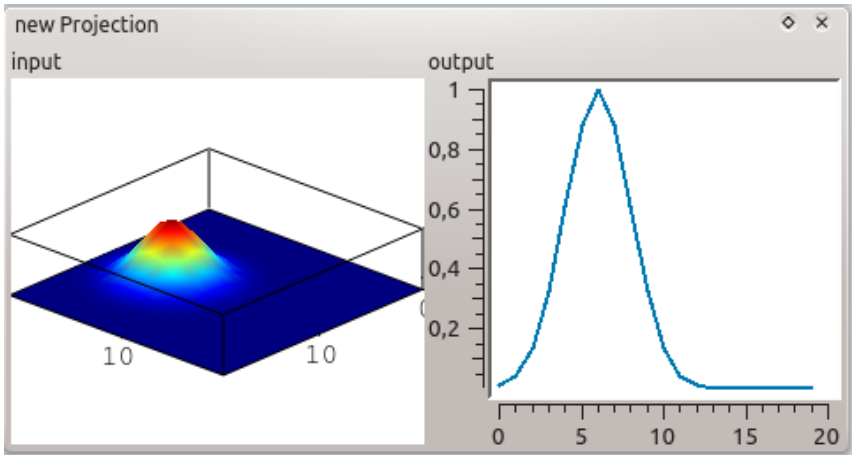

Figure 2.5: Projection (x-direction) example of a gaussian input

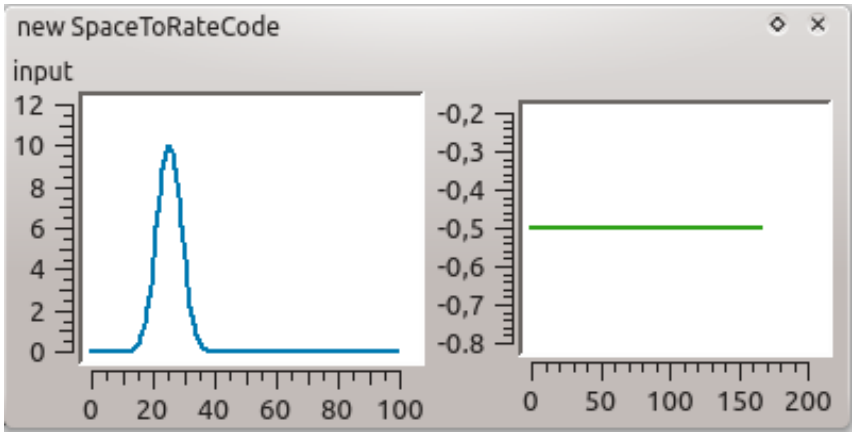

Figure 2.6: Example of a horizontal projected Gaussian input transformed from space code to rate code

- τ is the time scale parameter
- s is the slope, with $s = \sum S(x,t)$
- o is the offset, with $o = \sum (S(x,t)\, r(x))$
  - $S(x,t)$ is the input field
  - $r(x)$ is a ramp function

Figure 2.6 shows an example of a 1D gaussian input. The input field has a size of 100 and the center of the Gaussian is at 25. It can be seen very well that the rate code value settles at -0.5 (rate code is in the range [-1,1]), which corresponds with the position of the input field. Especially in this situation the value -0.5 means that the maximum value of the input field is left from the center (because it is negative) and approximately at the half of the left side. In more

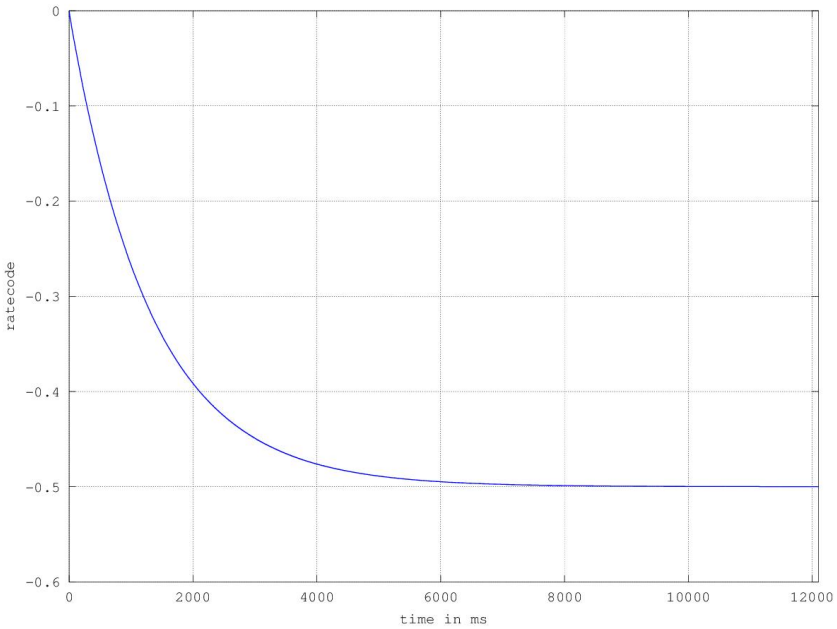

Figure 2.7: Convergence of a rate code

general terms, this means the closer it is to the center, the smaller is the rate code value and the sign determines the spatial direction. In Figure 2.7 the parameter tau is set to 10000 to obtain a very slow change. So it is possible to show how the rate code value of the example from figure 2.6 converges to -0.5.

2.4 Hopf Oscillator

Oscillators are widely used to encode timing [Large and Kolen, 1994]. In this work a Hopf oscillator is used to encode velocity and timing of saccades. A Hopf oscillator is a dynamical system, which can form a stable limit cycle around the parameter α. There are various versions of a Hopf oscillator, but in this thesis equation 2.6 is used (see also subsection 4.5.4 for the implementation characteristics of this equation). For a detailed explanation of Hopf oscillators see [Kuznetsov, 2006].

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\tau} c_{on} \left(\begin{pmatrix} \gamma\alpha & -\omega \\ \omega & \gamma\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \gamma\left(x^2+y^2\right) \begin{pmatrix} x \\ y \end{pmatrix} \right) - (1-c_{on}) \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.6}$$

where

- τ is the time scale parameter
- γ is a fixed parameter

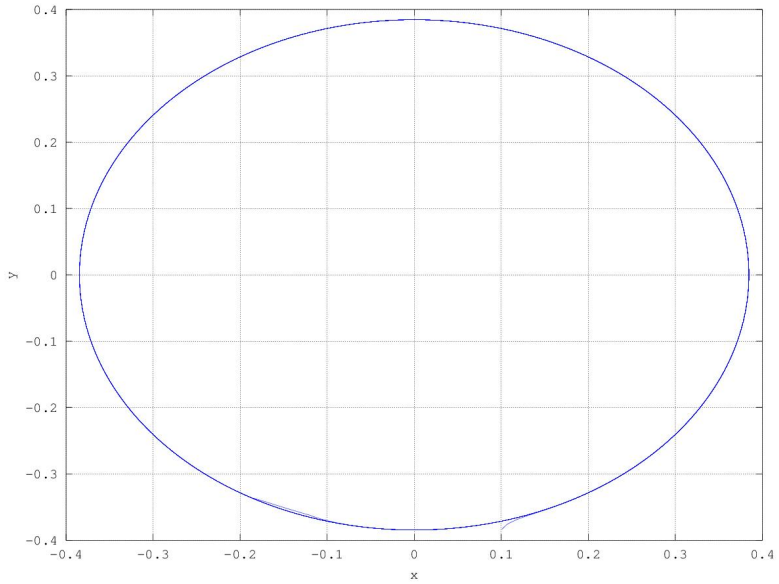

Figure 2.8: Example of a Hopf oscillation

- ω is the frequency
- $\alpha = \mu^2$ is the amplitude
- c_{on} is an activation neuron. If c_{on} is 1, the oscillation part is active and if c_{on} is 0, the oscillation is deactivated.

Figure 2.8 shows an hopf oscillator example with an amplitude of $\alpha = 0.384538$.

2.5 Learning Dynamics

Equation 2.7 shows a dynamical learning system.

$$\tau \dot{\mathbf{w}}_{gain,d}(x,t) = c_{learning} PER_d \delta_d(u_{target}(x,t)) \tag{2.7}$$

where

- τ is the time scale parameter
- $\dot{\mathbf{w}}_{gain,d}(x,t)$ is the resulting gain for dimension d
- $c_{learning}$ is an activation neuron, which is 1, if the system has to learn, 0 otherwise
- $PER_d = r_{pe,d} r_{tar,d}$ is the perceptual error rating for dimension d, where

  - $r_{pe,d}$ is the rate code of the perceptual error field with dimension d
  - $r_{target,d}$ is the rate code of the target field with dimension d and a peak at the origin position of the saccade

- $\delta_d\left(u_{target}\left(x,t\right)\right)$ is the target field projected to dimension d with a peak at the origin position of the saccade

If the system is activated to learn, that means, $c_{learning}$ is set to 1 (see also subsection 4.6.7 to get an explanation about the activation of this neuron), it multiplies the perceptual error rating with the target field. The target field has a peak at the origin of the saccade, so the system just learns the new weights at the position of the saccade. This does not only happen for one value at the maximum of the peak, but also for the neighboring values.

The perceptual error rating is a result of the multiplication of the rate code of the perceptual error field and the rate code of the target field. This simple multiplication has the great advantage that it cannot only tell, how far away the saccade landed from the center, but also if the saccade was too short or too long. That means in detail: If the perceptual error rating is positive, then the saccade was too short. If the perceptual error rating is negative, then the saccade was too long. Within the dynamical learning system this helps to decrease or increase the gain due to the sign of the perceptual error rating.

3 Installation and system

This chapter describes the test environment including the used hardware and software and explains how to install and setup the framework and the architecture.

3.1 cedar

Cedar is a library for C++. The name cedar is the abbreviation of "framework for *c*ognition, *e*mbodiment, *d*ynamics, and *a*utonomy in *r*obotics". It was developed by the "Institut für Neuroinformatik"[1] in Bochum, Germany. In this thesis cedars processing framework is used for the implementation of our architecture. Figure 3.1 shows a screenshot of cedars processing framework. In this processing framework each software module is called processing step. Therefore, I will also speak about steps in the remaining part of this thesis, when I am talking about the implemented software modules. Each step can have input slots and output slots. An output slot of one step can be connected to one or more input slots of other steps. Some steps accept more than one input on one slot. Please refer to [Lomp et al., 2012] and [Institut für Neuroinformatik, 2011] for a detailed description of cedar, its features, some programming tutorials and of course the software itself.

3.2 Test environment

The test environment consisted of an Acer Aspire 7741G with Kubuntu 12.10[2]. The main hardware specifications were:

- Intel Core i3-330M processor with 2.13 GHz and 3 MB L3 cache
- ATI Mobility Radeon HD 5470
- 4 GB RAM

[1] Url: http://www.ini.rub.de
[2] Url: http://www.kubuntu.org

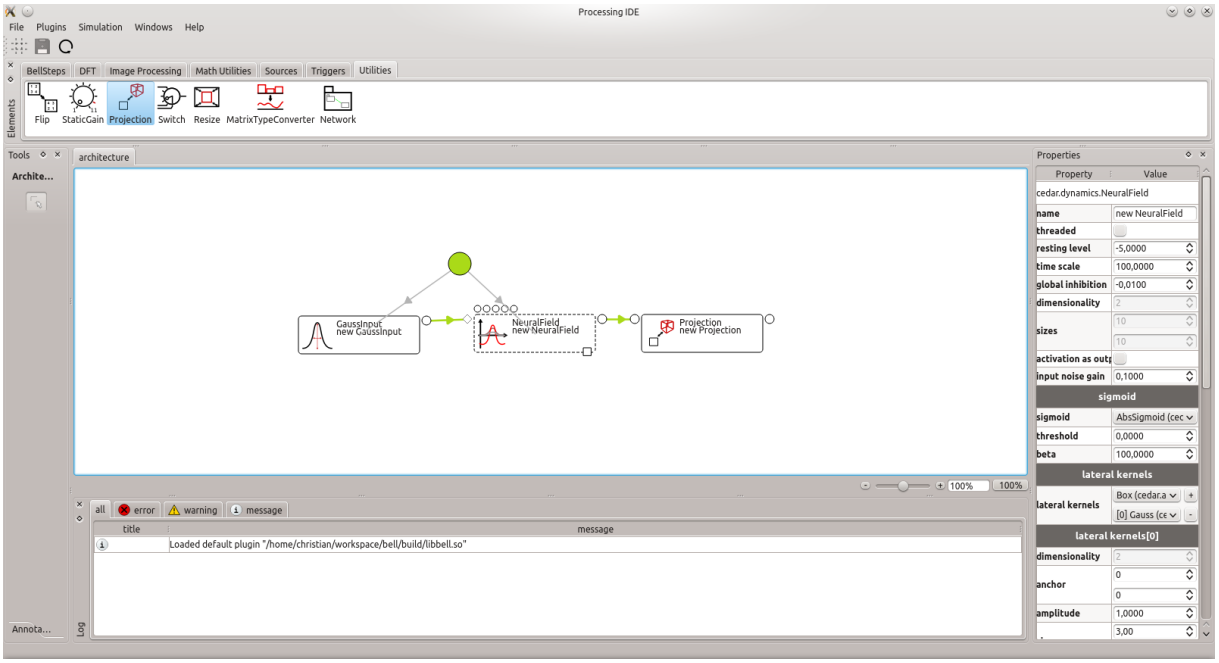

Figure 3.1: Screenshot of cedars processing framework

- 320 GB HDD

Eclipse[3] Juno (Service Release 1) was used as the development environment and Octave[4] version 3.6.4 was used for plotting most of the graphs in this thesis.

3.3 Installation

This section describes how to install and setup cedar and the architecture. It is assumed that the system will run on Kubuntu 12.10 and that basic skills in Linux are existing.

3.3.1 Dependencies

cedar has a few dependencies, which need to be installed[5] first before cedar can be installed. These are the minimum requirements:

- CMake
- Boost 1.47 or higher (except version 1.49)

[3] Url: http://www.eclipse.org
[4] Url: http://www.gnu.org/software/octave/index.html
[5] A detailed version of the complete installation process for subsection 3.3.1 and 3.3.2 can be found under https://bitbucket.org/cedar/unstable

- OpenCV 2.2 or higher
- Qt 4.6.2 or higher
- qwt 5.2.1 or higher
- qwtplot3d 0.3 or higher
- libqglviewer 2.3.6 or higher

These dependencies can be manually installed via, for example, the package manager or by using the *cedar-dependencies.deb* package following these steps (make sure that gdebi is installed):

```
wget https://bitbucket.org/cedar/dependencies/downloads
       /cedar-dependencies.deb
sudo gdebi cedar-dependencies.deb
```

3.3.2 cedar

This subsection describes how to get the correct version of cedar and how to compile it. First of all cedar has to be cloned from the repository by executing the following command:

```
hg clone https://bitbucket.org/cedar/unstable -r 2697[6] [cedar directory]
```

Now execute the following commands step-by-step:

```
cd [cedar directory]
cp cedar.conf.example cedar.conf
mkdir build
cd build
cmake ..
make -j8
```

[6]The architecture in this thesis is tested up to revision 2697 of cedar.

3.3.3 Architecture

The architecture has to be cloned by the following command:

```
hg clone https://bitbucket.org/CJb3LL/bell [architecture directory]
```

Now execute the following commands:

```
cd [architecture directory]
cp project.conf.example cedar.conf
vim cedar.conf <----- "Change CEDAR_HOME to the correct directory!"
mkdir build
cd build
cmake ..
make -j8
```

3.3.4 Configuration

In the root folder of the cloned project are two files:

- schunk_head.json
- schunk_camera.json

These files need to be copied to *[cedar directory]/resources/configs* by following these steps:

```
cd [architecture directory]
cp schunk_head.json [cedar directory]/resources/configs/
cp schunk_camera.json [cedar directory]/resources/configs/
```

3.4 Execution

To start the architecture just type in these commands into the console:

```
cd [cedar directory]/bin
./processingIDE
```

The processing framework opens. Now follow these steps:

1. Click on *Plugins*

2. Click on *Load Plugin ...*
3. Click on *browse...* and browse to *[architecture diretory]/build*
4. Select the file *libbell.so*
5. Click on *Open*
6. Click on *Ok*

The architecture plugin is loaded. After that the architecture.json file has to be loaded:

1. Click on *File*
2. Click on *Load ...*
3. Click on *browse...* and browse to *[architecture directory]*
4. Select the file *architecture.json*
5. Click on *Open*

Now everything is loaded. To run the architecture, first start the Robot simulator by right-clicking on the Robot Simulator step, left-clicking on *actions* and *startVisualSimulator*. The windows from figure 4.2 open. The last step is to start all triggers (the circles) by right-clicking on the triggers and left-clicking on *start*.

4 Architecture and implementation

This chapter describes the architecture of the system which was implemented in this work. Figure 4.1 shows a graphic where the architecture is divided into five main parts (Robot Simulator, Image Processing, Saccade System, Learning System and Fixation System) and a perceptual field.

The following sections describe these parts. As the robot simulator is the start and end point of one cycle the first section 4.1 describes the robot simulator. Then I address the image processing in section 4.2, the perceptual field in section 4.3, the fixation system in section 4.4, the saccade system in section 4.5 and section 4.6 with a description of the learning system. Finally, section 4.7 shows a complete overview of the architecture.

As a reminder: In cedar every node of the architecture is called step, so in this work I also talk about steps. Every subsection contains a description of the respective step, as well as some words of the implementation and a table with the mainly used parameter settings. If a subsection does not include such a table, then the step has no parameters.

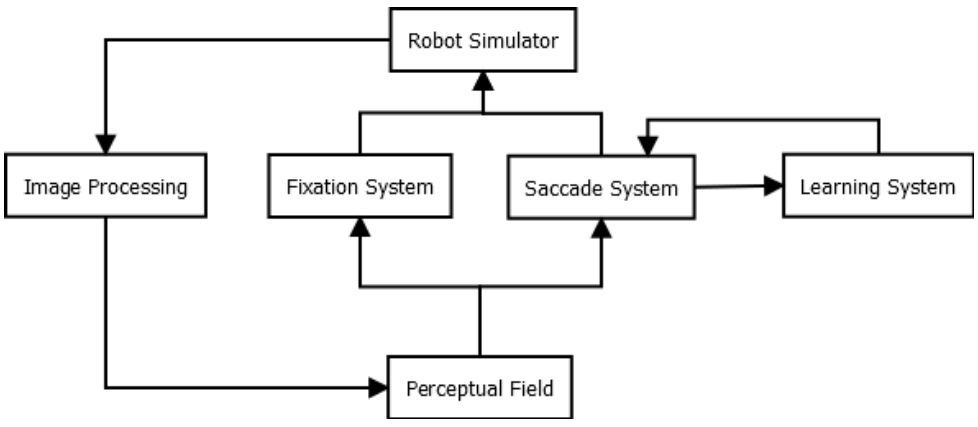

Figure 4.1: Main parts of the architecture

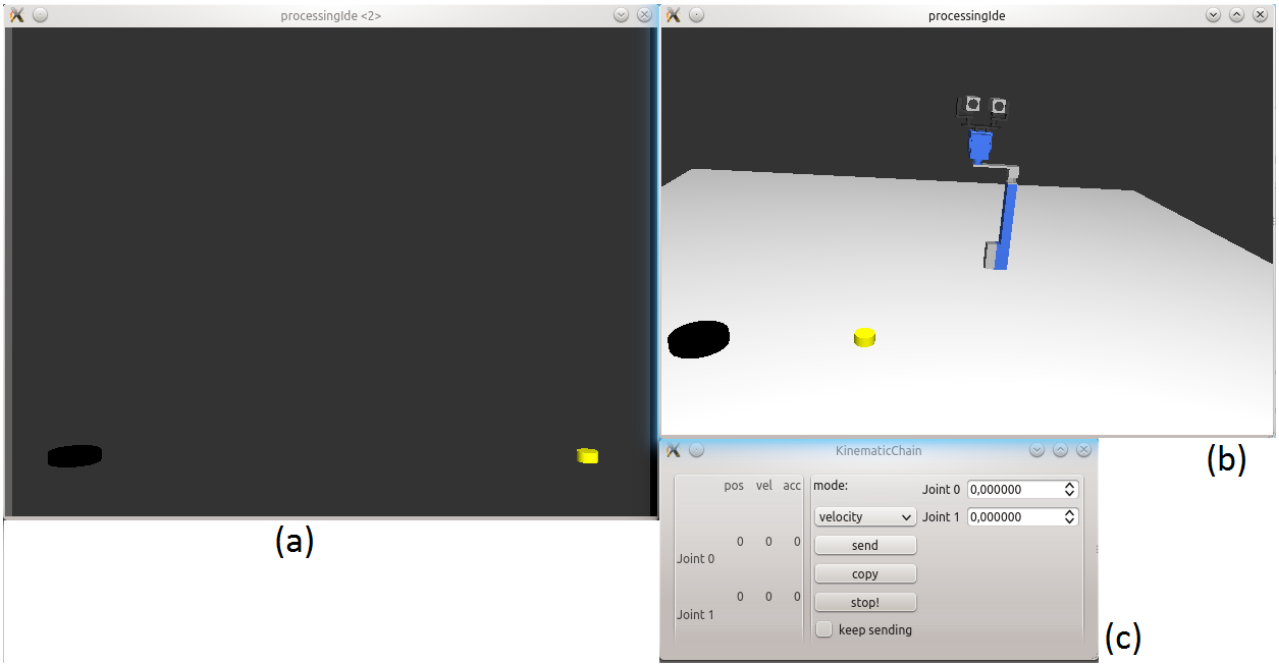

Figure 4.2: Snapshot of a robot simulator environment

4.1 Robot Simulator

The *robot simulator* step is the starting point of the architecture. It integrates *cedar::devices::robot::SimulatedKinematicChain*. The *robot simulator* step has three input slots, the first two receive the velocities from the *Hopf oscillator* steps (see subsection 4.5.4) and the last one receives the velocity vector from the *fixation* step (see subsection 4.4.4). Since the two velocity values from the *Hopf oscillator* steps are separated from each other, they are first combined to a velocity vector. Then the velocity vector from the *fixation* step is added. Note that always one of the both vectors is zero, because either the saccade system is active or the fixation system is active. Formula 4.1 shows the resulting equation. The negative sign of the vertical velocities is due to the coordinate system of OpenCV.

$$\mathbf{v} = v_{saccade} + v_{fixation} = \begin{pmatrix} v_{saccade,x} \\ -v_{saccade,y} \end{pmatrix} + \begin{pmatrix} v_{fixation,x} \\ -v_{fixation,y} \end{pmatrix} \tag{4.1}$$

This velocity vector is send to the *SimulatedKinematicChain* which moves the two chains (pan and tilt) of the camera within the simulation environment. Figure 4.2 shows an example screenshot of the simulation environment. The first window (Figure 4.2 (a)) is the camera input or in other words, it is the input of the image processing (see section 4.2). The second window (Figure 4.2 (b)) is a view at the simulation environment and the third window (4.2 (c)) is a

Parameter	Value
Cylinder Red Value	1
Cylinder Green Value	1
Cylinder Blue Value	0
Cylinder Diameter	0.02
Cylinder Height	0.02
Cylinder X Position	0.5
Cylinder Y Position	1.5
Cylinder Z Position	0.5
input noise gain	0.01

Table 4.1: Parameter settings for the *robot simulator* step on the example of the yellow cylinder from figure 4.2 (a) and (b) and additional input noise gain

control widget for manually controlling the chains.

The simulation environment has three implemented cylinders, which can be configured by the parameters of Table 4.1 (the values in this table are an example and correspond to the yellow cylinder in Figure 4.2 (a) and (b). The parameters of the other two cylinders were omitted). On top of that there is the parameter *input noise gain* for setting some noise at the velocities.

4.2 Image Processing

Figure 4.3 shows the sequence of the *image processing* steps. Every step is described in the below subsections.

4.2.1 Camera

The camera supplies the input for the follwing *image processing* steps. Actually, it depends on the *robot simulator* step (see section 4.1). The *robot simulator* step includes a *cedar::dev::sensors::visual::GrabbableGrabber*. This grabber grabs an image of the *cedar::dev::robot::gui::MountedCameraViewer* (the window from Figure 4.2 (a)) and sends this image to the output slot from the *robot simulator* step.

4.2.2 Resize image

The *resize image* step gets its input from the camera of the *robot simulator* step (see subsection 4.2.1). In the case of this thesis the received image has a size of 640x480 pixels. For a better performance it is recommended to resize the image to a smaller size. I mostly resized the image to 128x96 pixels.

