Testing and improving a vision-model for acuity prediction
Soede, M.J.

Published: 01/01/1994

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Dec. 2018
Testing and Improving a Vision-Model for Acuity Prediction

Michiel Soede

coach at the IPO: drs. A.A.J. Roelofs
coach at WFW: dr. ir. Bram de Jager

December 1994

1 Institute for Perception Research, PO Box 513, 5600 MB Eindhoven
7.1.1 mkgauss.pane ........................................ 31
7.1.2 mkgauss.prog ....................................... 31
7.2 Second derivative of gauss filter .................. 33
  7.2.1 ZCfilter.pane ................................... 33
  7.2.2 ZCfilter.prog ................................... 34
7.3 Convolution of images ................................ 36
  7.3.1 convolXY.pane .................................. 36
  7.3.2 convolXY.prog .................................. 36
7.4 Cartesian to polar transformation ................ 37
  7.4.1 phi_r.pane ...................................... 41
  7.4.2 phi_r.prog ...................................... 41
7.5 Envelop detection .................................. 44
  7.5.1 envelop.pane .................................... 44
  7.5.2 envelop.prog .................................... 45
7.6 Unix batch file for convolution .................. 49
  7.6.1 batmod .......................................... 49
  7.6.2 getfilt .......................................... 52
  7.6.3 getring .......................................... 52
  7.6.4 convring ........................................ 52
7.7 Unix batch file for envelop detection ........... 54
  7.7.1 batenvelop ...................................... 54
  7.7.2 addline.c ....................................... 54
7.8 C-program for extracting parameters: transform.c 55
7.9 Computing the eye filter ............................ 58

8. References ............................................. 59
1. Introduction

A good visual system (eyesight) is very important in today's society. Reduced visual capacities can be corrected surgically in a number of cases. Usually problems still exist after surgery. What can be done to improve the vision of persons with eye deficiencies is the subject of a research project at the IPO performed by drs. A.A.J. Roelofs. The nature of the relation between different eye deficiencies and the effects of them on the human visual system has not yet been fully determined. More fundamental research on this relation is needed to be able to link the fields of low-vision and image processing. This can be done by first choosing and testing a model for normal-sighted persons. If this model is accurate enough for our purposes (predict experimental results), it can be adapted to bring in certain eye deficiencies. In order to test a model, experiments were excercised with a number of normal sighted observers. This report describes the first steps of a model for the eye.

A way to determine the visual acuity of a person is to test which size of letters he can read at a certain distance. Instead of letters Landolt-rings (also called C-rings) can be used. A landolt ring is a ring with an opening at different positions (up, down, left or right). The person whose acuity has to be determined must say at which side the opening was. At a certain size he can no longer recognize the opening. If you offer the same ringsize a number of times and make a graph with on the x-axis the ringsize and on the y-axis the chance he recognized the ring (0 not recognized, 1 recognized), it will not display a sudden change from 0 to 1 but rather a smooth transition. The threshold ringsize could be defined as the ringsize at which the chance of