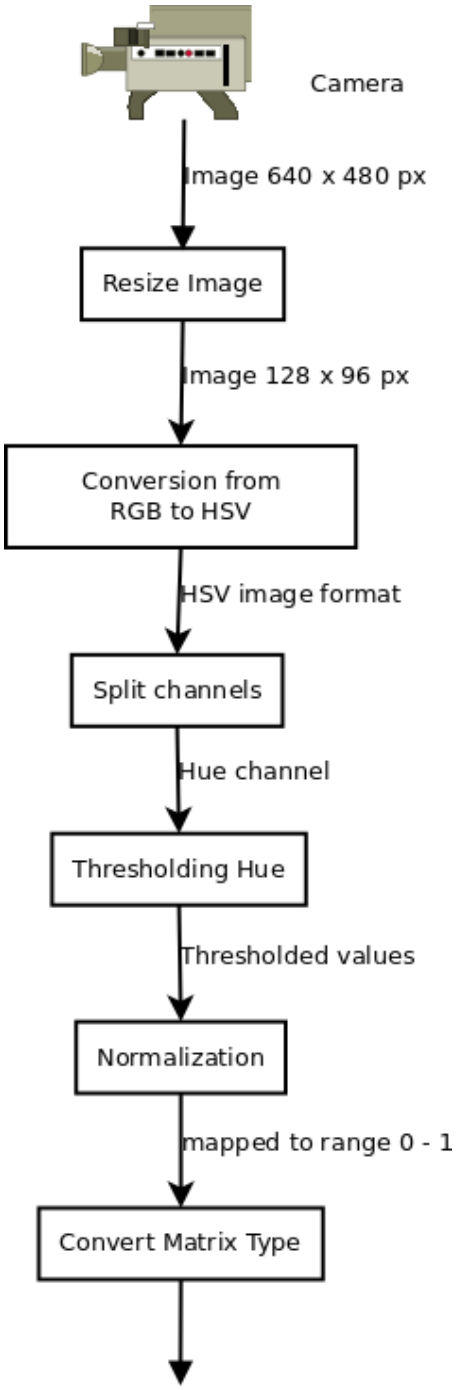

Figure 4.3: Sequence of the image processing

Parameter	Value
output size (1)	96
output size (2)	128
interpolation	Area

Table 4.2: Parameter settings for the *resize image* step

The *resize image* step is a default step from cedar (*cedar::proc::steps::Resize*). It uses the OpenCV function

```
void resize(InputArray src, OutputArray dst, Size dsize,
     double fx=0, double fy=0, int interpolation=INTER_LINEAR)
```

This function offers five different types of interpolation:

- INTER_NEAREST
- INTER_LINEAR
- INTER_AREA
- INTER_CUBIC
- INTER_LANCZOS4

The programmers of OpenCV suggest:

> "To shrink an image, it will generally look best with CV_INTER_AREA interpolation, whereas to enlarge an image, it will generally look best with CV_INTER_CUBIC (slow) or CV_INTER_LINEAR (faster but still looks OK)."[1]

Since in this work a shrinking of the image is needed, the interpolation type *CV_INTER_AREA* is used. Table 4.2 shows the parameter settings of the *resize image* step.

4.2.3 Color Conversion from RGB to HSV

In computer vision the mostly used color format is HSV (*H*ue, *S*aturation, *V*alue), because its three components are independent of each other. For example, it is possible to change the hue without affecting the saturation or the intensity. Unfortunately the standard color format in most computers, cameras, etc. is RGB (*R*ed, *G*reen, *B*lue). In RGB every component depends on each other. Therefore a color format conversion is required.

[1] Source: http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#resize

Parameter	Value
source type	AUTO
target type	HSV

Table 4.3: Parameter settings of the *color conversion* step

Cedar provides a step called *ColorConversion* (*cedar::proc::steps::ColorConversion*) for converting the color format from RGB to HSV and vice versa. But this step just uses the default color conversion method (*cv::cvtColor*) from OpenCV which has a small disadvantage. Instead of using the possible 0 - 255 range for the hue, OpenCV just uses a range from 0 to 179 (as described in [Emami, 2010]). To obtain a somewhat larger color space a new color conversion step (*ColorConversionModified*) was written, which is a slightly modified version of [Emami, 2010] and [Hel-Or, 2004] respectively. The complete code can be reviewed in the appendix 7.2.6. Table 4.3 shows the parameter settings of this step.

4.2.4 Splitting channels

Since only the hue is needed for all later steps (saturation and value can be neglected), the default cedar step *ChannelSplit* (*cedar::proc::steps::ChannelSplit*) splits the incoming HSV image (which is a multi-channel array) into three separated channels by using the OpenCV function

```
void cv::split(const Mat& src, Mat* mvbegin)
```

where

- *src* is the multi-channel HSV image from the *modified color conversion* step of subsection 4.2.3
- *mvbegin* is the output vector of three single-channel arrays

Every channel has its own output slot, but in this work only the hue slot is used.

4.2.5 Thresholding hue

This step allows to threshold a range of hue values. It is a very simple step, which uses the following OpenCV function to filter the hue values between a lower limit and an upper limit:

```
void inRange(InputArray src, InputArray lowerb,
     InputArray upperb, OutputArray dst)
```

Parameter	Value
Hue lower limit	42
Hue upper limit	44

Table 4.4: Parameter settings of the *thresholding hue* step

Parameter	Value
gain factor	0.0039

Table 4.5: Parameter settings of the *normalization* step

where

- *src* is the input array (respectively the hue channel) received from the *ChannelSplit* step of subsection 4.2.4
- *lowerb* is the lower boundary scalar
- *upperb* is the upper boundary scalar
- *dst* is the resulting output array

Equation 4.2 describes how the thresholding is done. If the value of element i of the input array is in the range, then element i of the output array is set to 255. Otherwise it is set to 0.

$$dst(i) = \begin{cases} 255 & lowerb \leq src(i) \leq upperb \\ 0 & otherwise \end{cases} \tag{4.2}$$

Table 4.4 shows the parameter settings of the *thresholding hue* step.

4.2.6 Normalization

The *normalization* step is necessary because the *thresholding* step of subsection 4.2.5 returns a field with values 0 and 255. So the *normalization* step rescales these values to the range [0, 1] by multiplying the field with 0.0039, which is approximately $\frac{1}{255}$ (see table 4.5).

4.2.7 Convertion of matrix type

Up to this step the matrix type is a *CV_8U*, that is an 8-bit unsigned integer. The *perceptual field* step (see section 4.3) needs a *CV_32F* matrix type (that is a 32-bit float matrix) to work. This is done by the following OpenCV function:

Parameter	Value
target type	float

Table 4.6: Parameter settings of the *convertion of matrix type* step

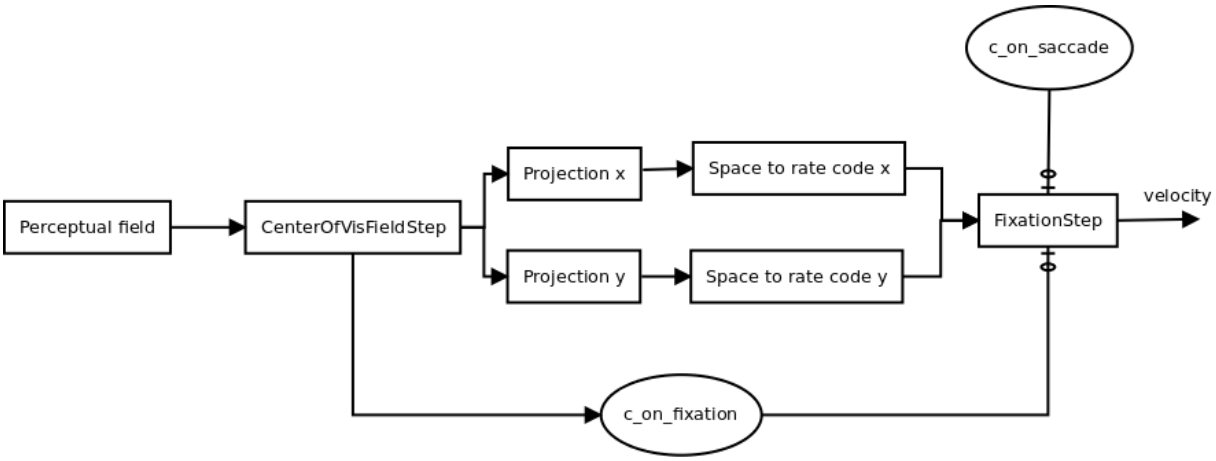

Figure 4.4: Fixation system

```
void Mat::convertTo(OutputArray m, int rtype,
     double alpha=1, double beta=0 ) const
```

Table 4.6 shows the paramter settings of this step.

4.3 Perceptual field

The perceptual field is a neuronal field based on the DFT, which is explained in detail in section 2.1. The input of this field comes from the *conversion of matrix type* step (see subsection 4.2.7). This field has a peak for every detected object.

Table 4.7 shows which parameters were used for this step.

4.4 Fixation System

Figure 4.4 shows the sequence of the fixation system. Every step is described in the following subsections.

Parameter	Value
resting level	-1
time scale	100
global inhibition	-0.005
dimensionality	2
sizes	96x128
input noise gain	0.1
sigmoid	absSigmoid
threshold	0
beta	100
Kernel 0	
anchors	0; 0
amplitude	2
sigmas	3; 3
shifts	0; 0
limit	5
Kernel 1	
anchors	0; 0
amplitude	-0.7
sigmas	3; 3
shifts	0; 0
limit	5
Noise Kernel	
anchors	0; 0
amplitude	1
sigmas	3; 3
shifts	0; 0
limit	5

Table 4.7: Parameter settings for the *perceptual field* step

Parameter	Value
size x	20
size y	20

Table 4.8: Parameter settings of the *center of visual field* step

4.4.1 Center of visual field

The *center of visual field* step gets its input from the perceptual field (see 4.3). This step has two parameters called *size_x* and *size_y* to define the size of the fovea centralis. The fovea centralis is implemented as a region of interest (ROI) which means that a rectangle *cv::Rect* is build with the specified size within the center of the input field. Only the values inside the ROI are of interest. So this ROI is inserted into the center of a zero matrix with the size of the input and is sent to the first output slot called *fovea centralis*.

To activate the *fixation* step (see subsection 4.4.4) the *center of visual field* step checks if a peak is in the fovea centralis and sends the result to the second slot called *c_on_fixation*.

Table 4.8 shows the parameter settings of this step.

4.4.2 Projection

The *projection* step projects neuronal activation between different dimensions. It can expand an input from a lower dimension to a higher dimension or it can compress an input from a higher dimension to a lower dimension. Especially in case of this work a compression of 2D to 1D is used to divide the neuronal activation field into a horizontal (x-direction) and a vertical (y-direction) vector. As you can see in figure 4.4 there is a separate *projection* step for each direction. To achieve a reduction from 2D to 1D an OpenCV function named *cv::reduce* is called which receives four parameters:

```
void reduce(InputArray src, OutputArray dst, int dim, int rtype)
```

- The *src* parameter is the neuronal activation from the *center of visual field* step as a 2D matrix (see subsection 4.4.1).
- The *dst* parameter is the neuronal activation after compression as a 1D vector.
- The *dim* parameter is 0, if the matrix is reduced to a single row or 1, if it is reduced to a single column.
- The *rtype* parameter describes the compression type which can be one of the following:

Parameter	Value
dimension mapping (1)	drop
dimension mapping (2)	0
output dimensionality	1
output dimension	128
compression type	Maximum

Parameter	Value
dimension mapping (1)	0
dimension mapping (2)	drop
output dimensionality	1
output dimension	96
compression type	Maximum

Table 4.9: Parameter settings of the horizontal (left) and vertical (right) *projection* steps

Parameter	Value
lowerLimit	-1
upperLimit	1
tau	100

Table 4.10: Parameter settings for both *space code to rate code* steps in the fixation system

- CV_REDUCE_SUM: It is a sum over all rows or over all columns.
- CV_REDUCE_MAX: It calculates the maximum of all rows or columns.
- CV_REDUCE_MIN: Similarly to CV_REDUCE_MAX, but it calculates the minimum of all rows or columns.
- CV_REDUCE_AVG: The mean vector of all rows or columns is built.

In this thesis only CV_REDUCE_SUM and CV_REDUCE_MAX play a role. Both of them can be used inside the fixation system and also in all other occurrences in the architecture, but it turned out that CV_REDUCE_MAX is slightly more efficient, so it is used throughout this work.

Table 4.9 shows the parameter settings of the horizontal and vertical *projection* step.

4.4.3 Transformation from space code to rate code

After the projection from 2D to 1D the outputs have to be transfered from space code to rate code to get information about where the maximum of the projected field is. This is also done by a cedar internal step called *SpaceToRadeCode* (*cedar::dyn::SpaceToRateCode*). The parameters for both *SpaceToRateCode* steps in the fixation system can be seen in Table 4.10.

4.4.4 Fixation step

The *fixation* step has four inputs. The first two receive the rate code (see subsection 4.4.3) for horizontal and vertical direction. The last two are activation neurons ($c_{onFixation}$ and $c_{onSaccadePause}$).

Parameter	Value
fixation active	<checked>
gain	1

Table 4.11: Parameter settings of the *fixation* step

This step calculates the velocity vector which is then sent to the robot simulator.

Equations 4.3 and 4.4 show how the two velocities are calculated:

$$v_x = c_{onFixation} c_{onSaccadePause} r_x g_x \tag{4.3}$$

$$v_y = c_{onFixation} c_{onSaccadePause} r_y g_y \tag{4.4}$$

where:

- $c_{onFixation}$ is 0, if no peak is detected in the *center of visual field* step, 1 otherwise
- $c_{onSaccadePause}$ is 0, if currently a saccade is executed, 1 otherwise
- r_x and r_y are the rate codes for horizontal and vertical direction
- $g_x$and g_y are gains, which can be set in the parameterlist of the *fixation* step

Thus, the fixation is only active, when no saccade is performed and a peak is on the fovea centralis. Additionally, it is possible to manually turn the fixation on or off by clicking the *fixation active* checkbox. Furthermore, there is a gain parameter to set the strengthness of the fixation (see Table 4.11).

4.5 Saccade System

Figure 4.5 shows a schematic overview of the sequence of the saccade system.

4.5.1 Target field

The target field uses the same dynamical field function as the perceptual field from section 4.3, but the parameters are a little bit different to obtain self-sustained peaks. Table 4.12 shows the parameters of the target field. The values with a double underscore are the values which differ from the parameters of the perceptual field.

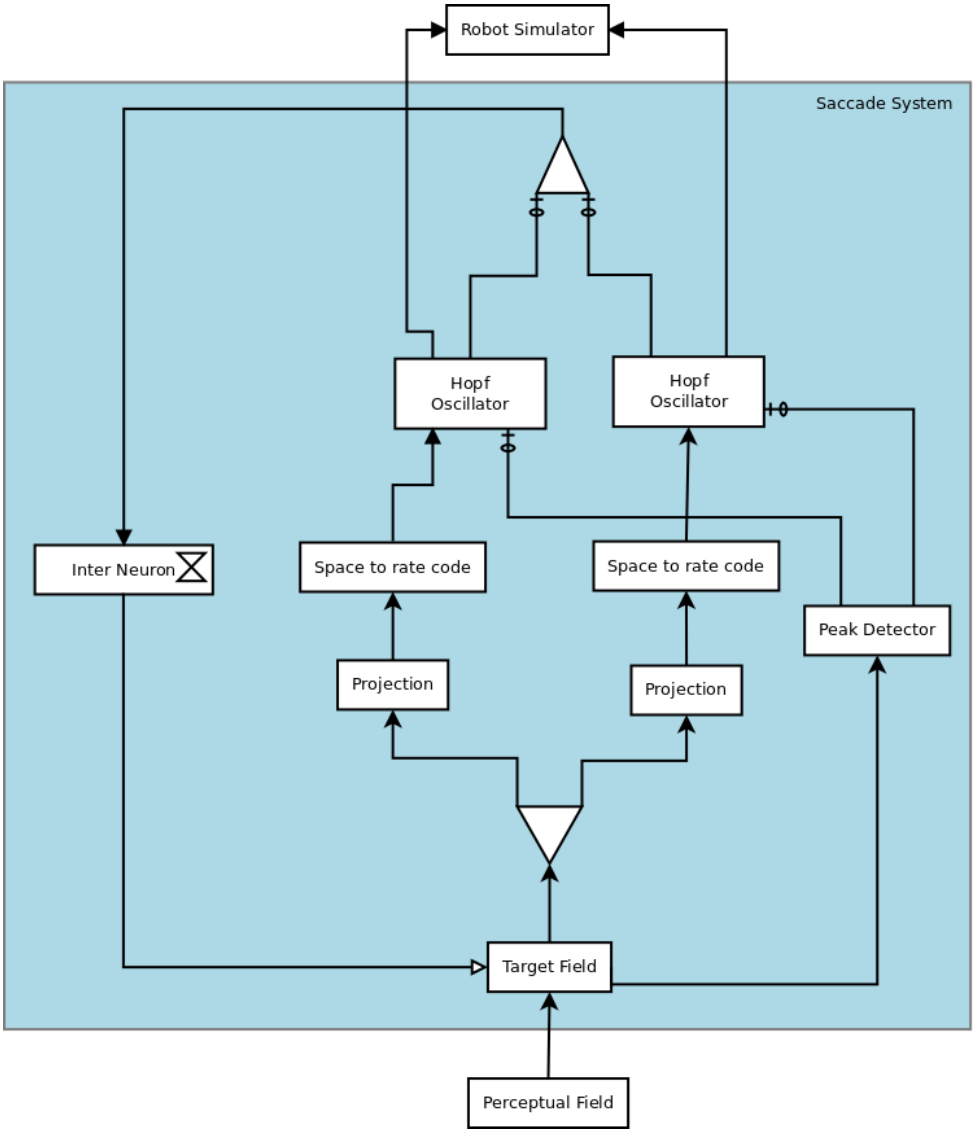

Figure 4.5: Sequence of the saccade system

Parameter	Value
resting level	0
time scale	100
global inhibition	-0.018
dimensionality	2
sizes	96x128
input noise gain	0.1
sigmoid	absSigmoid
threshold	0
beta	100
Kernel 0	
anchors	0; 0
amplitude	20
sigmas	3; 3
shifts	0; 0
limit	5
Kernel 1	
anchors	0; 0
amplitude	-6.5
sigmas	3; 3
shifts	0; 0
limit	5
Noise Kernel	
anchors	0; 0
amplitude	1
sigmas	3; 3
shifts	0; 0
limit	5

Table 4.12: Parameter settings for the *target field* step

Parameter	Value
sigmoid	ExpSigmoid
threshold	10
beta	1000

Table 4.13: Parameter settings of the *peak detector* step connected to the target field

Parameter	Value
lowerLimit	-1
upperLimit	1
tau	10

Table 4.14: Parameter settings of the *space code to rate code* step in the saccade system

Peak detector

A peak detector is connected to the target field. It checks, if the target field build a peak. This is done by building the sum on the input (the target field) and using that sum as the input for a sigmoidal function equal to equation 4.5. The result (1 if a peak is detected, 0 otherwise) of this peak detector is sent to the *Hopf oscillator* steps (see subsection 4.5.4) and to the *too far or too close* steps (see subsection 4.6.6).

Table 4.13 shows the parameter settings of this *peak detector* step.

4.5.2 Projection

This step is the same as the *projection* step in subsection 4.4.2. Even the parameter settings are the same as in Table 4.9.

4.5.3 Transformation from space code to rate code

This is the same step as in subsection 4.4.3 but with the parameter settings of table 4.14.

4.5.4 Hopf oscillator

The dynamical system of a Hopf oscillator is described above in section 2.4. Here, the special properties within the architecture are discussed. Figure 4.6 shows the sequence of a *Hopf oscillator* step. It can be seen that it is not just one Hopf oscillator, but there are two Hopf oscillators in one step and also two sigmoids and two activation neurons.

The Hopf oscillator, which is responsible for the saccades, has two inputs: The rate code from subsection 4.5.3 and the learned gain from subsection 4.6.7. The multiplication of the rate

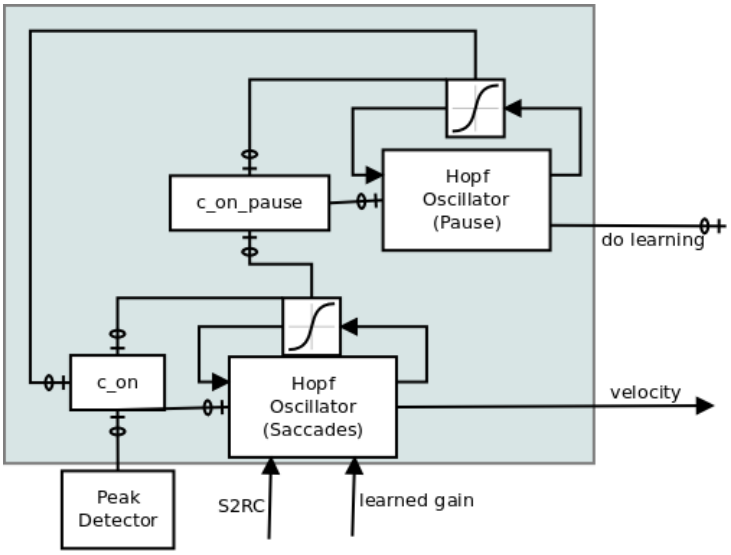

Figure 4.6: Sequence within the Hopf oscillator step

code, the learned gain and a gain parameter (can be set by the user) is used for the amplitude μ in equation 2.6. $\dot{x}$ is the output representing the velocity for the robot simulator. $\dot{y}$ is only used as an auxiliary variable. To start on the limit cylce x is set to $x = 0.01$ and y is set to $\sqrt{\mu^2 - 0.0001}$ at the beginning of every new saccade. A saccade starts, when the peak detector detected a peak on the target field and then the activation neuron c_{on} is set to 1.

With every time step there is a sigmoidal function check to find out whether the oscillator completed a half cycle. This is done by the implementation of equation 4.5.

$$S(x) = \frac{1}{1 + \exp(-\beta f(x) - \vartheta)} \tag{4.5}$$

where

- β is the slope
- ϑ is the threshold
- $f(x) = 2\exp\left(-10000x^2\right)$

Note: $S(x)$ can be also another sigmoidal function selectable via the parameters in the processing framework, but $f(x)$ stays the same.

If the sigmoidal function reaches its threshold, the oscillator has completed a half cycle and the saccade is finished. Now the activation neuron c_{pause} is activated. During the pause $c_{learning}$ is active (used in the learning system in section 4.6) and the resting level is decreased via the interneuron (see subsection 4.5.5), which automatically leads to a non-detection by the peak

Parameter	Value
tau saccade	300
tau relax	1000
angular speed	628.3185
gamma	50
gain	0.5
sigmoid saccade	
sigmoid	ExpSigmoid
threshold	1
beta	1000
sigmoid relax	
sigmoid	ExpSigmoid
threshold	1
beta	1000

Table 4.15: Parameter settings of the *Hopf oscillator* steps for both horizontal and vertical version

detector resulting in $c_{on} = 0$. The Hopf oscillator of the pause phase has a fixed amplitude of $\mu = 0.85$. The duration of the pause can be controlled by the parameter τ_{relax}. The pause phase is also stopped after a half cycle by the same sigmoidal function of equation 4.5, only $f(x)$ is changed to $f(x) = 2\exp\left(-100x^2\right)$. Now $c_{learning}$ is set to zero and the interneuron increases the resting level of the target field again. As soon as the peak detector detects a peak on the target field c_{pause} changes to 0 and c_{on} changes to 1. The whole process is now ready for a new saccade.

Table 4.15 shows the mainly used settings of the parameter of the *Hopf oscillator* step.

4.5.5 Interneuron

The interneuron could be also called a synchronizer neuron. It gets input from both *Hopf oscillator* steps. If in the Hopf oscillator the pause phase is active the interneuron receives a 0 value on its slot and 1 otherwise. Only if both slots are zero the interneuron can change the resting level of a field by adding h to that field. h is calculated as in equation 4.6.

$$h = r_{switch}(1 - r_x r_y) \tag{4.6}$$

where

- r_x is an input from the horizontal Hopf oscillator. It is 0 if the pause is active and 1 otherwise.

Parameter	Value
resting level switch	-100

Table 4.16: Parameter settings of the *interneuron* step of the saccade system

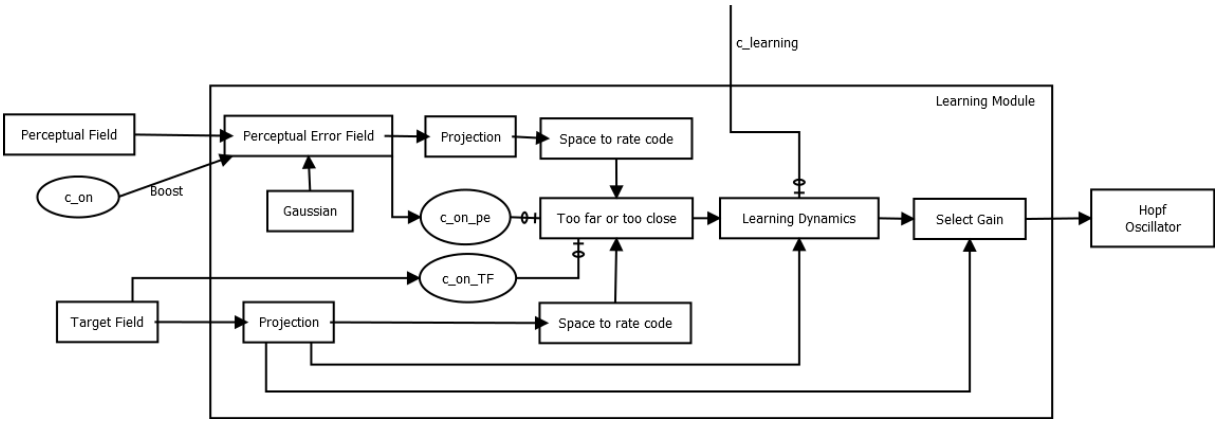

Figure 4.7: Learning System

- r_y is an input from the vertical Hopf oscillator. It is 0 if the pause is active and 1 otherwise.
- r_{switch} indicates how much the resting level of a field has to be decreased.

This interneuron is there to decrease the resting level of the target field only when both oscillators are in there pause phase. Then the target field has the opportunity to lost its peak and to build a new peak, when the pause phase is over.

Furthermore, there is a second output slot, which sends the result of equation 4.7 to the *fixation* step (see subsection 4.4.4), that means if $c_{fixation} = 1$, no saccade is performed and the *fixation* step is allowed to fixate.

$$c_{fixation} = 1 - r_x r_y \tag{4.7}$$

Table 4.16 shows the paramter settings of the *interneuron* step of the saccade system.

4.6 Learning System

Figure 4.7 shows the sequence of the learning system. Note, that the figure only shows the sequence for one dimension, but in the implemented system this learning module occurs twice (one for each dimension). All steps are described in the subsections below.

Parameter	Value
amplitude	-1
dimensionality	2
sigma (1)	20
sigma (2)	20
centers (1)	48
centers (2)	64
sizes (1)	96
sizes (2)	128

Table 4.17: Parameter settings for the *gaussian input* step

Parameter	Value
resting level switch	100

Table 4.18: Parameter settings of the *interneuron* step of the learning system

4.6.1 Gaussian input

The *gaussian input* step is a default cedar step (*cedar::processing::sources::GaussInput*). As the name suggests it builds a gaussian. This gaussian is one of three inputs to the perceputal error field described in subsection 4.6.3. It serves the attenuation of learning in the center of the perceptual error field.

Table 4.17 shows the parameter settings that were chosen in this thesis.

4.6.2 Interneuron

The interneuron is described in subsection 4.5.5. Here in the learning system, the difference is, that the second output slot with the value of $c_{fixation}$ is not used and more import the parameter resting level switch is set to a positive value, what leads to a boost of a field and not a decrease. The perceptual error field is boosted as soon as both Hopf oscillators are in pause state. Then, the perceptual error field can create peaks, which are necessary for the learning in the subsections below.

Table 4.18 shows the parameter settings of the *interneuron* step of the learning system.

4.6.3 Perceptual error field

The perceptual error field is also a neuronal dynamical field as in section 4.3 and subsection 4.5.1, but the parameters are somewhat different in relation to the perceptual field. It can be seen in table 4.19, where the different parameters are marked with a double underscore.

Parameter	Value
resting level	-100
time scale	100
global inhibition	-0.018
dimensionality	2
sizes	96x128
input noise gain	0.1
sigmoid	absSigmoid
threshold	0
beta	100
Kernel 0	
anchors	0; 0
amplitude	2
sigmas	3; 3
shifts	0; 0
limit	5
Kernel 1	
anchors	0; 0
amplitude	0
sigmas	3; 3
shifts	0; 0
limit	5
Noise Kernel	
anchors	0; 0
amplitude	1
sigmas	3; 3
shifts	0; 0
limit	5

Table 4.19: Parameter settings for the *perceptual error field* step

Parameter	Value
sigmoid	ExpSigmoid
threshold	1
beta	1000

Table 4.20: Parameter settings of the *peak detector* step connected to the perceptual error field

Peak detector

Also the perceptual error field is connected to a peak detector. This *peak detector* step is equal to the *peak detector* step in subsection 4.5.1. Only the value of the threshold parameter is different (see Table 4.20).

4.6.4 Projection

This step is the same as the projection step in subsection 4.4.2. Even the parameter settings are the same as in Table 4.9.

4.6.5 Transformation from space code to rate code

This is the same step as in subsection 4.4.3, even with the same parameter settings of table 4.10.

4.6.6 Too far or too close

The *too far or too close* step has four input slots: The rate code of the target field, the rate code of the perceptual error field, c_{target} (1 if there is a peak on the target field, 0 otherwise) and c_{pe} (1 if there is a peak on the perceptual error field, 0 otherwise). If c_{target} is 1 and c_{pe} is 0, that means, that a saccade is actually executed and until now no perceptual error can be received, so the rate code of the target field is cached. As soon as c_{target} and c_{pe} switch their state, a perceptual error is detected on the perceptual error field. Straightway the cached rate code of the target field is multiplied by the rate code of the perceptual error field (see also section 2.5). For this multiplication this OpenCV function is used:

```
void multiply(const Mat& src1, const Mat& src2,
              Mat& dst, double scale=1)
```

The result (named as perceptual error rating) is sent to the *learning dynamics* step of subsection 4.6.7.

Parameter	Value
tau	10000
sigmoid	ExpSigmoid
threshold	1
beta	1000
size	128 (horizontal step) or 96 (vertical step)

Table 4.21: Parameter settings for the *learning dynamics* step

4.6.7 Learning Dynamics

The *learning dynamics* step receives three inputs: $c_{learning}$ (from subsection 4.5.4), the projected target field of dimension d (from subsection 4.5.2) and the perceptual error rating (from subsection 4.6.6). The mathematical equation of this step can be found in section 2.5.

In every cycle it is checked, if a peak is on the target field by using a sigmoidal function over the sum over the target field as shown in this code example:

```
cv::Scalar summe = sum(input);
cv::Mat& sigmoid_u = this->mSigmoidalActivation->getData();
this->mSigmoidalActivation->getData().at<float>(0,0) = summe[0];
sigmoid_u = _mSigmoid->getValue()->compute<float>
              (mSigmoidalActivation->getData());
```

The sigmoid itself is equivalent to equation 4.5. If *sigmoid_u* is 1, that means that there is a peak on the target field, the target field is cached. Only when the target field was cached, accordingly a target really existed, and $c_{learning}$ is set to 1 by the Hopf oscillator, the learning system starts to learn. For that it multiplies the perceptual error rating with the projection of the target field. A short excerpt of the implementation:

```
double c_learning = this->mDoLearning->getData().at<float>(0,0);
cv::multiply(this->mErrorSign->getData(),
             this->mTargetTemp->getData(), dot_u);
output += (c_learning * cedar::unit::Milliseconds(time)
           / cedar::unit::Milliseconds(tau) * dot_u);
cv::threshold(output, output, 0.0, 0.0, cv::THRESH_TOZERO);
```

The OpenCV threshold function ensures that the learned gain does not exceed the value 0.

Table 4.21 shows the parameter settings which were mostly used for this step.

Parameter	Value
sigmoid	ExpSigmoid
threshold	1
beta	1000

Table 4.22: Parameter settings of the *select gain* step

4.6.8 Select gain

The *select gain* step receives the projected target field and the learning dynamics field as input. It checks via a sigmoidal function (like the function in subsection 4.6.7), if a peak is on the target field. If that is true, than the location of that peak is fetched with the OpenCV function *cv::minMaxLoc()*, which returns a x,y-location. Now the corresponding learned gain is picked from the learning dynamics field and sent via the output slot to the Hopf oscillator from subsection 4.5.4. This code snippet illustrates it:

```
cv::Point maxLoc;
minMaxLoc(this->mTarget->getData(), NULL, NULL, NULL, &maxLoc);
this->mOutput->getData().at<float>(0,0) =
   this->mLearnedGain->getData().at<float>(maxLoc.x, maxLoc.y);
```

Table 4.22 shows the parameter settings of the *select gain* step.

4.7 The system overview

For conclusion figure 4.8 shows a general overview over the whole system, which was explained step-by-step in the above sections. Only the image processing is compressed to one module.

4.8 Auxiliary classes

4.8.1 Buffer to file

This class was implemented for buffering data output and writing it to harddisk. It is possible to connect the *buffer to file* step to any output slot of any other step. If the step, where the *buffer to file* step is connected to, is started, a new *std::ofstream* is created. Every time the buffer is full, the data is written into the file (the path and name of the file can be set by parameter) and the buffer is emptied. Also if the step, where the *buffer to file* step is connected to, is stopped, the buffer is flushed (that means, all remaining data is written to file) and closed. The buffer size depends on the hardware it is running on.

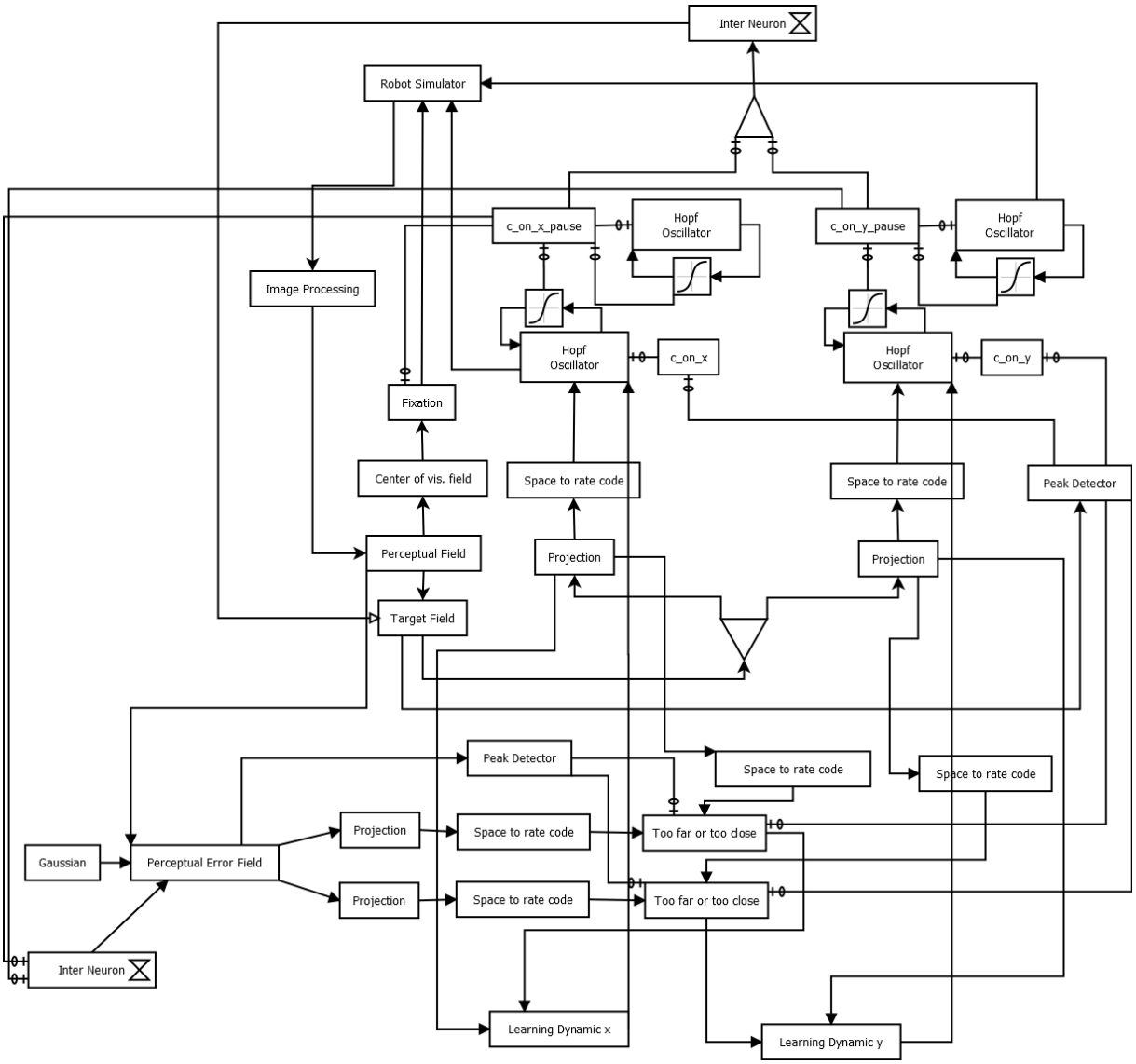

Figure 4.8: The complete system in detail

4.8.2 Distance error

The *distance error* step is a step, which was used to calculate the perceptual error and send it to the *buffer to file* step for later plotting (see subsection 5.4.1). The *distance errror* step has three output slots. The first two slots each receive the distance error for the horizontal direction ε_x (equation 4.8) or the distance error for vertical direction ε_y (equation 4.9). The third slot receives the distance error for an oblique direction $\varepsilon_{x,y}$ (equation 4.10).

$$\varepsilon_x = x - \frac{sizeX}{2} \tag{4.8}$$

$$\varepsilon_y = y - \frac{sizeY}{2} \tag{4.9}$$

$$\varepsilon_{x,y} = \sqrt{\varepsilon_x^2 - \varepsilon_y^2} \tag{4.10}$$

4.8.3 Plugin

The plugin class just lists all implemented steps for including them as a plugin into cedars processing framework.

5 Results

This section shows a few interesting results, which will show that the architecture works as expected.

5.1 Saccade trajectories

Figures 5.1 and 5.2 show saccade trajectories in horizontal, vertical and oblique direction with a different noise gain of 0.1 or 0.5. The higher the noise is, the higher are the fluctuations of the trajectories. However, the saccades are performed nearly good.

Because the saccade system is based on a cartesian coordinate system and an oblique saccade consists of a horizontal and a vertical part, which are separated from each other (they only start at the same time), the trajectory of an oblique saccade should be curved [Grossman and Robinson, 1988]. Figure 5.3 shows that this is also found in the saccade system of this thesis. The blue line is an oblique saccade. It can be seen that the trajectory is curved. The black line should only represent how it would look like in a polar coordinate system and is just supplemented manually.

5.2 Gains accross visual field

In this section a yellow cylinder was placed to different pre-saccadic locations. Then a saccade was initiated and the velocity was plotted.

5.2.1 Gains from different horizontal positions

Figure 5.4 shows a comparison of ten horizontal saccades from different horizontal locations. The legend indicates the different starting positions. It can be clearly seen that the saccades differ in their amplitude and also something in the duration of the saccades.

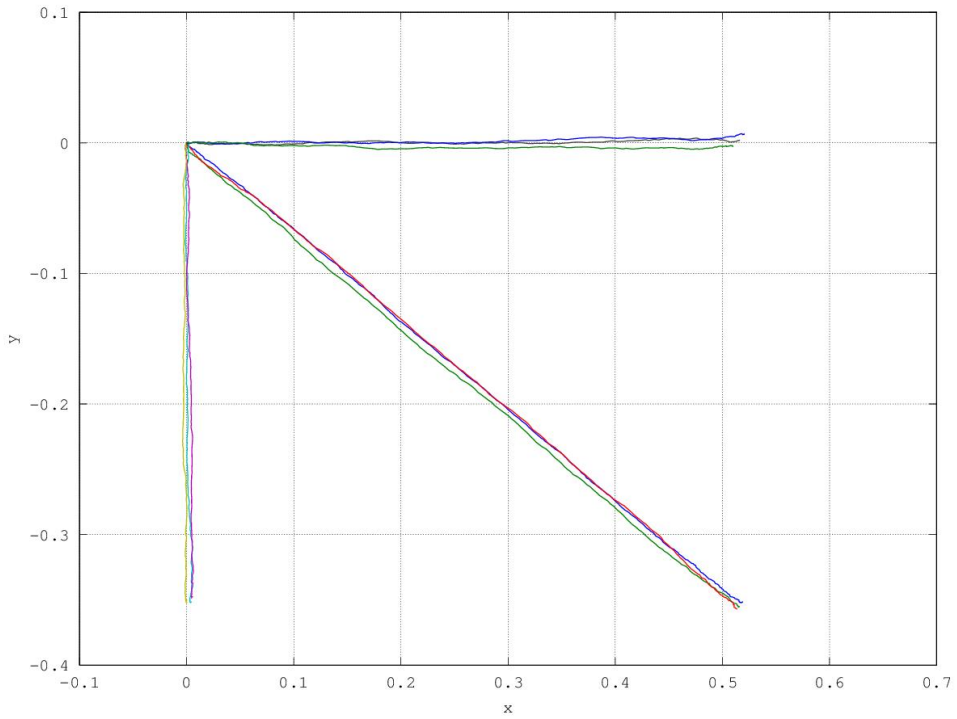

Figure 5.1: Saccade trajectories with a noise gain of 0.1

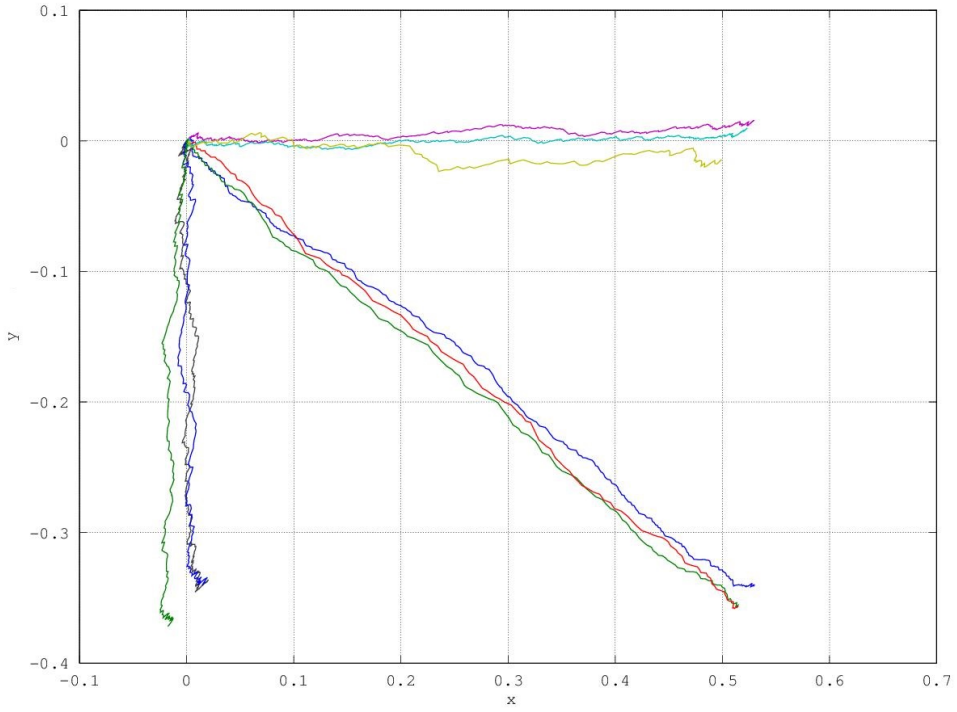

Figure 5.2: Saccade trajectories with a noise gain of 0.5

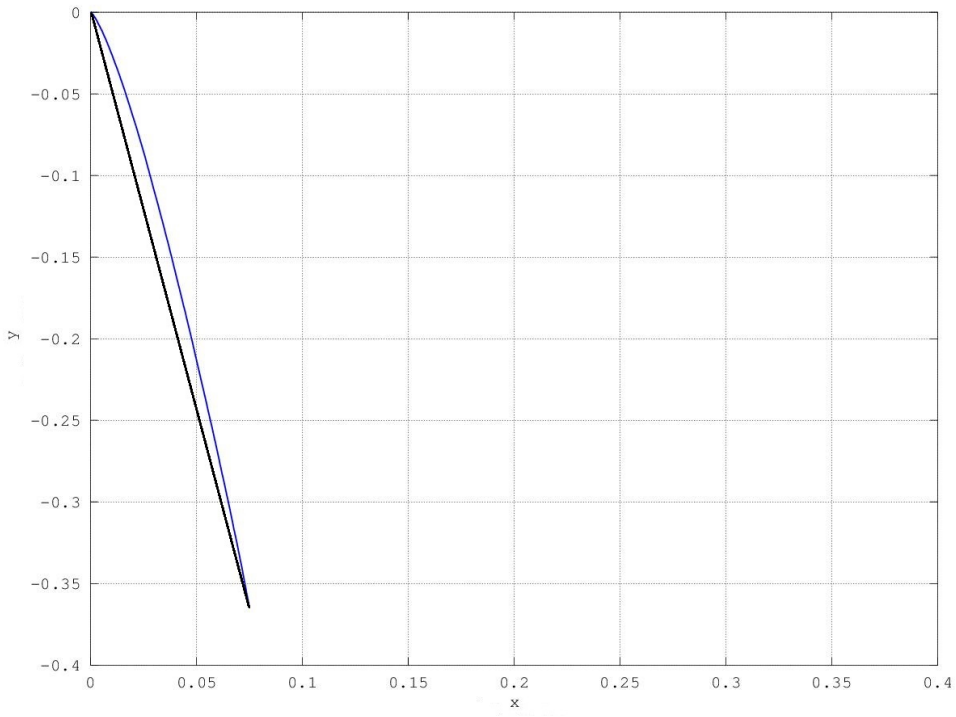

Figure 5.3: An oblique saccade with a curved trajectory

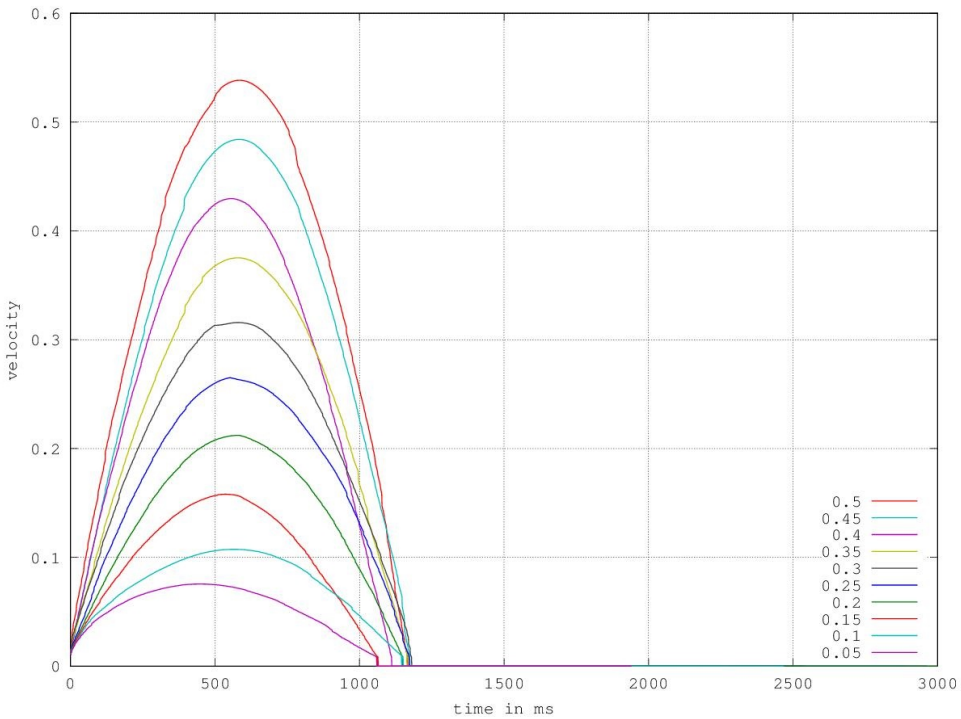

Figure 5.4: Horizontal saccades from different horizontal locations

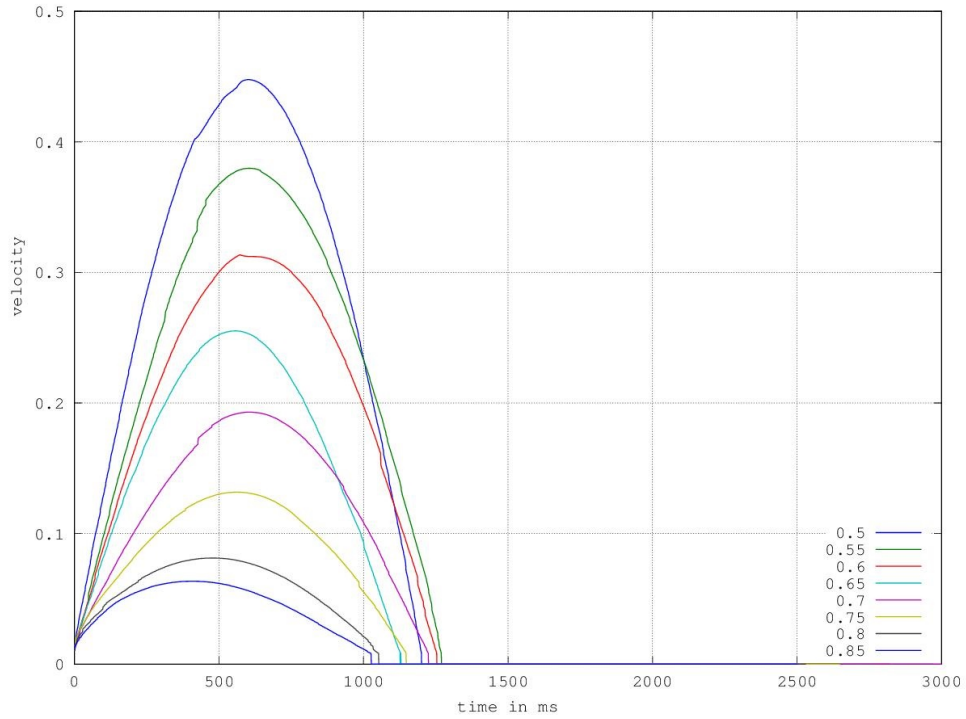

Figure 5.5: Horizontal saccades from different vertical locations

5.2.2 Gains from different vertical positions

Figure 5.5 shows a comparison of eight horizontal saccades from different vertical locations. The legend indicates the different starting positions. It can be clearly seen that the saccades differ in their amplitude and also something in the duration of the saccades.

5.3 Alternation between saccades and pause phases

Figure 5.6 shows the alternation between saccades and pause phases with different values for τ_{pause}. The green line is the pause cycle with a fixed amplitude of 0.85. The blue line is the saccade cycle. It can be seen, that the saccade and the pause alternate in regular rythm. The duration of the pause is proportionally dependent on τ_{pause}. The greater τ_{pause} is, the greater is the duration of the pause. The saccade amplitudes and duration stay the same.

5.4 Adaptation experiments

This subsection shows that the visual error of saccades is correctly adapted and that it also generalizes for neighboring locations.

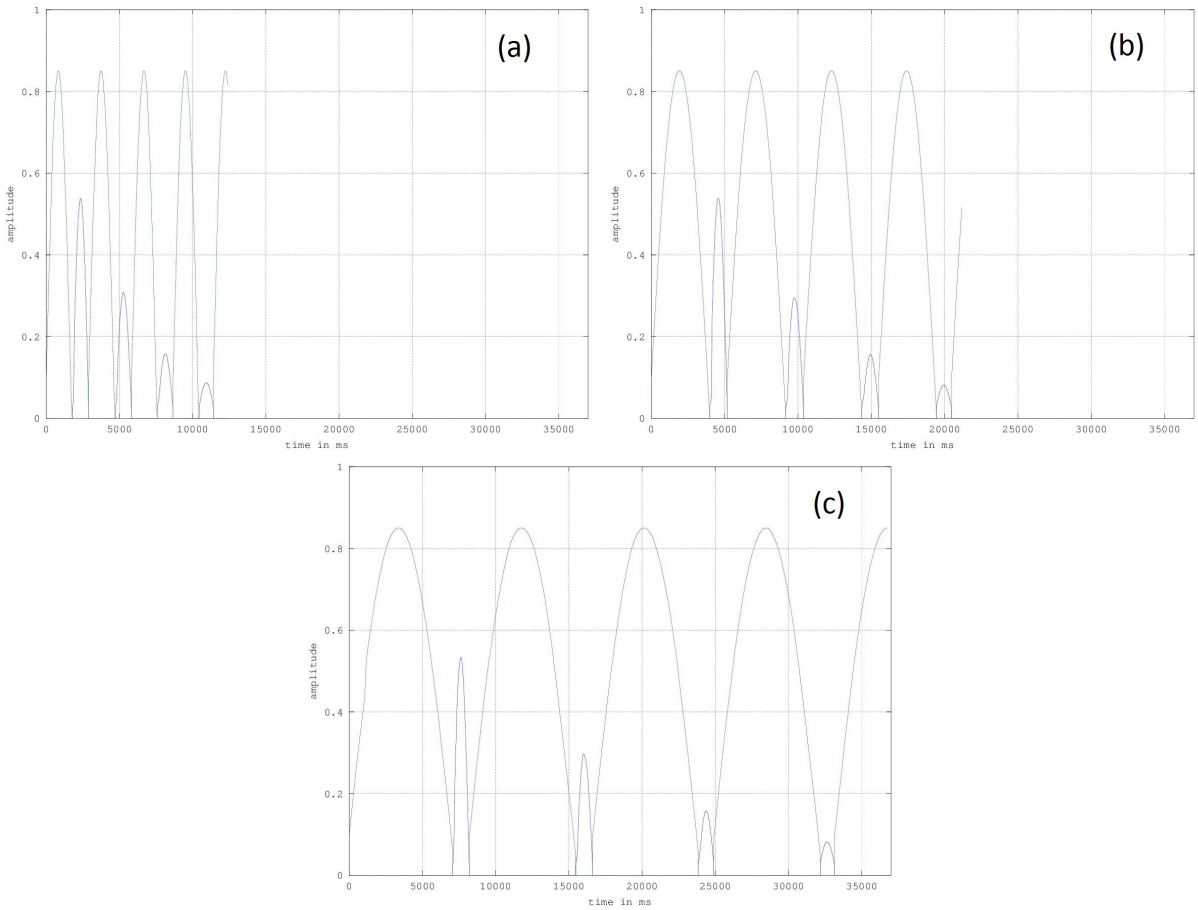

Figure 5.6: Alternation between saccades and pause phases (a): $\tau_{pause} = 500$, (b): $\tau_{pause} = 1000$ and (c) $\tau_{pause} = 2000$

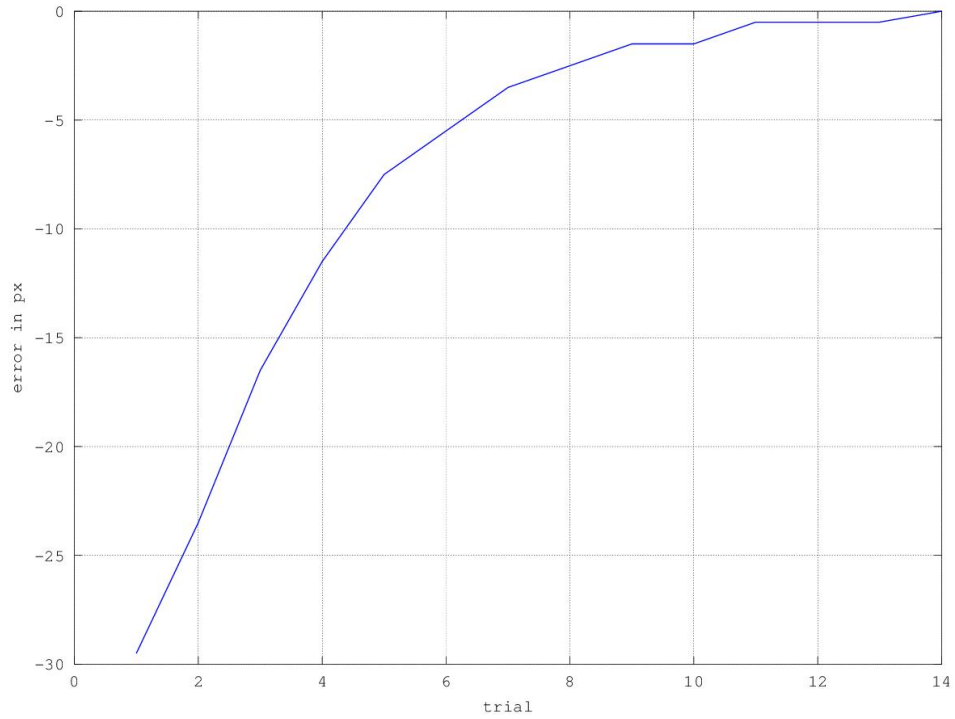

Figure 5.7: Error in pixels of horizontal saccade (gain 0.7, tau-learning 10000)

5.4.1 Error

Figure 5.7 shows a learning curve of the reduced error (trial-by-trial) of a horizontal saccade. The starting amplitude gain was set to 0.7. This was intentionally a false gain. The learning rate was set to $\tau_{learning} = 10000$. After the first trial the landing position of the horizontal saccade was misplaced approximately -29.5 px. But after 14 trials the error was approximately zero.

Figure 5.8 shows the same for a horizontal saccade. Here, the starting amplitude gain was set to 0.5 and the learning rate to $\tau_{learning} = 10000$. At the beginning of the learning the error was approximately -19 px. After 14 trials it was nearly zero.

Figure 5.9 shows an oblique saccade. Here, the starting amplitude gain was set to 0.5 for vertical direction and 0.7 for horizontal direction and the learning rate for both directions to $\tau_{learning} = 10000$. At the beginning of the learning the error was approximately 36 px, but after just 14 trials it was nearly zero.

Note, that the error is negative in figure 5.7 and 5.8, but in figure 5.9 it is positive. This is due to the calculation of the error. While it is calculated for one dimension (horizontal or vertical) equation 4.8 (horizontal) and equation 4.9 (vertical) is used, but for both dimensions (oblique) equation 4.10 is used.

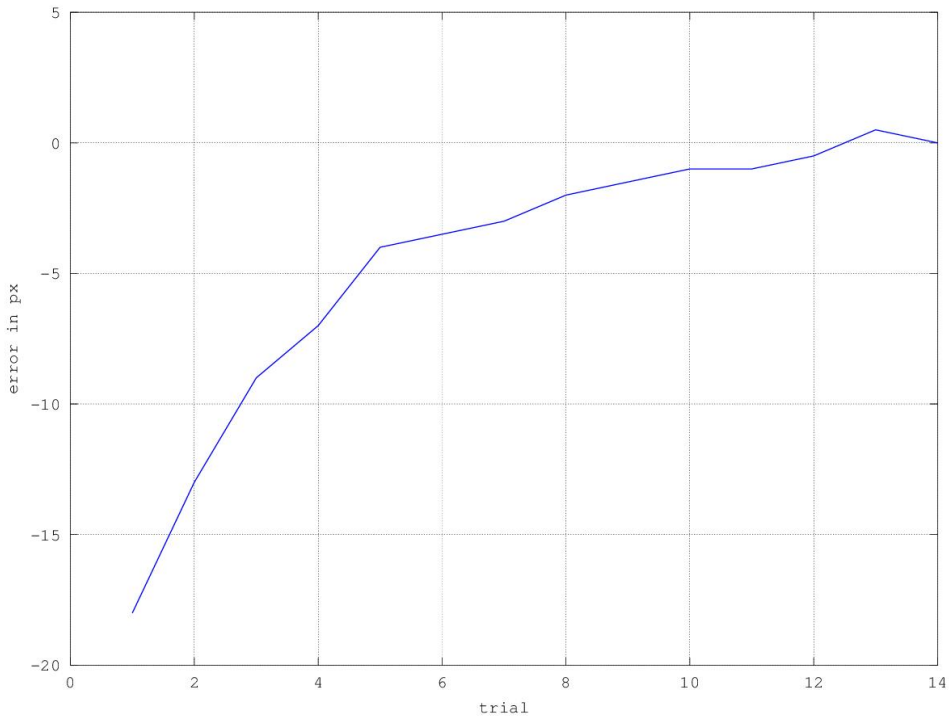

Figure 5.8: Error in pixels of horizontal saccade (gain 0.5, tau-learning 10000)

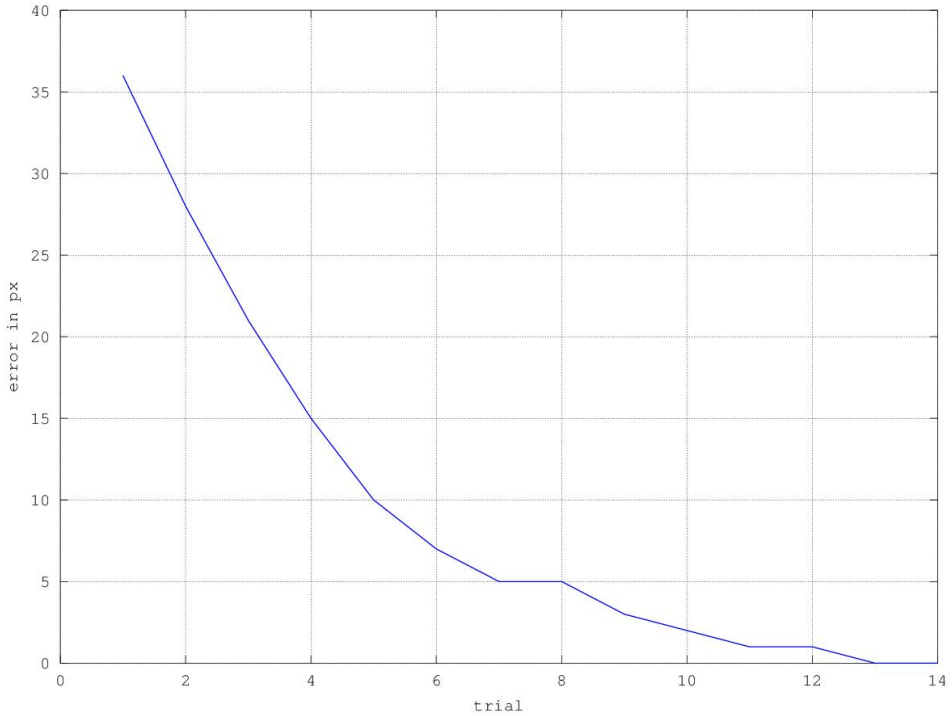

Figure 5.9: Error in pixels of horizontal saccade (gain 0.5, tau-learning 10000)

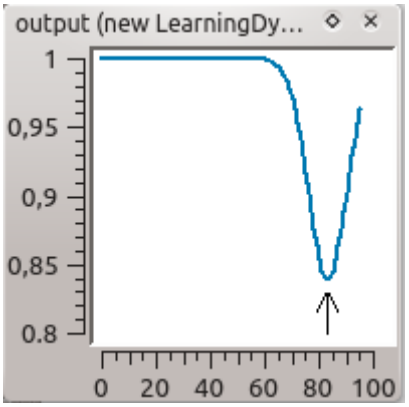

Figure 5.10: Example of a learned gain after one vertical saccade

5.4.2 Generalization to neighboring locations

When a gain is learned by the learning system, it not only learns the gain for the specific pre-saccadic location, but also for its neighbors. Figure 5.10 shows an example of the learning dynamics (the vertical part) after one vertical saccade. The arrow marks the pre-saccadic position. It can be seen very well that there are also learned gains for the neighbors of that location due to the implemented generalization to neighboring locations. There could be a parallelism to the *adaptation field* from [Frens and van Opstal, 1994].

6 Conclusion and further research

The present thesis proposed an architecture for looking behavior in a simulated environment with special focus on saccades. The architecture consisted of six primary components: The saccade system, the fixation system, the learning system, the robot simulator, the image processing and a perceptual field.

Much emphasis was put on the timing of all steps. For example, there were the Hopf oscillators, which encoded the timing and velocity for the saccades, so that the vertical and the horizontal parts started and finished at the same time. The same step also integrated a pause phase, which modeled the duration for the fixation system. So there was an alternation between the saccade system and the fixation system. Also the learning took part during the fixation phase. After every saccade the learning system learned a new gain for the just completed saccade. But not only for that single pre-saccadic location it was learned, but also for their neighboring locations due to generalization.

The results showed that the system worked as expected. Especially the gain of the saccades were learned and the error decreased after every trial. Depending on the parameter settings a correct saccade was made after 14 trials. Furthermore, saccades with different distance had a different amplitude, but the duration of these saccades was substantially less different.

By the way, the system was also observed in the process of possibly performing micro-saccades. Here it would still be appropriate to vary this in further works.

How can it go on? The architecture now only runs in a simulated environment. It would be interesting to connect it to real hardware. The pause phase of the Hopf oscillator should be replaced by a proper inhibition interplay between the saccade system and the fixation system, for example by implementing an initiation field [Wilimzig et al., 2006]. Also, a stereo view could be implemented. The eye movements not yet taken into account like the smooth pursuit eye movement could be programmed. Furthermore, other learning methods could be tested, for example like the saccadic adaptation by reinforcement learning from [Madelain et al., 2011]. Perhaps, it would be also possible to implement a faster adaptation for overshooting saccades than for undershooting saccades [Girard and Berthoz, 2004]. Some experiments could be made following the double-step paradigm first introduced by [McLaughlin, 1967].

Bibliography

[Amari, 1977] Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27:77 – 87.

[Emami, 2010] Emami, S. (2010). Hsv color conversion. http://www.shervinemami.info/colorConversion.html.

[Enderle, 1994] Enderle, J. D. (1994). A physiological neural network for saccadic eye movement control. Technical report, North Dakota State University, Department of Electrical Engineering.

[Ethier et al., 2008] Ethier, V., Zee, D. S., and Shadmehr, R. (2008). Changes in control of saccades during gain adaptation. *The Journal of Neuroscience*, 28(51):13929–13937.

[Findlay and Walker, 2012] Findlay, J. and Walker, R. (2012). Human saccadic eye movements. *Scholarpedia*, 7(7):5095.

[Frens and van Opstal, 1994] Frens, M. A. and van Opstal, A. J. (1994). Transfer of short-term adaptation in human saccadic eye movements. *Experimental Brain Research*, 100(2):293–306.

[Gancarz and Grossberg, 1998] Gancarz, G. and Grossberg, S. (1998). A neural model of saccadic eye movement control explains task-specific adaptation. Technical report, Boston University Center for Adaptive Systems and Department of Cognitive and Neural Systems.

[Girard and Berthoz, 2004] Girard, B. and Berthoz, A. (2004). From brainstem to cortex: Computational models of saccade generation circuitry. Technical report, Laboratoire de Physiologie de la Perception et de l'Action, France.

[Grossberg, 1969] Grossberg, S. (1969). On learning of spatiotemporal patterns by networks with ordered sensory and motor components. 1. excitatory components of the cerebellum. *Studies in Applied Mathematics*, 48:105–132.

[Grossberg and Kuperstein, 1986] Grossberg, S. and Kuperstein, M. (1986). Neural dynamics of adaptive sensory-motor control. *Pergamon Press*.

[Grossman and Robinson, 1988] Grossman, G. E. and Robinson, D. A. (1988). Ambivalence in modelling oblique saccades. *Biological Cybernetics*, 58:13–18.

[Guthrie et al., 1983] Guthrie, B. L., Porter, J. D., and Sparks, D. L. (1983). Corollary discharge provides accurate eye position information to the oculomotor system. *Science*, 221:1193–1195.

[Hel-Or, 2004] Hel-Or, H. (2004). Color conversion algorithms. http://cs.haifa.ac.il/hagit/courses/ist/Lectures/Demos/ColorApplet2/t_convert.html.

[Institut für Neuroinformatik, 2011] Institut für Neuroinformatik (2011). cedar - a framework for cognition, embodiment, dynamics, and autonomy in robotics. http://cedar.ini.rub.de.

[Kuznetsov, 2006] Kuznetsov, Y. A. (2006). Andronov-hopf bifurcation. *Scholarpedia*, 1(10):1858.

[Large and Kolen, 1994] Large, E. W. and Kolen, J. F. (1994). Resonance and the perception of musical meter. *Connection Science*, 6(1):177–208.

[Lomp et al., 2012] Lomp, O., Zibner, S. K. U., Richter, M., Rano, I., and Schöner, G. (2012). A software framework for cognition, embodiment, dynamics, and autonomy in robotics: cedar.

[Madelain et al., 2011] Madelain, L., Paeye, C., and Wallman, J. (2011). Modification of saccadic gain by reinforcement. *Journal of Neurophysiology*, 106(1):219–232.

[McLaughlin, 1967] McLaughlin, S. C. (1967). Parametric adjustment in saccadic eye movements. *Perception & Psychophysics*, 2(8):359–362.

[Pelisson et al., 2010] Pelisson, D., Alahyane, N., Panouilleres, M., and Tilikete, C. (2010). Sensorimotor adaptation of saccadic eye movements. *Neuroscience and Biobehavioral Reviews*, 34:1103–1120.

[Richter, 2011] Richter, M. M. (2011). A neural dynamic architecture for the behavioral organization of an autonomous robotic agent. Master's thesis, Ruhr-Universität Bochum.

[Schöner, 2008] Schöner, G. (2008). Dynamical systems approaches to cognition. *Cambridge Handbook of Computational Cognitive Modelling*, pages 101 – 126.

[Wilimzig et al., 2006] Wilimzig, C., Schneider, S., and Schöner, G. (2006). The time course of saccadic decision making: Dynamic field theory. *Neural Networks*, 19:1059–1074.

7 Appendix

7.1 DVD content

- /Documentation/
  - A coloured version of this thesis as PDF-File
  - A HTML-Version of this thesis
  - Lyx-Files
  - Pictures
  - Dia-Files
  - BibTEX-File
- /Architecture/
  - Source code
  - Scripts
- /Data/
  - measured data

7.2 Source code

This section lists the source code files in alphabetical order.

7.2.1 BufferToFile.h

```
#ifndef UTILITIES_BUFFER_THIEF_H
#define UTILITIES_BUFFER_THIEF_H
#include <iostream>
#include <fstream>
```

```
// CEDAR INCLUDES
#include <cedar/processing/Step.h>
class BufferToFile : public cedar::proc::Step
{
  Q_OBJECT
  //--------------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------------
public:
  //!@brief The standard constructor.
  BufferToFile();
  virtual ~BufferToFile();
  //--------------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------------
public:
  void onStart();
  void onStop();
  //--------------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------------
protected:
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr,
    cedar::aux::ConstDataPtr
  ) const;
  void inputConnectionChanged(const std::string& inputName);
  //--------------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------------
private:
  void compute(const cedar::proc::Arguments&);
  inline const std::string& getPath() const
  {
    return this->_mPath->getValue();
  }
private slots:
  void bufferPathChanged();
  //--------------------------------------------------------------------
  // members
  //--------------------------------------------------------------------
protected:
```

```
  // none yet
private:
  // none yet
  //--------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------
protected:
  // none yet
private:
  cedar::aux::StringParameterPtr _mPath;
  std::ofstream ofs;
  cedar::aux::ConstMatDataPtr mInput;
}; // class utilities::BufferToFile
#endif // UTILITIES_BUFFER_THIEF_H
```

7.2.2 BufferToThief.cpp

```
// LOCAL INCLUDES
#include "BufferToFile.h"
#include <cedar/auxiliaries/MatData.h>
#include "cedar/auxiliaries/sleepFunctions.h"
//--------------------------------------------------------------
// constructors and destructor
//--------------------------------------------------------------
BufferToFile::BufferToFile()
:
_mPath(new cedar::aux::StringParameter
  (
    this, "buffer path", "/home/christian/tmp/data1.csv"
  ))
{
  QObject::connect
  (
    this->_mPath.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(bufferPathChanged())
  );
  this->declareInput("input", false);
}
BufferToFile::~BufferToFile()
{
ofs.flush();
```

```
    ofs.close();
    cedar::aux::sleep(cedar::unit::Seconds(1));
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
cedar::proc::DataSlot::VALIDITY BufferToFile::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr CEDAR_DEBUG_ONLY(slot),
  cedar::aux::ConstDataPtr data
) const
{
CEDAR_DEBUG_ASSERT(slot->getName() == "input");
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
void BufferToFile::inputConnectionChanged(const std::string& inputName)
{
  CEDAR_DEBUG_ASSERT(inputName == "input");
  if (inputName == "input")
  {
    // Assign the input to the member. This saves us from
        // casting in every computation step.
    this->mInput = boost::shared_dynamic_cast<const cedar::aux::MatData>
          (
            this->getInput(inputName)
          );
    // This should always work since other types should not be accepted.
    if(!this->mInput)
    {
      return;
    }
  }
}
void BufferToFile::compute(const cedar::proc::Arguments&)
```

```
{
const cv::Mat& input = this->mInput->getData();
    ofs << input << "\n";
}
void BufferToFile::onStart()
{
std::string test = getPath();
char * buffer = new char[test.length()];
strcpy(buffer,test.c_str());
ofs.open(buffer);
}
void BufferToFile::onStop()
{
ofs.flush();
ofs.close();
}
void BufferToFile::bufferPathChanged()
{
  //nothing happens here, perhaps in the future useful
}
```

7.2.3 CenterOfVisField.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
#include "cedar/auxiliaries/DoubleParameter.h"
/*!@brief
 *
 * @remarks
 */
class CenterOfVisField : public cedar::proc::Step
{
  //--------------------------------------------------------
  // macros
  //--------------------------------------------------------
  Q_OBJECT
  //--------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------
public:
```

```
  //!@brief The standard constructor.
  CenterOfVisField();
  //--------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------
public:
  //--------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
                                  (
                                    cedar::proc::ConstDataSlotPtr slot,
                                    cedar::aux::ConstDataPtr data
                                  ) const;
  //--------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  public slots:
    void parameterChanged();
  //--------------------------------------------------------------
  // members
  //--------------------------------------------------------------
protected:
  cedar::aux::ConstMatDataPtr mInput;
  //!@brief The data containing the output.
  cedar::aux::MatDataPtr mFovea;
  cedar::aux::MatDataPtr mOutput;
  cedar::aux::MatDataPtr mMaxVal;
private:
  //--------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------
protected:
//
private:
  cedar::aux::IntParameterPtr _mSizeX;
```

```
  cedar::aux::IntParameterPtr _mSizeY;
}; // class cedar::proc::steps::CenterOfVisField
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.4 CenterOfVisField.cpp

```
// CEDAR INCLUDES
#include "CenterOfVisField.h"
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
CenterOfVisField::CenterOfVisField()
:
mFovea(new cedar::aux::MatData(cv::Mat())),
mOutput(new cedar::aux::MatData(cv::Mat())),
mMaxVal(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
_mSizeX(new cedar::aux::IntParameter(this, "size x", 20, 2, 100)),
_mSizeY(new cedar::aux::IntParameter(this, "size y", 20, 2, 100))
{
  // declare all data
  this->declareInput("input");
  this->declareOutput("fovea centralis", mFovea);
  this->declareOutput("max value, useful for c_on", mMaxVal);
  this->declareOutput("output", mOutput);
  QObject::connect
    (
      _mSizeX.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
    );
  QObject::connect
  (
    _mSizeY.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(parameterChanged())
  );
}
void CenterOfVisField::parameterChanged()
{
this->_mSizeX->getValue();
this->_mSizeY->getValue();
```

```
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
void CenterOfVisField::compute(const cedar::proc::Arguments&)
{
const cv::Mat input = this->mInput->getData();
double maxVal;
int rows = input.rows;
int cols = input.cols;
cv::Mat output = this->mOutput->getData();
int sizeX = this->_mSizeX->getValue();
int sizeY = this->_mSizeY->getValue();
int startX = cols/2 - sizeX/2;
int startY = rows/2 - sizeY/2;
//Make a rectangle
cv::Rect roi(startX, startY, sizeX, sizeY);
//Point a cv::Mat header at it (no allocation is done)
cv::Mat image_roi(input, roi);
minMaxLoc(image_roi, NULL, &maxVal, NULL, NULL);
this->mMaxVal->getData().at<float>(0,0) = maxVal;
cv::Mat roiImg;
roiImg = output(roi);
image_roi.copyTo(roiImg);
output = roiImg;
this->mFovea->getData() = image_roi;
}
cedar::proc::DataSlot::VALIDITY CenterOfVisField::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr CEDAR_DEBUG_ONLY(slot),
  cedar::aux::ConstDataPtr data
) const
{
  // First, let's make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(slot->getName() == "input")
  if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
  {
    // Mat data is accepted.
    return cedar::proc::DataSlot::VALIDITY_VALID;
  }
  else
  {
```

```
    // Everything else is rejected.
    return cedar::proc::DataSlot::VALIDITY_ERROR;
  }
}
void CenterOfVisField::inputConnectionChanged(const std::string& inputName)
{
  // Again, let's first make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(inputName == "input");
  // Assign the input to the member. This saves us from
  // casting in every computation step.
  this->mInput = boost::shared_dynamic_cast<const cedar::aux::MatData>
    (
      this->getInput(inputName)
    );
  if(!this->mInput)
  {
    return;
  }
  // Let's get a reference to the input matrix.
  const cv::Mat& input = this->mInput->getData();
  // Make a copy to create a matrix of the same type, dimensions, ...
  this->mOutput->setData(input.clone());
  this->mOutput->copyAnnotationsFrom(this->mInput);
}
```

7.2.5 ColorConversionModified.h

```
#ifndef CEDAR_PROC_STEPS_COLOR_CONVERSION_MODIFIED_H
#define CEDAR_PROC_STEPS_COLOR_CONVERSION_MODIFIED_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/annotation/namespace.h"
#include "cedar/auxiliaries/EnumParameter.h"
// SYSTEM INCLUDES
/*!@brief A processing step that converts an image from rgb to hsv
 */
class ColorConversionModified : public cedar::proc::Step
{
  //----------------------------------------------------------
  // macros
  //----------------------------------------------------------
  Q_OBJECT
```

```
  //--------------------------------------------------------
  // nested types
  //--------------------------------------------------------
public:
  //!@brief Enum class for color spaces.
  class ColorSpace
  {
    public:
      //! the id of an enum entry
      typedef cedar::aux::EnumId Id;
      //! constructs the enum for all ids
      static void construct()
      {
        mType.type()->def(cedar::aux::Enum(AUTO, "AUTO"));
        mType.type()->def(cedar::aux::Enum(BGR, "BGR"));
        mType.type()->def(cedar::aux::Enum(HSV, "HSV"));
      }
      //! @returns A const reference to the base enum object.
      static const cedar::aux::EnumBase& type()
      {
        return *(mType.type());
      }
      //! @returns A pointer to the base enum object.
      static const cedar::proc::DataRole::TypePtr& typePtr()
      {
        return mType.type();
      }
    public:
      //! flag for automatically determining the color space (using annotations)
      static const Id AUTO = 0;
      //! flag for BGR color space (blue, green, red)
      static const Id BGR = 1;
      //! flag for HSV color space (hue, saturation, value)
      static const Id HSV = 2;
    private:
      static cedar::aux::EnumType<ColorSpace> mType;
  };
  //--------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------
public:
  //!@brief The standard constructor.
  ColorConversionModified();
```

```
  //--------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------
public:
  /*!@brief Returns the color space into which the input is transformed.
   */
  inline ColorSpace::Id getTargetColorSpace() const
  {
    return this->_mTargetType->getValue();
  }
  //--------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
                                  (
                                    cedar::proc::ConstDataSlotPtr slot,
                                    cedar::aux::ConstDataPtr data
                                  ) const;
  //--------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  cv::Mat convertImageRGBtoHSV(const cv::Mat imageRGB);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
private slots:
  //!@brief Updates the constant passed to the cv::convert function.
  void updateCvConvertConstant();
  //!@brief Determines the color space of the source image
  void updateSourceImageColorSpace();
  //!@brief Updates the color space of the target image.
  void updateTargetImageColorSpace();
  //!@brief Computes the output again.
  void recompute();
  //--------------------------------------------------------------
  // members
  //--------------------------------------------------------------
protected:
  //not yet
```

```
private:
  cedar::aux::ConstMatDataPtr mInput;
  cedar::aux::annotation::ConstColorSpacePtr mInputColorSpaceAnnotation;
  ColorSpace::Id mInputColorSpace;
  cedar::aux::MatDataPtr mOutput;
  int mCvConversionConstant;
  //----------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------
protected:
//nothing yet
private:
  //! Type of the source image. Usually determined automatically.
  cedar::aux::EnumParameterPtr _mSourceType;
  //! Type of the target image.
  cedar::aux::EnumParameterPtr _mTargetType;
}; // class cedar::proc::steps::ColorConversionModified
#endif // CEDAR_PROC_STEPS_COLOR_CONVERSION_H
```

7.2.6 ColorConversionModified.cpp

```
// CEDAR INCLUDES
#include "ColorConversionModified.h"
#include "cedar/auxiliaries/annotation/ColorSpace.h"
#include "cedar/auxiliaries/MatData.h"
//----------------------------------------------------------------
// static members
//----------------------------------------------------------------
cedar::aux::EnumType<ColorConversionModified::ColorSpace>
ColorConversionModified::ColorSpace::mType("ColorSpace::");
#ifndef CEDAR_COMPILER_MSVC
const ColorConversionModified::ColorSpace::Id ColorConversionModified::ColorSpace::BGR;
const ColorConversionModified::ColorSpace::Id ColorConversionModified::ColorSpace::HSV;
#endif // CEDAR_COMPILER_MSVC
//----------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------
ColorConversionModified::ColorConversionModified()
:
mOutput(new cedar::aux::MatData(cv::Mat(2, 2, CV_32F))),
mCvConversionConstant(-1),
_mSourceType
(
```

```
  new cedar::aux::EnumParameter
  (
    this,
    "source type",
    ColorConversionModified::ColorSpace::typePtr(),
    ColorSpace::AUTO
  )
),
_mTargetType
(
  new cedar::aux::EnumParameter
  (
    this,
    "target type",
    ColorConversionModified::ColorSpace::typePtr(),
    ColorSpace::HSV
  )
)
{
  // declare inputs
  this->declareInput("input image");
  // declare outputs
  this->declareOutput("converted image", mOutput);
  // the target type cannot be determined automatically
  _mTargetType->disable(ColorSpace::AUTO);
  // connect signals from the changed source type
  QObject::connect
  (
    this->_mSourceType.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(updateSourceImageColorSpace())
  );
  QObject::connect
  (
    this->_mSourceType.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(updateCvConvertConstant())
  );
  QObject::connect
  (
    this->_mSourceType.get(),
```

```
    SIGNAL(valueChanged()),
    this,
    SLOT(recompute())
  );
  // connect signals from the changed target type
  QObject::connect
  (
    this->_mTargetType.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(updateTargetImageColorSpace())
  );
  QObject::connect
  (
    this->_mTargetType.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(updateCvConvertConstant())
  );
  QObject::connect
  (
    this->_mTargetType.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(recompute())
  );
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
void ColorConversionModified::recompute()
{
  this->onTrigger();
}
cedar::proc::DataSlot::VALIDITY ColorConversionModified::determineInputValidity
                                (
                                  cedar::proc::ConstDataSlotPtr /* slot */,
                                  cedar::aux::ConstDataPtr data
                                ) const
{
  if (cedar::aux::ConstMatDataPtr mat_data = boost::dynamic_pointer_cast
                                    <const cedar::aux::MatData>(data))
  {
```

```
    try
    {
      data->getAnnotation<cedar::aux::annotation::ColorSpace>();
      return cedar::proc::DataSlot::VALIDITY_VALID;
    }
    catch (cedar::aux::AnnotationNotFoundException)
    {
      return cedar::proc::DataSlot::VALIDITY_WARNING;
    }
  }
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void ColorConversionModified::updateTargetImageColorSpace()
{
  cedar::aux::annotation::ColorSpacePtr target_space;
  switch (this->getTargetColorSpace())
  {
    case ColorSpace::HSV:
      target_space = cedar::aux::annotation::ColorSpacePtr
                     (
                       new cedar::aux::annotation::ColorSpace
                       (
                         cedar::aux::annotation::ColorSpace::Hue,
                         cedar::aux::annotation::ColorSpace::Saturation,
                         cedar::aux::annotation::ColorSpace::Value
                       )
                     );
      break;
    case ColorSpace::BGR:
      target_space = cedar::aux::annotation::ColorSpacePtr
                     (
                       new cedar::aux::annotation::ColorSpace
                       (
                         cedar::aux::annotation::ColorSpace::Blue,
                         cedar::aux::annotation::ColorSpace::Green,
                         cedar::aux::annotation::ColorSpace::Red
                       )
                     );
      break;
    default:
      // unhandled color space, do nothing.
      return;
  }
```

```
  this->mOutput->setAnnotation(target_space);
}
void ColorConversionModified::updateSourceImageColorSpace()
{
  if (!this->mInput)
  {
    return;
  }
  this->_mSourceType->enableAll();
  switch (this->_mSourceType->getValue())
  {
    case ColorSpace::AUTO:
    {
      try
      {
        this->mInputColorSpaceAnnotation = this->mInput->getAnnotation
          <cedar::aux::annotation::ColorSpace>();
        switch (this->mInputColorSpaceAnnotation->getNumberOfChannels())
        {
          case 3:
            if
            (
              this->mInputColorSpaceAnnotation->getChannelType(0)
                == cedar::aux::annotation::ColorSpace::Blue
              && this->mInputColorSpaceAnnotation->getChannelType(1)
                == cedar::aux::annotation::ColorSpace::Green
              && this->mInputColorSpaceAnnotation->getChannelType(2)
                == cedar::aux::annotation::ColorSpace::Red
            )
            {
              this->mInputColorSpace = ColorSpace::BGR;
            }
            else if
            (
              this->mInputColorSpaceAnnotation->getChannelType(0)
                == cedar::aux::annotation::ColorSpace::Hue
              && this->mInputColorSpaceAnnotation->getChannelType(1)
                == cedar::aux::annotation::ColorSpace::Saturation
              && this->mInputColorSpaceAnnotation->getChannelType(2)
                == cedar::aux::annotation::ColorSpace::Value
            )
            {
              this->mInputColorSpace = ColorSpace::HSV;
```

```
            }
            else
            {
              CEDAR_THROW(cedar::aux::UnhandledValueException,
                "The channel combination of the source image is not handled.");
            }
            break;
          default:
            CEDAR_THROW(cedar::aux::UnhandledValueException,
              "This step can currently only process three-channel images.");
        }
      }
      catch (cedar::aux::AnnotationNotFoundException&)
      {
        this->mInputColorSpaceAnnotation.reset();
      }
      break;
    } // case ColorSpace::AUTO
    default:
    {
      this->mInputColorSpace = this->_mSourceType->getValue();
    } // default
  }
  if (!this->mInputColorSpaceAnnotation)
  {
    this->_mSourceType->disable(ColorSpace::AUTO);
  }
  this->_mTargetType->enableAll();
  this->_mTargetType->disable(ColorSpace::AUTO);
  this->_mTargetType->disable(this->mInputColorSpace);
}
void ColorConversionModified::updateCvConvertConstant()
{
  switch (this->mInputColorSpace)
  {
    case ColorSpace::HSV:
      switch (this->getTargetColorSpace())
      {
        case ColorSpace::BGR:
          this->mCvConversionConstant = CV_HSV2BGR;
          break;
      }
      break;
```

```
    case ColorSpace::BGR:
      switch (this->getTargetColorSpace())
      {
        case ColorSpace::HSV:
          this->mCvConversionConstant = CV_BGR2HSV;
          break;
      }
      break;
  }
}
void ColorConversionModified::inputConnectionChanged(const std::string& inputName)
{
  this->mInput = cedar::aux::asserted_pointer_cast
    <const cedar::aux::MatData>(this->getInput(inputName));
  this->mOutput->copyAnnotationsFrom(this->mInput);
  this->updateSourceImageColorSpace();
  this->updateTargetImageColorSpace();
  this->updateCvConvertConstant();
  this->onTrigger();
}
// Create a HSV image from the RGB image using the full 8-bits,
// since OpenCV only allows Hues up to 180 instead of 255.
// ref: "http://cs.haifa.ac.il/hagit/courses/ist/Lectures
//       /Demos/ColorApplet2/t_convert.html"
// Remember to free the generated HSV image.
cv::Mat ColorConversionModified::convertImageRGBtoHSV(const cv::Mat imageRGB)
{
float fR, fG, fB;
float fH, fS, fV;
const float FLOAT_TO_BYTE = 255.0f;
const float BYTE_TO_FLOAT = 1.0f / FLOAT_TO_BYTE;
// Create a blank HSV image
cv::Mat imageHSV = cv::Mat::zeros(imageRGB.rows, imageRGB.cols, CV_8UC3);
int h = imageRGB.rows; // Pixel height.
int w = imageRGB.cols; // Pixel width.
int rowSizeRGB = imageRGB.step;
// Size of row in bytes, including extra padding.
unsigned char *imRGB = imageRGB.data;
// Pointer to the start of the image pixels.
int rowSizeHSV = imageHSV.step;
// Size of row in bytes, including extra padding.
unsigned char *imHSV = imageHSV.data;
// Pointer to the start of the image pixels.
```

```
for (int y=0; y<h; y++) {
for (int x=0; x<w; x++) {
// Get the RGB pixel components. NOTE that OpenCV stores RGB pixels in B,G,R order.
unsigned char *pRGB = (uchar*)(imRGB + y*rowSizeRGB + x*3);
int bB = *(unsigned char*)(pRGB+0); // Blue component
int bG = *(unsigned char*)(pRGB+1); // Green component
int bR = *(unsigned char*)(pRGB+2); // Red component
// Convert from 8-bit integers to floats.
fR = bR * BYTE_TO_FLOAT;
fG = bG * BYTE_TO_FLOAT;
fB = bB * BYTE_TO_FLOAT;
// Convert from RGB to HSV, using float ranges 0.0 to 1.0.
float fDelta;
float fMin, fMax;
int iMax;
// Get the min and max, but use integer comparisons for slight speedup.
if (bB < bG) {
if (bB < bR) {
fMin = fB;
if (bR > bG) {
iMax = bR;
fMax = fR;
}
else {
iMax = bG;
fMax = fG;
}
}
else {
fMin = fR;
fMax = fG;
iMax = bG;
}
}
else {
if (bG < bR) {
fMin = fG;
if (bB > bR) {
fMax = fB;
iMax = bB;
}
else {
fMax = fR;
```

```
iMax = bR;
}
}
else {
fMin = fR;
fMax = fB;
iMax = bB;
}
}
fDelta = fMax - fMin;
fV = fMax; // Value (Brightness).
if (iMax != 0) { // Make sure its not pure black.
fS = fDelta / fMax; // Saturation.
               // Make the Hues between 0.0 to 1.0 instead of 6.0
float ANGLE_TO_UNIT = 1.0f / (6.0f * fDelta);
if (iMax == bR) { // between yellow and magenta.
fH = (fG - fB) * ANGLE_TO_UNIT;
}
else if (iMax == bG) { // between cyan and yellow.
fH = (2.0f/6.0f) + ( fB - fR ) * ANGLE_TO_UNIT;
}
else { // between magenta and cyan.
fH = (4.0f/6.0f) + ( fR - fG ) * ANGLE_TO_UNIT;
}
// Wrap outlier Hues around the circle.
if (fH < 0.0f)
fH += 1.0f;
if (fH >= 1.0f)
fH -= 1.0f;
}
else {
// color is pure Black.
fS = 0;
fH = 0; // undefined hue
}
// Convert from floats to 8-bit integers.
int bH = (int)(0.5f + fH * 255.0f);
int bS = (int)(0.5f + fS * 255.0f);
int bV = (int)(0.5f + fV * 255.0f);
// Clip the values to make sure it fits within the 8bits.
if (bH > 255)
bH = 255;
if (bH < 0)
```

```
bH = 0;
if (bS > 255)
bS = 255;
if (bS < 0)
bS = 0;
if (bV > 255)
bV = 255;
if (bV < 0)
bV = 0;
// Set the HSV pixel components.
unsigned char *pHSV = (uchar*)(imHSV + y*rowSizeHSV + x*3);
*(pHSV+0) = bH; // H component
*(pHSV+1) = bS; // S component
*(pHSV+2) = bV; // V component
}
}
return imageHSV;
}
void ColorConversionModified::compute(const cedar::proc::Arguments&)
{
  cv::Mat& output = this->mOutput->getData();
  const cv::Mat& imRGB = this->mInput->getData();
  cv::Mat imHSV = convertImageRGBtoHSV(imRGB);
  output = imHSV;
}
```

7.2.7 DistanceError.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
class DistanceError : public cedar::proc::Step
{
  //----------------------------------------------------------------
  // macros
  //----------------------------------------------------------------
  Q_OBJECT
  //----------------------------------------------------------------
  // constructors and destructor
  //----------------------------------------------------------------
public:
```

```
  //!@brief The standard constructor.
  DistanceError();
  //--------------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------------
public:
  //--------------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
                                  (
                                    cedar::proc::ConstDataSlotPtr slot,
                                    cedar::aux::ConstDataPtr data
                                  ) const;
  //--------------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  //--------------------------------------------------------------------
  // members
  //--------------------------------------------------------------------
protected:
  cedar::aux::ConstMatDataPtr mInput;
  //!@brief The data containing the output.
  cedar::aux::MatDataPtr mOutput;
  cedar::aux::MatDataPtr mDistanceErrorOblique;
  cedar::aux::MatDataPtr mDistanceErrorX;
  cedar::aux::MatDataPtr mDistanceErrorY;
 // cedar::aux::DoubleDataPtr mY;
private:
  //--------------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------------
protected:
//
private:
}; // class cedar::proc::steps::DistanceError
```

```
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.8 DistanceError.cpp

```
// CEDAR INCLUDES
#include "DistanceError.h"
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
//----------------------------------------------------------------------
// register the class
//----------------------------------------------------------------------
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr distance_error_decl
    (
      new ElementDeclarationTemplate<DistanceError>
      (
        "BellSteps",
        "DistanceError"
      )
    );
    distance_error_decl->setIconPath(":/steps/distance_error.svg");
    distance_error_decl->setDescription
    (
      "Calculates the distance between the center of a field and its maximum input"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass
      (
        distance_error_decl
      );
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
DistanceError::DistanceError()
```

```
:
// outputs
mOutput(new cedar::aux::MatData(cv::Mat())),
mDistanceErrorOblique(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
mDistanceErrorX(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
mDistanceErrorY(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F)))
{
  // declare all data
  this->declareInput("input");
  this->declareOutput("output", mOutput);
  this->declareOutput("distance error oblique", mDistanceErrorOblique);
  this->declareOutput("distance error x", mDistanceErrorX);
  this->declareOutput("distance error y", mDistanceErrorY);
  //this->declareOutput("y", mY);
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
void DistanceError::compute(const cedar::proc::Arguments&)
{
const cv::Mat input = this->mInput->getData();
double maxVal,minVal;
cv::Point minLoc, maxLoc;
int sizeX = input.cols;
int sizeY = input.rows;
minMaxLoc(input, &minVal, &maxVal, &minLoc, &maxLoc);
this->mDistanceErrorX->getData().at<float>(0,0) = maxLoc.x - sizeX/2;
this->mDistanceErrorY->getData().at<float>(0,0) = maxLoc.y - sizeY/2;
double radius = sqrt(pow(this->mDistanceErrorX->getData().at<float>(0,0),2.0)
+ pow(this->mDistanceErrorY->getData().at<float>(0,0),2.0));
this->mDistanceErrorOblique->getData().at<float>(0,0) = radius;
}
cedar::proc::DataSlot::VALIDITY DistanceError::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr CEDAR_DEBUG_ONLY(slot),
  cedar::aux::ConstDataPtr data
) const
{
  // First, let's make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(slot->getName() == "input")
  if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
  {
```

```
    // Mat data is accepted.
    return cedar::proc::DataSlot::VALIDITY_VALID;
  }
  else
  {
    // Everything else is rejected.
    return cedar::proc::DataSlot::VALIDITY_ERROR;
  }
}
void DistanceError::inputConnectionChanged(const std::string& inputName)
{
  // Again, let's first make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(inputName == "input");
  // Assign the input to the member. This saves us
  // from casting in every computation step.
  this->mInput = boost::shared_dynamic_cast
    <const cedar::aux::MatData>
    (
      this->getInput(inputName)
    );
  if(!this->mInput)
  {
    return;
  }
  // Let's get a reference to the input matrix.
  const cv::Mat& input = this->mInput->getData();
  // Make a copy to create a matrix of the same type, dimensions, ...
  this->mOutput->setData(input.clone());
  this->mOutput->copyAnnotationsFrom(this->mInput);
}
```

7.2.9 Fixation.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
#include <cedar/auxiliaries/NumericParameter.h>
/*!@brief
 *
 * @remarks
```

```
 */
class Fixation : public cedar::proc::Step
{
  //--------------------------------------------------------
  // macros
  //--------------------------------------------------------
  Q_OBJECT
  //--------------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------------
public:
  //!@brief The standard constructor.
  Fixation();
  //--------------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------------
public:
  /*!@brief Whether the fixation is active or not.
   */
  bool fixationIsActive() const;
public slots:
void parameterChanged();
  //--------------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
                                  (
                                    cedar::proc::ConstDataSlotPtr slot,
                                    cedar::aux::ConstDataPtr data
                                  ) const;
  //--------------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  //--------------------------------------------------------------------
  // members
  //--------------------------------------------------------------------
```

```
protected:
  cedar::aux::ConstMatDataPtr mCOn;
  cedar::aux::ConstMatDataPtr mCOn2;
  cedar::aux::ConstMatDataPtr mRateCode1;
  cedar::aux::ConstMatDataPtr mRateCode2;
  //!@brief The data containing the output.
  cedar::aux::MatDataPtr mVelVec;
private:
  //----------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------
protected:
  //!@brief Parameter that lets the user decide
  //whether the fixation should be active or not.
  cedar::aux::BoolParameterPtr _mFixationActive;
private:
  cedar::aux::DoubleParameterPtr _mGain;
}; // class cedar::proc::steps::Fixation
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.10 Fixation.cpp

```
// CEDAR INCLUDES
#include "Fixation.h"
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
#include <cedar/auxiliaries/DoubleData.h>
//----------------------------------------------------------------
// register the class
//----------------------------------------------------------------
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr fixation_decl
    (
      new ElementDeclarationTemplate<Fixation>
      (
        "BellSteps",
        "Fixation"
      )
```

```
    );
    fixation_decl->setIconPath(":/steps/static_gain.svg");
    fixation_decl->setDescription
    (
      "Simulation of head"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass
      (
        fixation_decl
      );
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
Fixation::Fixation()
:
  mCOn(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
  mCOn2(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mRateCode1(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mRateCode2(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mVelVec(new cedar::aux::MatData(cv::Mat(2, 1, CV_32F))),
  _mFixationActive
  (
    new cedar::aux::BoolParameter(this, "fixation active", false)),
    _mGain(new cedar::aux::DoubleParameter(this, "gain", 1.0, -10000.0, 10000.0)
  )
{
// declare all data
this->declareInput("rate_code1");
this->declareInput("rate_code2");
this->declareInput("c_on");
this->declareInput("c_on saccade pause");
this->declareOutput("velocity vector", mVelVec);
QObject::connect
    (
      _mGain.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
```

```
    );
}
void Fixation::parameterChanged()
{
this->_mGain->getValue();
}
//----------------------------------------------------------------------
// methods
//----------------------------------------------------------------------
bool Fixation::fixationIsActive() const
{
  return this->_mFixationActive->getValue();
}
void Fixation::compute(const cedar::proc::Arguments&)
{
double c_on = this->mCOn->getData().at<float>(0,0);
double c_on2 = this->mCOn2->getData().at<float>(0,0);
double rate_code1 = this->mRateCode1->getData().at<float>(0,0);
double rate_code2 = this->mRateCode2->getData().at<float>(0,0);
double gain = this->_mGain->getValue();
if(c_on > 0.1 && fixationIsActive() && c_on2 == 1.0)
{
this->mVelVec->getData().at<float>(0,0) = c_on * rate_code1 * gain;
this->mVelVec->getData().at<float>(1,0) = c_on * rate_code2 * gain;
}
else
{
this->mVelVec->getData().at<float>(0,0) = 0.0;
this->mVelVec->getData().at<float>(1,0) = 0.0;
}
}
cedar::proc::DataSlot::VALIDITY Fixation::determineInputValidity
(
cedar::proc::ConstDataSlotPtr slot,
cedar::aux::ConstDataPtr data
) const
{
if (this->getInputSlot("c_on saccade pause") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
```

```
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
else if (this->getInputSlot("c_on") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
else if (this->getInputSlot("rate_code1") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
else if (this->getInputSlot("rate_code2") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
```

```
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void Fixation::inputConnectionChanged(const std::string& inputName)
{
cedar::aux::ConstDataPtr data = this->getInput(inputName);
if (inputName == "c_on saccade pause")
{
this->mCOn2.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
              boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
this->mCOn2 = mat_data;
}
else
{
return;
}
}
else if (inputName == "c_on")
{
this->mCOn.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
              boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
      this->mCOn = mat_data;
    }
     else
     {
      return;
     }
  }
  else if (inputName == "rate_code1")
  {
  this->mRateCode1.reset();
  if (cedar::aux::ConstMatDataPtr mat_data =
                boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
  {
  this->mRateCode1 = mat_data;
  }
  else
```

```
  {
  return;
  }
  }
  else if (inputName == "rate_code2")
  {
  this->mRateCode2.reset();
  if (cedar::aux::ConstMatDataPtr mat_data =
              boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
  {
  this->mRateCode2 = mat_data;
  }
  else
  {
  return;
  }
  }
}
```

7.2.11 HopOscillator.h

```
#ifndef DYNAMICS_HOPF_OSCILLATOR_H
#define DYNAMICS_HOPF_OSCILLATOR_H
#include <cedar/dynamics/Dynamics.h>
#include <cedar/auxiliaries/NumericParameter.h>
#include "cedar/auxiliaries/ObjectParameterTemplate.h"
class HopfOscillator : public cedar::dyn::Dynamics
{
  //----------------------------------------------------------------
  // macros
  //----------------------------------------------------------------
  Q_OBJECT
  //----------------------------------------------------------------
  // nested types
  //----------------------------------------------------------------
public:
  //!@brief a parameter for sigmoid objects
  typedef cedar::aux::ObjectParameterTemplate
     <cedar::aux::math::TransferFunction> SigmoidParameter;
  //!@cond SKIPPED_DOCUMENTATION
  CEDAR_GENERATE_POINTER_TYPES_INTRUSIVE(SigmoidParameter);
  //!@endcond
  //----------------------------------------------------------------
```

```
  // constructors and destructor
  //----------------------------------------------------------------------
public:
  //!@brief The standard constructor.
  HopfOscillator();
  //----------------------------------------------------------------------
  // public methods
  //----------------------------------------------------------------------
public:
  void eulerStep(const cedar::unit::Time& time);
  void onStart();
  void onStop();
  //----------------------------------------------------------------------
  // protected methods
  //----------------------------------------------------------------------
protected:
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr,
    cedar::aux::ConstDataPtr
  ) const;
  void inputConnectionChanged(const std::string& inputName);
  //----------------------------------------------------------------------
  // private methods
  //----------------------------------------------------------------------
private:
public slots:
  void parameterChanged();
  //----------------------------------------------------------------------
  // members
  //----------------------------------------------------------------------
protected:
  cedar::aux::MatDataPtr mSigmoidalActivation;
  cedar::aux::MatDataPtr mSigmoidalActivationRelax;
private:
  // inputs
  cedar::aux::ConstMatDataPtr mCOn;
  cedar::aux::ConstMatDataPtr mMu;
  // buffers
  cedar::aux::DoubleDataPtr mDotX;
  cedar::aux::DoubleDataPtr mDotY;
  cedar::aux::ConstMatDataPtr mLearnedGain;
  // outputs
```

```
  cedar::aux::DoubleDataPtr mX;
  cedar::aux::DoubleDataPtr mY;
  //buffers
  cedar::aux::DoubleDataPtr mDotX2;
  cedar::aux::DoubleDataPtr mDotY2;
  cedar::aux::DoubleDataPtr mX2;
  cedar::aux::DoubleDataPtr mY2;
  cedar::aux::DoubleDataPtr mMuBlocked;
  cedar::aux::DoubleDataPtr mRelax;
  cedar::aux::DoubleDataPtr mCOnPause;
  // outputs
  cedar::aux::MatDataPtr mOutput;
  cedar::aux::MatDataPtr mOutput2;
  cedar::aux::MatDataPtr mDoLearning;
  cedar::aux::MatDataPtr mBufferThiefOutput;
  //----------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------
private:
  cedar::aux::DoubleParameterPtr _mTau;
  cedar::aux::DoubleParameterPtr _mTauRelax;
  cedar::aux::DoubleParameterPtr _mAngularSpeed;
  cedar::aux::DoubleParameterPtr _mGamma;
  SigmoidParameterPtr _mSigmoid;
  SigmoidParameterPtr _mSigmoidRelax;
  cedar::aux::DoubleParameterPtr _mGain;
}; // class utilities::SteeringFilter
#endif // DYNAMICS_HOPF_OSCILLATOR_H
```

7.2.12 HopfOscillator.cpp

```
#include "HopfOscillator.h"
#include <cedar/auxiliaries/MatData.h>
#include <cedar/auxiliaries/DoubleData.h>
#include <cedar/auxiliaries/math/constants.h>
#include <cmath>
#include "cedar/dynamics/fields/NeuralField.h"
#include "cedar/auxiliaries/math/sigmoids/ExpSigmoid.h"
//----------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------
HopfOscillator::HopfOscillator()
:
```

```
mSigmoidalActivation(new cedar::aux::MatData
(
  cv::Mat::zeros(1, 1, CV_32F))
),
mSigmoidalActivationRelax(new cedar::aux::MatData
(
  cv::Mat::zeros(1, 1, CV_32F))
),
mDotX(new cedar::aux::DoubleData(0.0)),
mDotY(new cedar::aux::DoubleData(0.0)),
mLearnedGain(new cedar::aux::MatData
(
  cv::Mat::ones(1, 1, CV_32F))
),
mX(new cedar::aux::DoubleData(0.01)),
mY(new cedar::aux::DoubleData(-1.0)),
mDotX2(new cedar::aux::DoubleData(0.0)),
mDotY2(new cedar::aux::DoubleData(0.0)),
mX2(new cedar::aux::DoubleData(0.1)),
mY2(new cedar::aux::DoubleData(-1.0)),
mMuBlocked(new cedar::aux::DoubleData(0.0)),
mRelax(new cedar::aux::DoubleData(0.0)),
mOutput(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
mOutput2(new cedar::aux::MatData
(
  cv::Mat::ones(1,1, CV_32F))
),
mDoLearning(new cedar::aux::MatData
(
  cv::Mat::zeros(1, 1, CV_32F))
),
mBufferThiefOutput(new cedar::aux::MatData
(
  cv::Mat(2,1, CV_32F))
),
_mTau(new cedar::aux::DoubleParameter
(
  this,
  "tau",
  1.0,
  0.001,
  10000.0
)),
```

```
    _mTauRelax(new cedar::aux::DoubleParameter
    (
      this,
      "tau relax",
      1.0,
      0.001,
      10000.0
    )),
    _mAngularSpeed(new cedar::aux::DoubleParameter
    (
      this,
      "angularSpeed",
      200.0*cedar::aux::math::pi,
      0.0,
      100000.0
    )),
    _mGamma(new cedar::aux::DoubleParameter
    (
      this,
      "gamma",
      0.1,
      0.0,
      1000.0
    )),
    _mSigmoid
    (
      new cedar::dyn::NeuralField::SigmoidParameter
  (
    this,
    "sigmoid saccade",
    cedar::aux::math::SigmoidPtr
        (
          new cedar::aux::math::ExpSigmoid
          (
            1.0,
            1000.0
          )
        )
  )
    ),
    _mSigmoidRelax
    (
      new cedar::dyn::NeuralField::SigmoidParameter
```

```
(
  this,
  "sigmoid relax",
  cedar::aux::math::SigmoidPtr
      (
        new cedar::aux::math::ExpSigmoid
        (
          1.0,
          1000.0
        )
      )
)
  ),
  _mGain(new cedar::aux::DoubleParameter
  (
    this,
    "gain",
    1.0,
    -10000.0,
    10000.0
  ))
{
// inputs
this->declareInput("mMu", false);
this->declareInput("mLearnedGain", false);
this->declareInput("mCOn", false);
// buffers
this->declareBuffer("dot(x)", mDotX);
this->declareBuffer("dot(y)", mDotY);
this->declareBuffer("dot2(x)", mDotX2);
    this->declareBuffer("dot2(y)", mDotY2);
    // outputs
    this->declareOutput("x", mX);
    this->declareOutput("y", mY);
    this->declareOutput("output", mOutput);
    this->declareOutput("output2", mOutput2);
    this->declareOutput("do learning", mDoLearning);
    this->declareOutput("data for matlab", mBufferThiefOutput);
    // connect the parameter's change signal
    QObject::connect
    (
      _mTau.get(),
      SIGNAL(valueChanged()),
```

```
      this,
      SLOT(parameterChanged())
    );
    QObject::connect
    (
      _mTauRelax.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
    );
    QObject::connect
    (
      _mAngularSpeed.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
    );
    QObject::connect
    (
      _mGamma.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
    );
    QObject::connect
    (
      _mGain.get(),
      SIGNAL(valueChanged()),
      this,
      SLOT(parameterChanged())
    );
}
//----------------------------------------------------------------------
// methods
//----------------------------------------------------------------------
cedar::proc::DataSlot::VALIDITY HopfOscillator::determineInputValidity
(
cedar::proc::ConstDataSlotPtr slot,
cedar::aux::ConstDataPtr data
) const
{
if (this->getInputSlot("mMu") == slot)
{
```

```
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
   {
// Mat data is accepted.
   return cedar::proc::DataSlot::VALIDITY_VALID;
   }
   else
   {
   // Everything else is rejected.
   return cedar::proc::DataSlot::VALIDITY_ERROR;
   }
}
else if (this->getInputSlot("mCOn") == slot)
{
  if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
  {
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
  }
  else
  {
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
  }
}
else if (this->getInputSlot("mLearnedGain") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
     return cedar::proc::DataSlot::VALIDITY_VALID;
     }
     else
     {
     // Everything else is rejected.
     return cedar::proc::DataSlot::VALIDITY_ERROR;
     }
}
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void HopfOscillator::parameterChanged()
{
this->_mGain->getValue();
this->_mGamma->getValue();
```

```
this->_mTau->getValue();
this->_mTauRelax->getValue();
this->_mAngularSpeed->getValue();
}
void HopfOscillator::inputConnectionChanged(const std::string& inputName)
{
  cedar::aux::ConstDataPtr data = this->getInput(inputName);
  if (inputName == "mMu")
  {
  // Assign the input to the member.
      // This saves us from casting in every computation step.
  this->mMu = boost::shared_dynamic_cast
        <const cedar::aux::MatData>(this->getInput(inputName));
  // This should always work since other types should not be accepted.
  if(!this->mMu)
  {
  return;
  }
  }
  else if (inputName == "mCOn")
  {
  // Assign the input to the member.
      // This saves us from casting in every computation step.
  this->mCOn = boost::shared_dynamic_cast
        <const cedar::aux::MatData>(this->getInput(inputName));
  // This should always work since other types should not be accepted.
  if(!this->mCOn)
  {
  return;
  }
  }
  else if (inputName == "mLearnedGain")
  {
  // Assign the input to the member.
      // This saves us from casting in every computation step.
  this->mLearnedGain = boost::shared_dynamic_cast
        <const cedar::aux::MatData>(this->getInput(inputName));
  // This should always work since other types should not be accepted.
  if(!this->mLearnedGain)
  {
  return;
  }
  }
```

```
}
void HopfOscillator::eulerStep(const cedar::unit::Time& time)
{
double& dot_x = this->mDotX->getData();
double& dot_y = this->mDotY->getData();
double& x = this->mX->getData();
double& y = this->mY->getData();
double& muTemp = this->mMuBlocked->getData();
double tau = this->_mTau->getValue();
double gain = this->_mGain->getValue();
double omega = this->_mAngularSpeed->getValue() / 1000.0;
double mu;
double c_on = this->mCOn->getData().at<float>(0,0);
double gamma = this->_mGamma->getValue();
mu = this->mMu->getData().at<float>(0,0);
mu = gain * mu * this->mLearnedGain->getData().at<float>(0,0);
if(muTemp == 0.0 && c_on == 0.0)
{
this->mX->getData() = 0.01;
x = 0.01;
//To start on the limit cycle
if(fabs(mu) > 0.01)
{
y = -sqrt(pow(mu,2.0)-pow(0.01,2.0));
this->mY->getData() = -sqrt(pow(mu,2.0)-pow(0.01,2.0));
}
else
{
y = -fabs(mu);
this->mY->getData() = -fabs(mu);
}
this->mDotX->getData() = 0.0;
dot_x = 0.0;
this->mDotY->getData() = 0.0;
dot_y = 0.0;
muTemp = 1.0;
}
    double pow_x = pow(x, 2.0);
    double pow_y = pow(y, 2.0);
    double pow_mu = mu* mu;
    double powXPlusPowY = pow_x + pow_y;
    double timeDivTau =
           cedar::unit::Milliseconds(time) / cedar::unit::Milliseconds(tau);
```

```
//saccadic Hopf oscillator
dot_x = timeDivTau * ( c_on * (gamma*(pow_mu - (powXPlusPowY) ) * x - omega * y))
        -  (1. - c_on) * x ;
dot_y = timeDivTau * ( c_on * (gamma*(pow_mu - (powXPlusPowY) ) * y + omega * x))
        -  (1. - c_on) * y ;
cv::Mat& sigmoid_u = this->mSigmoidalActivation->getData();
if(c_on == 1.0)
{
 pow_x = pow(x+dot_x, 2.0);
 this->mSigmoidalActivation->getData().at<float>(0,0) = 2. * exp(-10000. * pow_x);
 sigmoid_u = _mSigmoid->getValue()->compute<float>
 (
 mSigmoidalActivation->getData()
 );
}
cv::Mat& sigmoid_uRelax = this->mSigmoidalActivationRelax->getData();
if(sigmoid_u.at<float>(0,0) == 1.0)
{
 this->mOutput->getData().at<float>(0,0) = 0.0;
 double amp = 0.85;
 if(this->mRelax->getData() == 0.0)
 {
 this->mOutput2->getData().at<float>(0,0) = 0.0;
 this->mX2->getData() = 0.1;
 this->mY2->getData() = -sqrt(fabs(fabs(amp) - 0.01));
 this->mDotX2->getData() = 0.0;
 this->mDotY2->getData() = 0.0;
 this->mRelax->getData() = 1.0;
 this->mDoLearning->getData().at<float>(0,0) = 1.0;
 }
 else
 {
 double tauRelax = this->_mTauRelax->getValue();
 double& dot_x2 = this->mDotX2->getData();
 double& dot_y2 = this->mDotY2->getData();
 double& x2 = this->mX2->getData();
 double& y2 = this->mY2->getData();
 double pow_x2 = pow(x2, 2.0);
 double pow_y2 = pow(y2, 2.0);
 double powXPlusPowY2 = pow_x2 + pow_y2;
 double pow_mu2 = amp* amp;
 double timeStep = cedar::unit::Milliseconds(time)
```

```
                        / cedar::unit::Milliseconds(tauRelax);
 //------------------------------------------------------------
 //pause oscillator
 dot_x2 = timeStep * ( gamma*(pow_mu2 - (powXPlusPowY2) ) * x2 - omega * y2);
 dot_y2 = timeStep * ( gamma*(pow_mu2 - (powXPlusPowY2) ) * y2 + omega * x2);
 //------------------------------------------------------------
 this->mSigmoidalActivationRelax->getData().at<float>(0,0) =
             2.*exp(-100. * pow(x2,2));
 sigmoid_uRelax = _mSigmoidRelax->getValue()->compute<float>
 (
 mSigmoidalActivationRelax->getData()
 );
 x2 += dot_x2;
     y2 += dot_y2;
     this->mBufferThiefOutput->getData().at<float>(0,0) = 0.0;
     this->mBufferThiefOutput->getData().at<float>(1,0) = this->mX2->getData();
     if (sigmoid_uRelax.at<float>(0,0) == 1.0)
     {
     this->mDoLearning->getData().at<float>(0,0) = 0.0;
     this->mOutput2->getData().at<float>(0,0) = 1.0;
     if (c_on == 1.0)
     {
     this->mRelax->getData() = 0.0;
     muTemp = 0.0;
     this->mMuBlocked->getData() = 0.0;
     }
     }
     else
     {
     //resting level down
     this->mOutput2->getData().at<float>(0,0) = 0.0;
     }
 }
}
else
{
 x += dot_x;
 y += dot_y;
 this->mBufferThiefOutput->getData().at<float>(0,0) = this->mX->getData();
 this->mBufferThiefOutput->getData().at<float>(1,0) = 0.0;
 if(mu < 0)
 {
 this->mOutput->getData().at<float>(0,0) = -1. * x;
```

```
      }
      else
      {
      this->mOutput->getData().at<float>(0,0) =  x;
      }
     }
}
void HopfOscillator::onStart()
{
this->mOutput2->getData().at<float>(0,0) = 1.0;
this->mX->getData() = 0.01;
this->mY->getData() = -1.0;
this->mDotX->getData() = 0.0;
this->mDotY->getData() = 0.0;
this->mMuBlocked->getData() = 0.0;
this->mRelax->getData() = 0.0;
}
void HopfOscillator::onStop()
{
this->mOutput2->getData().at<float>(0,0) = 1.0;
this->mOutput->getData().at<float>(0,0) = 0.0;
this->mX->getData() = 0.01;
this->mY->getData() = -1.0;
}
```

7.2.13 InterStep.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
#include <cedar/auxiliaries/NumericParameter.h>
class InterStep : public cedar::proc::Step
{
  //--------------------------------------------------------------------
  // macros
  //--------------------------------------------------------------------
  Q_OBJECT
  //--------------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------------
public:
```

```
  //!@brief The standard constructor.
InterStep();
  //----------------------------------------------------------------------
  // public methods
  //----------------------------------------------------------------------
public:
  //----------------------------------------------------------------------
  // protected methods
  //----------------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
  cedar::proc::ConstDataSlotPtr slot,
  cedar::aux::ConstDataPtr data
  ) const;
  //----------------------------------------------------------------------
  // private methods
  //----------------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  //----------------------------------------------------------------------
  // members
  //----------------------------------------------------------------------
protected:
  cedar::aux::ConstMatDataPtr mRestingLevel1;
  cedar::aux::ConstMatDataPtr mRestingLevel2;
  //!@brief The data containing the output.
  cedar::aux::MatDataPtr mRestingLevelOutput;
  cedar::aux::MatDataPtr mCOnFixation;
  public slots:
    void parameterChanged();
  //----------------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------------
private:
  cedar::aux::DoubleParameterPtr _mRestingSwitch;
protected:
private:
}; // class cedar::proc::steps::InterStep
```

```
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.14 InterStep.cpp

```
// CEDAR INCLUDES
#include "InterStep.h"
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
#include <cedar/auxiliaries/DoubleData.h>
//----------------------------------------------------------------------
// register the class
//----------------------------------------------------------------------
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr InterStep_decl
    (
      new ElementDeclarationTemplate<InterStep>
      (
        "BellSteps",
        "InterStep"
      )
    );
    InterStep_decl->setIconPath(":/steps/static_gain.svg");
    InterStep_decl->setDescription
    (
      "Simulation of head"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass(InterStep_decl);
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
InterStep::InterStep()
:
  mRestingLevel1(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
```

```
  mRestingLevel2(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mRestingLevelOutput(new cedar::aux::MatData(cv::Mat::ones(1, 1, CV_32F))),
  mCOnFixation(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
  _mRestingSwitch(new cedar::aux::DoubleParameter
  (
    this,
    "resting level switch",
    1.0,
    -10000.0,
    10000.0
  ))
{
  this->declareInput("resting_level1");
  this->declareInput("resting_level2");
  this->declareOutput("resting level", mRestingLevelOutput);
  this->declareOutput("c_on fixation", mCOnFixation);
  QObject::connect
  (
    _mRestingSwitch.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(parameterChanged())
  );
}
void InterStep::parameterChanged()
{
this->_mRestingSwitch->getValue();
}
//----------------------------------------------------------------------
// methods
//----------------------------------------------------------------------
void InterStep::compute(const cedar::proc::Arguments&)
{
double resting_level1 = this->mRestingLevel1->getData().at<float>(0,0);
double resting_level2 = this->mRestingLevel2->getData().at<float>(0,0);
double restingSwitch = this->_mRestingSwitch->getValue();
this->mRestingLevelOutput->getData().at<float>(0,0) =
          restingSwitch * (1.0 - (resting_level1 * resting_level2));
this->mCOnFixation->getData().at<float>(0,0) =
          1.0 - (resting_level1 * resting_level2);
}
cedar::proc::DataSlot::VALIDITY InterStep::determineInputValidity
(
```

```
cedar::proc::ConstDataSlotPtr slot,
cedar::aux::ConstDataPtr data
) const
{
if (this->getInputSlot("resting_level1") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
else if (this->getInputSlot("resting_level2") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void InterStep::inputConnectionChanged(const std::string& inputName)
{
cedar::aux::ConstDataPtr data = this->getInput(inputName);
if (inputName == "resting_level1")
{
this->mRestingLevel1.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
                boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
this->mRestingLevel1 = mat_data;
}
```

```
else
{
return;
}
}
else if (inputName == "resting_level2")
{
this->mRestingLevel2.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
                boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
this->mRestingLevel2 = mat_data;
}
else
{
return;
}
}
}
```

7.2.15 LearningDynamic.h

```
#ifndef DYNAMICS_HOPF_OSCILLATOR_H
#define DYNAMICS_HOPF_OSCILLATOR_H
#include <cedar/dynamics/Dynamics.h>
#include <cedar/auxiliaries/NumericParameter.h>
#include "cedar/auxiliaries/UIntParameter.h"
#include "cedar/auxiliaries/DoubleVectorParameter.h"
#include "cedar/auxiliaries/ObjectParameterTemplate.h"
class LearningDynamic : public cedar::dyn::Dynamics
{
  //----------------------------------------------------------------
  // macros
  //----------------------------------------------------------------
  Q_OBJECT
  //----------------------------------------------------------------
  // nested types
  //----------------------------------------------------------------
public:
  //!@brief a parameter for sigmoid objects
  typedef cedar::aux::ObjectParameterTemplate
          <cedar::aux::math::TransferFunction> SigmoidParameter;
  //!@cond SKIPPED_DOCUMENTATION
```

```
  CEDAR_GENERATE_POINTER_TYPES_INTRUSIVE(SigmoidParameter);
  //!@endcond
  //--------------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------------
public:
  //!@brief The standard constructor.
  LearningDynamic();
  void resetLearnedGain();
  //--------------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------------
public:
  void eulerStep(const cedar::unit::Time& time);
  void onStart();
  void onStop();
  //--------------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------------
protected:
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr,
    cedar::aux::ConstDataPtr
  ) const;
  void inputConnectionChanged(const std::string& inputName);
  //--------------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------------
private:
  //!@brief Resets the field.
  void reset();
  //!@brief update the size of internal matrices
  void updateMatrices();
public slots:
  void parameterChanged();
  //!@brief handle a change in size along dimensions,
  //which leads to creating new matrices
  void dimensionSizeChanged();
  //--------------------------------------------------------------------
  // members
  //--------------------------------------------------------------------
protected:
```

```
  cedar::aux::MatDataPtr mSigmoidalActivation;
private:
  // inputs
  cedar::aux::ConstMatDataPtr mErrorSign;
  cedar::aux::ConstMatDataPtr mInput;
  cedar::aux::ConstMatDataPtr mDoLearning;
  cedar::aux::MatDataPtr mTargetTemp;
  //outputs
  cedar::aux::MatDataPtr mOutput;
  //----------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------
private:
  cedar::aux::DoubleParameterPtr _mTau;
  SigmoidParameterPtr _mSigmoid;
  cedar::aux::UIntParameterPtr _mSize;
}; // class utilities::SteeringFilter
#endif // DYNAMICS_HOPF_OSCILLATOR_H
```

7.2.16 LearningDynamic.cpp

```
#include "LearningDynamic.h"
#include <cedar/auxiliaries/MatData.h>
#include <cedar/auxiliaries/DoubleData.h>
#include <cedar/auxiliaries/math/constants.h>
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
#include <cmath>
#include "cedar/dynamics/fields/NeuralField.h"
#include "cedar/auxiliaries/math/sigmoids/ExpSigmoid.h"
//----------------------------------------------------------------------
// register the class
//----------------------------------------------------------------------
namespace
{
bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr LearningDynamic_decl
    (
      new ElementDeclarationTemplate<LearningDynamic>
      (
```

```
        "BellSteps",
        "LearningDynamic"
      )
    );
    LearningDynamic_decl->setIconPath(":/steps/static_gain.svg");
    LearningDynamic_decl->setDescription
    (
      "Simulation of head"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass(LearningDynamic_decl);
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
LearningDynamic::LearningDynamic()
:
  mSigmoidalActivation(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
  mDoLearning(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
  mTargetTemp(new cedar::aux::MatData(cv::Mat::zeros(96, 1, CV_32F))),
  mOutput(new cedar::aux::MatData(cv::Mat::ones(96, 1, CV_32F))),
  _mTau(new cedar::aux::DoubleParameter(this, "tau", 1.0, 0.001, 10000.0)),
  _mSigmoid
  (
    new cedar::dyn::NeuralField::SigmoidParameter
    (
      this,
  "sigmoid",
  cedar::aux::math::SigmoidPtr(new cedar::aux::math::ExpSigmoid(1.0, 10000.0))
    )
  ),
  _mSize
  (
    new cedar::aux::UIntParameter
    (
  this,
  "size",
  2,
  10,
  1000
```

```
)
  )
{
  // inputs
  this->declareInput("mInput", false);
  this->declareInput("mErrorSign", false);
  this->declareInput("do learning (c_on)", false);
  this->declareOutput("output", mOutput);
  // connect the parameter's change signal
  QObject::connect
  (
    _mTau.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(parameterChanged())
  );
  QObject::connect
  (
    _mSize.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(dimensionSizeChanged())
  );
  this->registerFunction
  (
    "reset learned error",
    boost::bind(&LearningDynamic::resetLearnedGain, this)
  );
}
void LearningDynamic::resetLearnedGain()
{
this->mOutput->setData(cv::Mat::ones(this->_mSize->getValue(), 1, CV_32F));
}
void LearningDynamic::dimensionSizeChanged()
{
  this->updateMatrices();
}
void LearningDynamic::updateMatrices()
{
    this->lockAll();
    this->mTargetTemp->setData(cv::Mat::ones(this->_mSize->getValue(), 1, CV_32F));
    this->mOutput->setData(cv::Mat::ones(this->_mSize->getValue(), 1, CV_32F));
    this->unlockAll();
```

```
}
cedar::proc::DataSlot::VALIDITY LearningDynamic::determineInputValidity
(
cedar::proc::ConstDataSlotPtr slot,
cedar::aux::ConstDataPtr data
) const
{
if (this->getInputSlot("mInput") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
   return cedar::proc::DataSlot::VALIDITY_VALID;
   }
   else
   {
   // Everything else is rejected.
   return cedar::proc::DataSlot::VALIDITY_ERROR;
   }
}
else if (this->getInputSlot("mErrorSign") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
   // Mat data is accepted.
   return cedar::proc::DataSlot::VALIDITY_VALID;
   }
   else
   {
   // Everything else is rejected.
   return cedar::proc::DataSlot::VALIDITY_ERROR;
   }
}
else if (this->getInputSlot("do learning (c_on)") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
   {
   // Mat data is accepted.
   return cedar::proc::DataSlot::VALIDITY_VALID;
   }
   else
   {
   // Everything else is rejected.
```

```
   return cedar::proc::DataSlot::VALIDITY_ERROR;
   }
}
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void LearningDynamic::parameterChanged()
{
this->_mTau->getValue();
}
void LearningDynamic::inputConnectionChanged(const std::string& inputName)
{
cedar::aux::ConstDataPtr data = this->getInput(inputName);
if (inputName == "mInput")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
this->mInput = boost::shared_dynamic_cast
                <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
if(!this->mInput)
{
return;
}
}
else if (inputName == "mErrorSign")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
this->mErrorSign = boost::shared_dynamic_cast
                <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
if(!this->mErrorSign)
{
return;
}
}
else if (inputName == "do learning (c_on)")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
this->mDoLearning = boost::shared_dynamic_cast
                <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
```

```
if(!this->mDoLearning)
{
return;
}
}
}
void LearningDynamic::eulerStep(const cedar::unit::Time& time)
{
double tau = this->_mTau->getValue();
cv::Mat dot_u;
cv::Mat& output = this->mOutput->getData();
cv::Mat input = this->mInput->getData();
cv::Scalar summe = sum(input);
cv::Mat& sigmoid_u = this->mSigmoidalActivation->getData();
this->mSigmoidalActivation->getData().at<float>(0,0) = summe[0];
sigmoid_u = _mSigmoid->getValue()->compute<float>
(
mSigmoidalActivation->getData()
);
if(sigmoid_u.at<float>(0,0) == 1)
{
cv::Mat test = this->mInput->getData();
this->mTargetTemp->getData() = test.clone();
}
double c_on = this->mDoLearning->getData().at<float>(0,0);
cv::multiply
      (
        this->mErrorSign->getData(),
        this->mTargetTemp->getData(),
        dot_u
      );
  output +=
    (c_on * cedar::unit::Milliseconds(time) / cedar::unit::Milliseconds(tau) * dot_u);
cv::threshold(output, output, 0.0, 0.0, cv::THRESH_TOZERO);
}
void LearningDynamic::reset()
{
}
void LearningDynamic::onStart()
{
this->_mSize->setConstant(true);
}
void LearningDynamic::onStop()
```

```
{
this->_mSize->setConstant(false);
}
```

7.2.17 PeakDetector.h

```
#ifndef PEAK_DETECTOR_H
#define PEAK_DETECTOR_H
// CEDAR INCLUDES
#include <cedar/processing/Step.h>
#include <cedar/auxiliaries/NumericParameter.h>
#include "cedar/auxiliaries/ObjectParameterTemplate.h"
class PeakDetector : public cedar::proc::Step
{
//--------------------------------------------------------------
// macros
//--------------------------------------------------------------
Q_OBJECT
//--------------------------------------------------------------
// nested types
//--------------------------------------------------------------
public:
  //!@brief a parameter for sigmoid objects
  typedef cedar::aux::ObjectParameterTemplate
     <cedar::aux::math::TransferFunction> SigmoidParameter;
  //!@cond SKIPPED_DOCUMENTATION
  CEDAR_GENERATE_POINTER_TYPES_INTRUSIVE(SigmoidParameter);
  //!@endcond
  //--------------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------------
public:
  //!@brief The standard constructor.
  PeakDetector();
  //--------------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------------
public:
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  void onStart();
  void onStop();
  //--------------------------------------------------------------------
```

```
  // protected methods
  //--------------------------------------------------------------------
protected:
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr,
    cedar::aux::ConstDataPtr
  ) const;
  void inputConnectionChanged(const std::string& inputName);
  //--------------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------------
private:
public slots:
  void parameterChanged();
  //--------------------------------------------------------------------
  // members
  //--------------------------------------------------------------------
protected:
  cedar::aux::MatDataPtr mSigmoidalActivation;
private:
  // inputs
  cedar::aux::ConstMatDataPtr mInput;
  // outputs
  cedar::aux::MatDataPtr mOutput;
  //--------------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------------
private:
  SigmoidParameterPtr _mSigmoid;
}; // class utilities::SteeringFilter
#endif // PEAK_DETECTOR_H
```

7.2.18 PeakDetector.cpp

```
#include "PeakDetector.h"
#include <cedar/auxiliaries/MatData.h>
#include <cedar/auxiliaries/DoubleData.h>
#include <cedar/auxiliaries/math/constants.h>
#include <cmath>
#include "cedar/dynamics/fields/NeuralField.h"
#include "cedar/auxiliaries/math/sigmoids/ExpSigmoid.h"
PeakDetector::PeakDetector()
```

```
:
mSigmoidalActivation
(
  new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))
),
mOutput(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
_mSigmoid
(
  new cedar::dyn::NeuralField::SigmoidParameter
  (
    this,
    "sigmoid",
    cedar::aux::math::SigmoidPtr(new cedar::aux::math::ExpSigmoid(1.0, 10000.0))
  )
)
{
this->declareInput("input", false);
    this->declareOutput("output", mOutput);
}
//----------------------------------------------------------------------
// methods
//----------------------------------------------------------------------
cedar::proc::DataSlot::VALIDITY PeakDetector::determineInputValidity
(
cedar::proc::ConstDataSlotPtr slot,
cedar::aux::ConstDataPtr data
) const
{
if (this->getInputSlot("input") == slot)
{
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
   {
   // Mat data is accepted.
   return cedar::proc::DataSlot::VALIDITY_VALID;
   }
   else
   {
   // Everything else is rejected.
   return cedar::proc::DataSlot::VALIDITY_ERROR;
   }
}
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
```

```
void PeakDetector::parameterChanged()
{
}
void PeakDetector::inputConnectionChanged(const std::string& inputName)
{
cedar::aux::ConstDataPtr data = this->getInput(inputName);
if (inputName == "input")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
this->mInput = boost::shared_dynamic_cast
            <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
if(!this->mInput)
{
return;
}
}
}
void PeakDetector::compute(const cedar::proc::Arguments&)
{
cv::Mat input = this->mInput->getData();
cv::Scalar summe = sum(input);
cv::Mat& sigmoid_u = this->mSigmoidalActivation->getData();
this->mSigmoidalActivation->getData().at<float>(0,0) = summe[0];
sigmoid_u = _mSigmoid->getValue()->compute<float>
(
mSigmoidalActivation->getData()
);
this->mOutput->getData().at<float>(0,0) = sigmoid_u.at<float>(0,0);
}
void PeakDetector::onStart()
{
}
void PeakDetector::onStop()
{
}
```

7.2.19 Plugin.h

```
#ifndef BELL_PLUGIN_H
#define BELL_PLUGIN_H
#include <cedar/processing/PluginDeclaration.h>
```

```
CEDAR_DECLARE_PROC_PLUGIN_FUNCTION
(
  void pluginDeclaration
  (
    cedar::proc::PluginDeclarationPtr plugin
  )
);
#endif // BELL_PLUGIN_H
```

7.2.20 Plugin.cpp

```
#include "Plugin.h"
#include "HopfOscillator.h"
#include "BufferToFile.h"
#include "CenterOfVisField.h"
#include "PeakDetector.h"
#include "ColorConversionModified.h"
#include "LearningDynamic.h"
#include <cedar/processing/ElementDeclaration.h>
void pluginDeclaration(cedar::proc::PluginDeclarationPtr plugin)
{
/*---- dynamics.ColorConversionModified --------------------------------------*/
cedar::proc::ElementDeclarationPtr color_conversion_modified_decl
(
new cedar::proc::ElementDeclarationTemplate<ColorConversionModified>
(
"BellSteps",
"ColorConversionModified"
    )
);
color_conversion_modified_decl->setIconPath(":/steps/color_conversion_modified.svg");
color_conversion_modified_decl->setDescription
(
"Color Conversion Modified"
);
plugin->add(color_conversion_modified_decl);
/*---- dynamics.PeakDetector -------------------------------------------------*/
cedar::proc::ElementDeclarationPtr peak_detector_decl
(
new cedar::proc::ElementDeclarationTemplate<PeakDetector>
(
"BellSteps",
"PeakDetector"
```

```
)
);
peak_detector_decl->setIconPath(":/steps/peak_detector.svg");
peak_detector_decl->setDescription
(
"Peak detector"
);
plugin->add(peak_detector_decl);
/*---- dynamics.HopfOscillator ------------------------------------------------*/
cedar::proc::ElementDeclarationPtr hopf_oscillator_decl
(
new cedar::proc::ElementDeclarationTemplate
<HopfOscillator>
(
"BellSteps",
"HopfOscillator"
)
);
hopf_oscillator_decl->setIconPath(":/steps/hopf_oscillator_decl.svg");
hopf_oscillator_decl->setDescription
(
"This is a hopf oscillator."
);
plugin->add(hopf_oscillator_decl);
/*-----------------------------------------------------------------------------*/
/*---- BufferToFile -----------------------------------------------------------*/
cedar::proc::ElementDeclarationPtr buffer_thief_decl
(
new cedar::proc::ElementDeclarationTemplate
<BufferToFile>
     (
     "BellSteps",
     "BufferToFile"
     )
);
buffer_thief_decl->setIconPath(":/steps/buffer_thief_decl.svg");
buffer_thief_decl->setDescription
(
"This buffer thief buffers input and writes it to a file."
);
plugin->add(buffer_thief_decl);
/*-----------------------------------------------------------------------------*/
/*---- CenterOfVisField -------------------------------------------------------*/
```

```
cedar::proc::ElementDeclarationPtr center_of_vis_field_decl
(
new cedar::proc::ElementDeclarationTemplate
            <CenterOfVisField>
            (
             "BellSteps",
             "CenterOfVisField"
            )
);
center_of_vis_field_decl->setIconPath(":/steps/center_of_vis_field.svg");
center_of_vis_field_decl->setDescription
(
"Checks if something is in the fovea centralis."
);
plugin->add(center_of_vis_field_decl);
/*--------------------------------------------------------------------------*/
}
```

7.2.21 RobotSimulator.h

```
//Thanks to Kai Kuchenbecker for providing the template
//at the beginning of the programming of this class
#ifndef SIMULATED_KINEMATIC_CHAIN_STEP_H_
#define SIMULATED_KINEMATIC_CHAIN_STEP_H_
#include <cedar/dynamics/Dynamics.h>
#include "cedar/devices/robot/SimulatedKinematicChain.h"
#include "cedar/auxiliaries/gui/SceneWidget.h"
#include "cedar/devices/robot/gui/KinematicChainWidget.h"
#include "cedar/devices/robot/gui/MountedCameraViewer.h"
#include "cedar/devices/sensors/visual/GrabbableGrabber.h"
class RobotSimulator : public cedar::dyn::Dynamics
{
  //----------------------------------------------------------------
  // macros
  //----------------------------------------------------------------
  Q_OBJECT
public slots:
void parameterChanged();
    void parameter2Changed();
    void parameter3Changed();
  public:
    RobotSimulator();
    virtual ~RobotSimulator();
```

```
  void eulerStep(const cedar::unit::Time& time);
  void inputConnectionChanged(const std::string& inputName);
  void onStart();
  void onStop();
  void startVisualSimulator();
  void reset();
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
   cedar::proc::ConstDataSlotPtr slot,
   cedar::aux::ConstDataPtr data
  ) const;
private:
  cedar::aux::ConstMatDataPtr mInput;
  cedar::aux::ConstMatDataPtr mInput2;
  cedar::aux::ConstMatDataPtr mInput3;
  cedar::aux::MatDataPtr mpReadOutJointVelocities;
  cedar::aux::MatDataPtr mpCameraOutput;
  cedar::aux::MatDataPtr mBufferThiefOutput;
  cedar::aux::MatDataPtr mInputNoise;
  cedar::dev::robot::KinematicChainPtr mpKinematicChain;
  cedar::aux::gui::SceneWidget* mpSceneWidget;
  cedar::dev::robot::gui::KinematicChainWidget* mpWidgetHead;
  cedar::aux::gui::Viewer* mpViewer;
  cedar::dev::robot::gui::MountedCameraViewer* mpCameraViewer;
  cedar::dev::sensors::visual::GrabbableGrabber* mpCameraGrabber;
  cedar::aux::DoubleParameterPtr mpCylinderRedValue;
  cedar::aux::DoubleParameterPtr mpCylinderGreenValue;
  cedar::aux::DoubleParameterPtr mpCylinderBlueValue;
  cedar::aux::DoubleParameterPtr mpCylinderDiameter;
  cedar::aux::DoubleParameterPtr mpCylinderHeight;
  cedar::aux::DoubleParameterPtr mpCylinderPosX;
  cedar::aux::DoubleParameterPtr mpCylinderPosY;
  cedar::aux::DoubleParameterPtr mpCylinderPosZ;
  cedar::aux::DoubleParameterPtr mpCylinder2RedValue;
  cedar::aux::DoubleParameterPtr mpCylinder2GreenValue;
  cedar::aux::DoubleParameterPtr mpCylinder2BlueValue;
  cedar::aux::DoubleParameterPtr mpCylinder2Diameter;
  cedar::aux::DoubleParameterPtr mpCylinder2Height;
  cedar::aux::DoubleParameterPtr mpCylinder2PosX;
  cedar::aux::DoubleParameterPtr mpCylinder2PosY;
  cedar::aux::DoubleParameterPtr mpCylinder2PosZ;
```

```
    cedar::aux::DoubleParameterPtr mpCylinder3RedValue;
    cedar::aux::DoubleParameterPtr mpCylinder3GreenValue;
    cedar::aux::DoubleParameterPtr mpCylinder3BlueValue;
    cedar::aux::DoubleParameterPtr mpCylinder3Diameter;
    cedar::aux::DoubleParameterPtr mpCylinder3Height;
    cedar::aux::DoubleParameterPtr mpCylinder3PosX;
    cedar::aux::DoubleParameterPtr mpCylinder3PosY;
    cedar::aux::DoubleParameterPtr mpCylinder3PosZ;
    cedar::aux::BoolParameterPtr mpLockHeadPos;
    cedar::aux::gl::ScenePtr mpScene;
    int mCylinderSceneIndex;
    int mCylinder2SceneIndex;
    int mCylinder3SceneIndex;
  protected:
    cedar::aux::DoubleParameterPtr _mInputNoiseGain;
};
#endif /* SIMULATED_KINEMATIC_CHAIN_STEP_H_ */
```

7.2.22 RobotSimulator.cpp

```
//Thanks to Kai Kuchenbecker for providing the template
//at the beginning of the programming of this class
#include "RobotSimulator.h"
#include <cedar/auxiliaries/MatData.h>
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
#include "cedar/auxiliaries/systemFunctions.h"
#include "cedar/devices/robot/gl/CoraHead.h"
#include "cedar/auxiliaries/sleepFunctions.h"
#include "cedar/auxiliaries/gl/Cylinder.h"
#include "cedar/auxiliaries/annotation/ColorSpace.h"
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr simulated_kinematic_chain_decl
    (
      new ElementDeclarationTemplate<RobotSimulator>
      (
        "BellSteps",
        "cedar.processing.RobotSimulator"
```

```
      )
    );
    simulated_kinematic_chain_decl->setIconPath(":/steps/static_gain.svg");
    simulated_kinematic_chain_decl->setDescription
    (
      "Simulation of head"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass
      (
        simulated_kinematic_chain_decl
      );
    return true;
  }
  bool declared = declare();
}
RobotSimulator::RobotSimulator() :
  mInput(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_64F))),
  mInput2(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_64F))),
  mInput3(new cedar::aux::MatData(cv::Mat::zeros(2, 1, CV_64F))),
  mpCameraOutput(new cedar::aux::MatData(cv::Mat::zeros(480, 640, CV_8UC3))),
  mBufferThiefOutput(new cedar::aux::MatData(cv::Mat(2,1, CV_32F))),
  mInputNoise(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
  mpSceneWidget(NULL),
  mpWidgetHead(NULL),
  mpViewer(NULL),
  mpCameraViewer(NULL),
  mpCameraGrabber(NULL),
  mpCylinderRedValue
  (
    new cedar::aux::DoubleParameter
    (
      this,
      "Cylinder Red Value",
      1,
      0,
      1
    )
  ),
  mpCylinderGreenValue
  (
    new cedar::aux::DoubleParameter
    (
```

```
    this,
    "Cylinder Green Value",
    1,
    0,
    1
  )
),
mpCylinderBlueValue
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder Blue Value",
    0,
    0,
    1
  )
),
mpCylinderDiameter
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder Diameter",
    0.05,
    0.01,
    1
  )
),
mpCylinderHeight
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder Height",
    0.02,
    0.01,
    1
  )
),
mpCylinderPosX
(
  new cedar::aux::DoubleParameter
```

```
    (
      this,
      "Cylinder X Position",
      0.5,
      -2,
      2
    )
  ),
  mpCylinderPosY
  (
    new cedar::aux::DoubleParameter
    (
      this,
      "Cylinder Y Position",
      1.5,
      -2,
      2
    )
  ),
  mpCylinderPosZ
  (
    new cedar::aux::DoubleParameter
    (
      this,
      "Cylinder Z Position",
      0.5,
      -2,
      2
    )
  ),
  mpCylinder2RedValue
  (
    new cedar::aux::DoubleParameter
    (
      this,
      "Cylinder2 Red Value",
      1,
      0,
      1
    )
  ),
  mpCylinder2GreenValue
  (
```

```
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Green Value",
    1,
    0,
    1
  )
),
mpCylinder2BlueValue
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Blue Value",
    0,
    0,
    1
  )
),
mpCylinder2Diameter
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Diameter",
    0.05,
    0.01,
    1
  )
),
mpCylinder2Height
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Height",
    0.02,
    0.01,
    1
  )
),
mpCylinder2PosX
```

```
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 X Position",
    -0.5,
    -2,
    2
  )
),
mpCylinder2PosY
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Y Position",
    1.5,
    -2,
    2
  )
),
mpCylinder2PosZ
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder2 Z Position",
    0.5,
    -2,
    2
  )
),
mpCylinder3RedValue
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Red Value",
    1,
    0,
    1
  )
),
```

```
mpCylinder3GreenValue
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Green Value",
    1,
    0,
    1
  )
),
mpCylinder3BlueValue
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Blue Value",
    0,
    0,
    1
  )
),
mpCylinder3Diameter
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Diameter",
    0.05,
    0.01,
    1
  )
),
mpCylinder3Height
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Height",
    0.02,
    0.01,
    1
  )
```

```
),
mpCylinder3PosX
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 X Position",
    0.8,
    -2,
    2
  )
),
mpCylinder3PosY
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Y Position",
    1.5,
    -2,
    2
  )
),
mpCylinder3PosZ
(
  new cedar::aux::DoubleParameter
  (
    this,
    "Cylinder3 Z Position",
    0.5,
    -2,
    2
  )
),
mpLockHeadPos
(
  new cedar::aux::BoolParameter
  (
    this,
    "Lock Head Position",
    false
  )
),
```

```
  mCylinderSceneIndex(-1),
  mCylinder2SceneIndex(-1),
  mCylinder3SceneIndex(-1),
  _mInputNoiseGain
  (
    new cedar::aux::DoubleParameter
    (
      this,
  "input noise gain",
  0.01,
  cedar::aux::DoubleParameter::LimitType::positiveZero()
)
  )
{
  cedar::dev::robot::KinematicChainPtr
       sim(new cedar::dev::robot::SimulatedKinematicChain());
  sim->setWorkingMode( cedar::dev::robot::KinematicChain::VELOCITY );
  std::string configFile = cedar::aux::locateResource("configs/schunk_head.json");
  sim->readJson( configFile );
  sim->start();
  mpKinematicChain = sim;
  QObject::connect
  (
    this->mpCylinderRedValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(parameterChanged())
  );
  QObject::connect
  (
    this->mpCylinderGreenValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameterChanged() )
  );
  QObject::connect
  (
    this->mpCylinderBlueValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameterChanged() )
  );
  QObject::connect
```

```
(
  this->mpCylinderDiameter.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameterChanged() )
);
QObject::connect
(
  this->mpCylinderHeight.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameterChanged() )
);
QObject::connect
(
  this->mpCylinderPosX.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameterChanged() )
);
QObject::connect
(
  this->mpCylinderPosY.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameterChanged() )
);
QObject::connect
(
  this->mpCylinderPosZ.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameterChanged() )
);
QObject::connect
(
  this->mpCylinder2RedValue.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
```

```
  this->mpCylinder2GreenValue.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2BlueValue.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2Diameter.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2Height.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2PosX.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2PosY.get(),
  SIGNAL(valueChanged()),
  this,
  SLOT( parameter2Changed() )
);
QObject::connect
(
  this->mpCylinder2PosZ.get(),
```

```
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter2Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3RedValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3GreenValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3BlueValue.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3Diameter.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3Height.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3PosX.get(),
    SIGNAL(valueChanged()),
```

```
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3PosY.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpCylinder3PosZ.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameter3Changed() )
  );
  QObject::connect
  (
    this->mpLockHeadPos.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT( parameterChanged() )
  );
  this->registerFunction
  (
    "startVisualSimulator",
    boost::bind(&RobotSimulator::startVisualSimulator, this)
  );
  this->declareInput("input");
  this->declareInput("input2");
  this->declareInput("input3");
  this->declareOutput("camera_output", mpCameraOutput );
  this->declareOutput("data for matlab", mBufferThiefOutput);
}
RobotSimulator::~RobotSimulator()
{
  mpKinematicChain->stop();
  cedar::aux::sleep(cedar::unit::Seconds(1));
}
cedar::proc::DataSlot::VALIDITY RobotSimulator::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr CEDAR_DEBUG_ONLY(slot),
```

```
  cedar::aux::ConstDataPtr data
) const
{
// First, let's make sure that this is really
    // the input in case anyone ever changes our interface.
CEDAR_DEBUG_ASSERT(slot->getName() == "input"
        || slot->getName() == "input2" || slot->getName() == "input3");
if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
// Mat data is accepted.
return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
// Everything else is rejected.
return cedar::proc::DataSlot::VALIDITY_ERROR;
}
}
void RobotSimulator::inputConnectionChanged(const std::string& inputName)
{
// Again, let's first make sure that this is really
    // the input in case anyone ever changes our interface.
CEDAR_DEBUG_ASSERT(inputName == "input"
        || inputName == "input2" || inputName == "input3");
if (inputName == "input")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
    this->mInput = boost::shared_dynamic_cast
            <const cedar::aux::MatData>(this->getInput(inputName));
    // This should always work since other types should not be accepted.
    if(!this->mInput)
    {
      return;
    }
}
else if (inputName == "input2")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
    this->mInput2 = boost::shared_dynamic_cast
            <const cedar::aux::MatData>(this->getInput(inputName));
}
```

```
else if (inputName == "input3")
{
// Assign the input to the member.
        // This saves us from casting in every computation step.
this->mInput3 = boost::shared_dynamic_cast
            <const cedar::aux::MatData>(this->getInput(inputName));
}
}
void RobotSimulator::parameterChanged()
{
  if( mpScene )
  {
    cedar::aux::gl::CylinderPtr cylinder =
         boost::dynamic_pointer_cast
              <cedar::aux::gl::Cylinder>
                (mpScene->getObjectVisualization(mCylinderSceneIndex));
    cylinder->setColor
      (
        mpCylinderRedValue->getValue(),
        mpCylinderGreenValue->getValue(),
        mpCylinderBlueValue->getValue()
      );
    cylinder->setRadius(mpCylinderDiameter->getValue());
    cylinder->setHeight(mpCylinderHeight->getValue());
    cylinder->getLocalCoordinateFrame()->setTranslation
    (
      mpCylinderPosX->getValue(),
      mpCylinderPosY->getValue(),
      mpCylinderPosZ->getValue()
    );
  }
}
void RobotSimulator::parameter2Changed()
{
  if( mpScene )
  {
    cedar::aux::gl::CylinderPtr cylinder =
         boost::dynamic_pointer_cast
              <cedar::aux::gl::Cylinder>
                (mpScene->getObjectVisualization(mCylinder2SceneIndex));
    cylinder->setColor
    (
      mpCylinder2RedValue->getValue(),
```

```
      mpCylinder2GreenValue->getValue(),
      mpCylinder2BlueValue->getValue()
    );
    cylinder->setRadius(mpCylinder2Diameter->getValue());
    cylinder->setHeight(mpCylinder2Height->getValue());
    cylinder->getLocalCoordinateFrame()->setTranslation
    (
      mpCylinder2PosX->getValue(),
      mpCylinder2PosY->getValue(),
      mpCylinder2PosZ->getValue()
    );
  }
}
void RobotSimulator::parameter3Changed()
{
  if( mpScene )
  {
    cedar::aux::gl::CylinderPtr cylinder =
        boost::dynamic_pointer_cast
          <cedar::aux::gl::Cylinder>
            (mpScene->getObjectVisualization(mCylinder3SceneIndex));
    cylinder->setColor
    (
      mpCylinder3RedValue->getValue(),
      mpCylinder3GreenValue->getValue(),
      mpCylinder3BlueValue->getValue()
    );
    cylinder->setRadius(mpCylinder3Diameter->getValue());
    cylinder->setHeight(mpCylinder3Height->getValue());
    cylinder->getLocalCoordinateFrame()->setTranslation
    (
      mpCylinder3PosX->getValue(),
      mpCylinder3PosY->getValue(),
      mpCylinder3PosZ->getValue()
    );
  }
}
void RobotSimulator::startVisualSimulator()
{
  // create gl visualization objects
  cedar::dev::robot::gl::KinematicChainPtr p_cora_head_visualization
  (
    new cedar::dev::robot::gl::CoraHead(mpKinematicChain)
```

```
);
p_cora_head_visualization->setDisplayEndEffectorVelocity(false);
// create scene and viewer to display the arm
mpScene = cedar::aux::gl::ScenePtr(new cedar::aux::gl::Scene() );
mpScene->setSceneLimit(2); //1?
mpScene->drawFloor(true);
mpViewer = new cedar::aux::gui::Viewer(mpScene);
mpViewer->show();
mpViewer->setSceneRadius(mpScene->getSceneLimit());
cv::Mat starting_pos = cv::Mat::zeros(2,1,CV_64F);
mpKinematicChain->setJointAngles(starting_pos);
// create visualization objects for the head and arm
cedar::aux::gl::ObjectVisualizationPtr p_object;
p_object = p_cora_head_visualization;
mpScene->addObjectVisualization(p_object);
// create a cylinder visualization and add it to the scene
cedar::aux::LocalCoordinateFramePtr cylinder_local_coordinate_frame
(
  new cedar::aux::LocalCoordinateFrame()
);
cylinder_local_coordinate_frame->setName("cylinder");
cylinder_local_coordinate_frame->setTranslation
(
  mpCylinderPosX->getValue(),
  mpCylinderPosY->getValue(),
  mpCylinderPosZ->getValue()
);
cedar::aux::gl::ObjectVisualizationPtr cylinder
(
  new cedar::aux::gl::Cylinder
  (
    cylinder_local_coordinate_frame,
    mpCylinderDiameter->getValue(),
    mpCylinderHeight->getValue(),
    mpCylinderRedValue->getValue(),
    mpCylinderGreenValue->getValue(),
    mpCylinderBlueValue->getValue()
  )
);
mCylinderSceneIndex = mpScene->addObjectVisualization(cylinder);
// create a second cylinder visualization and add it to the scene
cedar::aux::LocalCoordinateFramePtr cylinder2_local_coordinate_frame
(
```

```
  new cedar::aux::LocalCoordinateFrame()
);
cylinder2_local_coordinate_frame->setName("cylinder2");
cylinder2_local_coordinate_frame->setTranslation
(
  mpCylinder2PosX->getValue(),
  mpCylinder2PosY->getValue(),
  mpCylinder2PosZ->getValue()
);
cedar::aux::gl::ObjectVisualizationPtr cylinder2
(
  new cedar::aux::gl::Cylinder
  (
    cylinder2_local_coordinate_frame,
    mpCylinder2Diameter->getValue(),
    mpCylinder2Height->getValue(),
    mpCylinder2RedValue->getValue(),
    mpCylinder2GreenValue->getValue(),
    mpCylinder2BlueValue->getValue()
  )
);
mCylinder2SceneIndex = mpScene->addObjectVisualization(cylinder2);
// create a third cylinder visualization and add it to the scene
cedar::aux::LocalCoordinateFramePtr cylinder3_local_coordinate_frame
(
  new cedar::aux::LocalCoordinateFrame()
);
cylinder3_local_coordinate_frame->setName("cylinder3");
cylinder3_local_coordinate_frame->setTranslation
(
  mpCylinder3PosX->getValue(),
  mpCylinder3PosY->getValue(),
  mpCylinder3PosZ->getValue()
);
cedar::aux::gl::ObjectVisualizationPtr cylinder3
(
  new cedar::aux::gl::Cylinder
  (
    cylinder3_local_coordinate_frame,
    mpCylinder3Diameter->getValue(),
    mpCylinder3Height->getValue(),
    mpCylinder3RedValue->getValue(),
    mpCylinder3GreenValue->getValue(),
```

```
    mpCylinder3BlueValue->getValue()
  )
);
mCylinder3SceneIndex = mpScene->addObjectVisualization(cylinder3);
// create control widgets for the scene and the arm
mpSceneWidget = new cedar::aux::gui::SceneWidget(mpScene);
mpWidgetHead =
  new cedar::dev::robot::gui::KinematicChainWidget(mpKinematicChain);
// create a mounted camera viewer
mpCameraViewer =
  new cedar::dev::robot::gui::MountedCameraViewer(mpScene, mpKinematicChain);
std::string cameraConfigFile =
  cedar::aux::locateResource( "configs/schunk_camera.json");
mpCameraViewer->readJson(cameraConfigFile);
mpCameraGrabber =
  new cedar::dev::sensors::visual::GrabbableGrabber
  (
    mpCameraViewer,
    "mounted_camera_view_grabber"
  );
mpWidgetHead->show();
mpCameraViewer->show();
mpKinematicChain->startTimer(50.0);
mpViewer->startTimer(50);
mpCameraViewer->startTimer(20);
if (! mpCameraGrabber->applyParameter())
{
}
// allow to register the grabber in the viewer class and wait.
// at least one redraw is needed to write the GL-Image in the
// grab-buffer from interface "Grabbable"
for (int i = 0; i < 100; ++i)
{
while (QApplication::hasPendingEvents())
{
        QApplication::processEvents();
}
cedar::aux::sleep(cedar::unit::Milliseconds(10));
}
if (mpCameraGrabber->isCreated())
{
    mpCameraGrabber->grab();
    QReadWriteLock* p_lock = mpCameraGrabber->getReadWriteLockPointer();
```

```
        // the local image buffer
        cv::Mat frame;
        // get the picture from the grabber
        p_lock->lockForRead();
        frame = mpCameraGrabber->getImage().clone();
        p_lock->unlock();
        mpCameraOutput->setData(frame);
        cedar::aux::annotation::ColorSpacePtr color_space;
        switch (this->mpCameraOutput->getData().channels())
        {
            case 4:
             color_space = cedar::aux::annotation::ColorSpace::bgra();
             break;
            case 3:
             color_space = cedar::aux::annotation::ColorSpace::bgr();
             break;
            case 1:
             color_space = cedar::aux::annotation::ColorSpace::gray();
             break;
            default:
             // this should not happen.
             CEDAR_ASSERT(false);
        } // switch
        this->mpCameraOutput->setAnnotation(color_space);
    }
}
void RobotSimulator::reset()
{
    cv::Mat starting_pos = cv::Mat::zeros(2,1,CV_64F);
    mpKinematicChain->setJointAngles(starting_pos);
    if( mpViewer != NULL )
    {
      mpViewer->show();
    }
    if(mpWidgetHead != NULL)
    {
      mpWidgetHead->show();
    }
    if(mpCameraViewer != NULL)
    {
      mpCameraViewer->show();
    }
}
```

```
void RobotSimulator::eulerStep(const cedar::unit::Time& time)
{
  const cv::Mat& input = this->mInput->getData();
  const cv::Mat& input2 = this->mInput2->getData();
  const cv::Mat& input3 = this->mInput3->getData();
  cv::Mat velocityVector = cv::Mat::zeros(2, 1, CV_64F);
  cv::Mat& input_noise = this->mInputNoise->getData();
  cv::randn(input_noise, cv::Scalar(0), cv::Scalar(1));
  double noise =
    1.0 / sqrt(cedar::unit::Milliseconds(time)/cedar::unit::Milliseconds(1.0))
        *_mInputNoiseGain->getValue() * input_noise.at<float>(0,0);
  if(input.at<float>(0,0) == 0 && input2.at<float>(0,0) == 0
      && input3.at<float>(0,0) == 0 && input3.at<float>(1,0) == 0)
  {
noise = 0;
  }
  velocityVector.at<double>(0,0) =
    noise + input.at<float>(0,0) + input3.at<float>(0,0);
  velocityVector.at<double>(1,0) =
    noise + -1. * (input2.at<float>(0,0) + input3.at<float>(1,0));
  //set the joint velocities, if movable
  if (mpKinematicChain->isMovable() )
  {
    mpKinematicChain->setJointVelocities( velocityVector );
  }
  this->mBufferThiefOutput->getData().at<float>(0,0) =
        mpKinematicChain->getJointAngle(0);
  this->mBufferThiefOutput->getData().at<float>(1,0) =
        mpKinematicChain->getJointAngle(1);
  if (mpCameraGrabber->isCreated())
  {
    mpCameraGrabber->grab();
    QReadWriteLock* p_lock = mpCameraGrabber->getReadWriteLockPointer();
    // get the picture from the grabber
    p_lock->lockForRead();
    this->mpCameraOutput->setData(this->mpCameraGrabber->getImage().clone());
    p_lock->unlock();
  }
}
void RobotSimulator::onStart()
{
}
void RobotSimulator::onStop()
```

```
{
if (mpKinematicChain->isMovable() )
  {
cv::Mat test = cv::Mat::zeros(2, 1, CV_64F);
test.at<double>(0,0) =  0;
test.at<double>(1,0) = 0;
    mpKinematicChain->setJointVelocities( test );
  }
}
```

7.2.23 SelectGain.h

```
#ifndef DYNAMICS_HOPF_OSCILLATOR_H
#define DYNAMICS_HOPF_OSCILLATOR_H
#include "cedar/processing/Step.h"
#include <cedar/auxiliaries/NumericParameter.h>
#include "cedar/auxiliaries/ObjectParameterTemplate.h"
class SelectGain : public cedar::proc::Step
{
  //--------------------------------------------------------------
  // macros
  //--------------------------------------------------------------
  Q_OBJECT
  //--------------------------------------------------------------
  // nested types
  //--------------------------------------------------------------
public:
  //!@brief a parameter for sigmoid objects
  typedef cedar::aux::ObjectParameterTemplate
      <cedar::aux::math::TransferFunction> SigmoidParameter;
  //!@cond SKIPPED_DOCUMENTATION
  CEDAR_GENERATE_POINTER_TYPES_INTRUSIVE(SigmoidParameter);
  //!@endcond
  //--------------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------------
public:
  //!@brief The standard constructor.
  SelectGain();
  //--------------------------------------------------------------
  // public methods
  //--------------------------------------------------------------
public:
```

```
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  void onStart();
  void onStop();
  //----------------------------------------------------------------
  // protected methods
  //----------------------------------------------------------------
protected:
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr,
    cedar::aux::ConstDataPtr
  ) const;
  void inputConnectionChanged(const std::string& inputName);
  //----------------------------------------------------------------
  // private methods
  //----------------------------------------------------------------
private:
  //!@brief Resets the field.
  void reset();
public slots:
  void parameterChanged();
  //----------------------------------------------------------------
  // members
  //----------------------------------------------------------------
protected:
  cedar::aux::MatDataPtr mSigmoidalActivation;
private:
  // inputs
  cedar::aux::ConstMatDataPtr mTarget;
  cedar::aux::ConstMatDataPtr mLearnedGain;
  //outputs
  cedar::aux::MatDataPtr mOutput;
  //----------------------------------------------------------------
  // parameters
  //----------------------------------------------------------------
private:
  cedar::aux::DoubleParameterPtr _mTau;
  SigmoidParameterPtr _mSigmoid;
}; // class utilities::SteeringFilter
#endif // DYNAMICS_HOPF_OSCILLATOR_H
```

7.2.24 SelectGain.cpp

```
#include "SelectGain.h"
#include <cedar/auxiliaries/MatData.h>
#include <cedar/auxiliaries/DoubleData.h>
#include <cedar/auxiliaries/math/constants.h>
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
#include <cmath>
#include "cedar/dynamics/fields/NeuralField.h"
#include "cedar/auxiliaries/math/sigmoids/ExpSigmoid.h"
//----------------------------------------------------------------------
// register the class
//----------------------------------------------------------------------
namespace
{
bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr SelectGain_decl
    (
      new ElementDeclarationTemplate<SelectGain>
      (
        "BellSteps",
        "SelectGain"
      )
    );
    SelectGain_decl->setIconPath(":/steps/static_gain.svg");
    SelectGain_decl->setDescription
    (
      "gain * target"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass(SelectGain_decl);
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
SelectGain::SelectGain()
:
```

```
  mSigmoidalActivation
  (
    new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))
  ),
  mTarget(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mLearnedGain(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  mOutput(new cedar::aux::MatData(cv::Mat(1,1, CV_32F))),
  _mTau
  (
    new cedar::aux::DoubleParameter
    (
      this,
      "tau",
      1.0,
      0.001,
      10000.0
    )
  ),
  _mSigmoid
  (
    new cedar::dyn::NeuralField::SigmoidParameter
(
  this,
  "sigmoid",
  cedar::aux::math::SigmoidPtr
      (
        new cedar::aux::math::ExpSigmoid(1.0, 10000.0)
      )
)
  )
{
  // inputs
  this->declareInput("mTarget", false);
  this->declareInput("mLearnedGain", false);
  //outputs
  this->declareOutput("output", mOutput);
  // connect the parameter's change signal
  QObject::connect
  (
    _mTau.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(parameterChanged())
```

```
  );
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
cedar::proc::DataSlot::VALIDITY SelectGain::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr slot,
  cedar::aux::ConstDataPtr data
) const
{
  if (this->getInputSlot("mTarget") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
  }
  else if (this->getInputSlot("mLearnedGain") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
  }
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void SelectGain::parameterChanged()
{
this->_mTau->getValue();
}
```

```
void SelectGain::inputConnectionChanged(const std::string& inputName)
{
  cedar::aux::ConstDataPtr data = this->getInput(inputName);
  if (inputName == "mTarget")
  {
// Assign the input to the member.
    // This saves us from casting in every computation step.
this->mTarget = boost::shared_dynamic_cast
          <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
if(!this->mTarget)
{
  return;
}
  }
  else if (inputName == "mLearnedGain")
  {
// Assign the input to the member.
    // This saves us from casting in every computation step.
this->mLearnedGain = boost::shared_dynamic_cast
          <const cedar::aux::MatData>(this->getInput(inputName));
// This should always work since other types should not be accepted.
if(!this->mLearnedGain)
{
  return;
}
  }
}
void SelectGain::compute(const cedar::proc::Arguments&)
{
  cv::Mat input = this->mTarget->getData();
  cv::Scalar summe = sum(input);
  cv::Mat& sigmoid_u = this->mSigmoidalActivation->getData();
  this->mSigmoidalActivation->getData().at<float>(0,0) = summe[0];
  sigmoid_u = _mSigmoid->getValue()->compute<float>
  (
    mSigmoidalActivation->getData()
  );
  if(sigmoid_u.at<float>(0,0) == 1)
  {
    cv::Point maxLoc;
minMaxLoc(this->mTarget->getData(), NULL, NULL, NULL, &maxLoc);
this->mOutput->getData().at<float>(0,0) =
```

```
          this->mLearnedGain->getData().at<float>(maxLoc.x, maxLoc.y);
  }
  else
  {
    this->mOutput->getData().at<float>(0,0) = 1.0;
  }
}
void SelectGain::reset()
{
}
void SelectGain::onStart()
{
}
void SelectGain::onStop()
{
}
```

7.2.25 ThresholdingHue.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
#include "cedar/auxiliaries/DoubleParameter.h"
class ThresholdingHue : public cedar::proc::Step
{
//--------------------------------------------------------------
// macros
//--------------------------------------------------------------
Q_OBJECT
  //------------------------------------------------------------
  // constructors and destructor
  //------------------------------------------------------------
public:
  //!@brief The standard constructor.
  ThresholdingHue();
  //------------------------------------------------------------
  // public methods
  //------------------------------------------------------------
public:
  //!@brief Returns the current hue lower limit.
  inline double getHueLowerLimit() const
```

```
  {
    return this->_mHueLowerLimit->getValue();
  }
  //!@brief Sets the current hue lower limit.
  inline void setHueLowerLimit(double hueLowerLimit)
  {
    this->_mHueLowerLimit->setValue(hueLowerLimit);
  }
  //!@brief Returns the current hue upper limit.
  inline double getHueUpperLimit() const
  {
    return this->_mHueUpperLimit->getValue();
  }
  //!@brief Sets the current hue upper limit.
  inline void setHueUpperLimit(double hueUpperLimit)
  {
    this->_mHueUpperLimit->setValue(hueUpperLimit);
  }
public slots:
  //!@brief This slot is connected to the valueChanged()
  //event of the gain value parameter.
  void hueLimitChanged();
  //--------------------------------------------------------------
  // protected methods
  //--------------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr slot,
    cedar::aux::ConstDataPtr data
  ) const;
  //--------------------------------------------------------------
  // private methods
  //--------------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  //--------------------------------------------------------------
  // members
  //--------------------------------------------------------------
```

```
protected:
  //!@brief MatrixData representing the input.
  //Storing it like this saves time during computation.
  cedar::aux::ConstMatDataPtr mInput;
  //!@brief The data containing the output.
  cedar::aux::MatDataPtr mOutput;
private:
  //--------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------
protected:
  //!@brief Hue lower limit
  cedar::aux::UIntParameterPtr _mHueLowerLimit;
  //!@brief Hue upper limit
  cedar::aux::UIntParameterPtr _mHueUpperLimit;
private:
}; // class cedar::proc::steps::ThresholdingHue
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.26 ThresholdingHue.cpp

```
// CEDAR INCLUDES
#include "ThresholdingHue.h"
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
//----------------------------------------------------------------------
// register the class
//----------------------------------------------------------------------
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr static_gain_decl
    (
      new ElementDeclarationTemplate<ThresholdingHue>
      (
        "BellSteps",
        "cedar.processing.ThresholdingHue"
      )
    );
    static_gain_decl->setIconPath(":/steps/static_gain.svg");
```

```
    static_gain_decl->setDescription
    (
      "Thresholds the Hue. Can be also used for other channels between 0 und 255."
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass(static_gain_decl);
    return true;
  }
  bool declared = declare();
}
//----------------------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------------------
ThresholdingHue::ThresholdingHue()
:
  // outputs
  mOutput(new cedar::aux::MatData(cv::Mat())),
  // parameters
  _mHueLowerLimit
  (
    new cedar::aux::UIntParameter
(
  this,
  "Hue lower limit",
  0,
  cedar::aux::UIntParameter::LimitType::positiveZero(255)
)
  ),
  _mHueUpperLimit
  (
    new cedar::aux::UIntParameter
(
  this,
  "Hue upper limit",
  255,
  cedar::aux::UIntParameter::LimitType::positiveZero(255)
)
  )
{
  // declare all data
  this->declareInput("input");
  this->declareOutput("output", mOutput);
  // connect the parameter's change signal
```

```
  QObject::connect
  (
    _mHueLowerLimit.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(hueLimitChanged())
  );
  QObject::connect
  (
    _mHueUpperLimit.get(),
    SIGNAL(valueChanged()),
    this,
    SLOT(hueLimitChanged())
  );
}
//----------------------------------------------------------------
// methods
//----------------------------------------------------------------
void ThresholdingHue::compute(const cedar::proc::Arguments&)
{
  cv::Mat output = this->mOutput->getData();
  cv::Mat input = this->mInput->getData();
  cv::Scalar lowerLimit =
        cvScalarAll((int)this->_mHueLowerLimit->getValue());
  cv::Scalar upperLimit =
        cvScalarAll((int)this->_mHueUpperLimit->getValue());
  cv::inRange(input, lowerLimit, upperLimit, output);
}
void ThresholdingHue::hueLimitChanged()
{
  // when the gain changes, the output needs to be recalculated.
  this->onTrigger();
}
cedar::proc::DataSlot::VALIDITY ThresholdingHue::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr CEDAR_DEBUG_ONLY(slot),
  cedar::aux::ConstDataPtr data
) const
{
  // First, let's make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(slot->getName() == "input")
  if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
```

```
  {
    // Mat data is accepted.
    return cedar::proc::DataSlot::VALIDITY_VALID;
  }
  else
  {
    // Everything else is rejected.
    return cedar::proc::DataSlot::VALIDITY_ERROR;
  }
}
void ThresholdingHue::inputConnectionChanged(const std::string& inputName)
{
  // Again, let's first make sure that this is really
  // the input in case anyone ever changes our interface.
  CEDAR_DEBUG_ASSERT(inputName == "input");
  // Assign the input to the member.
  // This saves us from casting in every computation step.
  this->mInput = boost::shared_dynamic_cast
        <const cedar::aux::MatData>(this->getInput(inputName));
  if(!this->mInput)
  {
    return;
  }
  // Let's get a reference to the input matrix.
  const cv::Mat& input = this->mInput->getData();
  // Make a copy to create a matrix of the same type, dimensions, ...
  this->mOutput->setData(input.clone());
  this->mOutput->copyAnnotationsFrom(this->mInput);
}
```

7.2.27 TooFarOrTooClose.h

```
#ifndef CEDAR_PROC_STEPS_STATIC_GAIN_H
#define CEDAR_PROC_STEPS_STATIC_GAIN_H
// CEDAR INCLUDES
#include "cedar/processing/Step.h"
#include "cedar/auxiliaries/MatData.h"
/*!@brief
 *
 * @remarks
 */
class TooFarOrTooClose : public cedar::proc::Step
{
```

```
  //--------------------------------------------------
  // macros
  //--------------------------------------------------
  Q_OBJECT
  //--------------------------------------------------------
  // constructors and destructor
  //--------------------------------------------------------
public:
  //!@brief The standard constructor.
  TooFarOrTooClose();
  //--------------------------------------------------------
  // public methods
  //--------------------------------------------------------
public:
  //--------------------------------------------------------
  // protected methods
  //--------------------------------------------------------
protected:
  //!@brief Determines whether the data item can be connected to the slot.
  cedar::proc::DataSlot::VALIDITY determineInputValidity
  (
    cedar::proc::ConstDataSlotPtr slot,
    cedar::aux::ConstDataPtr data
  ) const;
  //--------------------------------------------------------
  // private methods
  //--------------------------------------------------------
private:
  //!@brief Reacts to a change in the input connection.
  void inputConnectionChanged(const std::string& inputName);
  //!@brief Updates the output matrix.
  void compute(const cedar::proc::Arguments& arguments);
  //--------------------------------------------------------
  // members
  //--------------------------------------------------------
protected:
  cedar::aux::ConstMatDataPtr mPE1;
  cedar::aux::ConstMatDataPtr mTarget1;
  cedar::aux::ConstMatDataPtr mCOn;
  cedar::aux::ConstMatDataPtr mCOnPE;
  cedar::aux::MatDataPtr mTarget1Temp;
  cedar::aux::MatDataPtr mOriginReceived;
  cedar::aux::MatDataPtr mOutput;
```

```
private:
  //--------------------------------------------------------------
  // parameters
  //--------------------------------------------------------------
protected:
private:
}; // class cedar::proc::steps::TooFarOrTooClose
#endif // CEDAR_PROC_STEPS_STATIC_GAIN_H
```

7.2.28 TooFarOrTooClose.cpp

```
// CEDAR INCLUDES
#include "TooFarOrTooClose.h"
#include "cedar/processing/ElementDeclaration.h"
#include "cedar/processing/DeclarationRegistry.h"
//--------------------------------------------------------------
// register the class
//--------------------------------------------------------------
namespace
{
  bool declare()
  {
    using cedar::proc::ElementDeclarationPtr;
    using cedar::proc::ElementDeclarationTemplate;
    ElementDeclarationPtr TooFarOrTooClose_decl
    (
      new ElementDeclarationTemplate<TooFarOrTooClose>
      (
        "BellSteps",
        "TooFarOrTooClose"
      )
    );
    TooFarOrTooClose_decl->setIconPath(":/steps/to_far_or_to_close.svg");
    TooFarOrTooClose_decl->setDescription
    (
      "Was the saccade to far or to close"
    );
    cedar::aux::Singleton
      <cedar::proc::DeclarationRegistry>::getInstance()->declareClass
      (
        TooFarOrTooClose_decl
      );
    return true;
```

```
  }
  bool declared = declare();
}
//----------------------------------------------------------
// constructors and destructor
//----------------------------------------------------------
TooFarOrTooClose::TooFarOrTooClose()
:
  mPE1(new cedar::aux::MatData(cv::Mat::zeros(1,1, CV_32F))),
  mTarget1(new cedar::aux::MatData(cv::Mat())),
  mCOn(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
  mCOnPE(new cedar::aux::MatData(cv::Mat(1, 1, CV_32F))),
  mTarget1Temp(new cedar::aux::MatData(cv::Mat())),
  mOriginReceived(new cedar::aux::MatData(cv::Mat::zeros(1, 1, CV_32F))),
  mOutput(new cedar::aux::MatData(cv::Mat(1,1, CV_32F)))
{
  this->declareInput("perceptual error 1");
  this->declareInput("target 1");
  this->declareInput("c_on");
  this->declareInput("c_on perceptual error");
  this->declareOutput("output", mOutput);
}
//----------------------------------------------------------
// methods
//----------------------------------------------------------
void TooFarOrTooClose::compute(const cedar::proc::Arguments&)
{
double c_on = this->mCOn->getData().at<float>(0,0);
double c_onPE = this->mCOnPE->getData().at<float>(0,0);
if(fabs(this->mPE1->getData().at<float>(0,0)) <= 0.001)
{
//do nothing
}
else
{
if(c_on == 1.0 && c_onPE == 0.0)
{
cv::Mat test = this->mTarget1->getData();
this->mTarget1Temp->getData() = test.clone();
this->mOutput->getData().at<float>(0,0) = 0.0;
}
else if (c_on == 0.0 && c_onPE == 1.0)
{
```

```
cv::multiply
            (
              this->mPE1->getData().at<float>(0,0),
              this->mTarget1Temp->getData(),
              this->mOutput->getData()
            );
}
else
{
//do nothing
}
}
}
cedar::proc::DataSlot::VALIDITY TooFarOrTooClose::determineInputValidity
(
  cedar::proc::ConstDataSlotPtr slot,
  cedar::aux::ConstDataPtr data
) const
{
  if (this->getInputSlot("perceptual error 1") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
  }
  else if (this->getInputSlot("target 1") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
```

```
}
  }
  else if (this->getInputSlot("c_on") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
{
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
  }
  else if (this->getInputSlot("c_on perceptual error") == slot)
  {
    if (boost::shared_dynamic_cast<cedar::aux::ConstMatData>(data))
    {
  // Mat data is accepted.
  return cedar::proc::DataSlot::VALIDITY_VALID;
}
else
{
  // Everything else is rejected.
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
  }
  return cedar::proc::DataSlot::VALIDITY_ERROR;
}
void TooFarOrTooClose::inputConnectionChanged(const std::string& inputName)
{
  cedar::aux::ConstDataPtr data = this->getInput(inputName);
  if (inputName == "perceptual error 1")
  {
    this->mPE1.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
        boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
  this->mPE1 = mat_data;
}
else
{
```

```
  return;
}
  }
  else if (inputName == "target 1")
  {
    this->mTarget1.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
            boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
  this->mTarget1 = mat_data;
}
else
{
  return;
}
  }
  else if (inputName == "c_on")
  {
this->mCOn.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
            boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
  this->mCOn = mat_data;
}
else
{
  return;
}
  }
  else if (inputName == "c_on perceptual error")
  {
    this->mCOnPE.reset();
if (cedar::aux::ConstMatDataPtr mat_data =
            boost::shared_dynamic_cast<const cedar::aux::MatData>(data))
{
  this->mCOnPE = mat_data;
}
else
{
  return;
}
  }
}
```

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für eine andere Prüfung eingereichten Arbeit.
Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

_____ _____

Datum Unterschrift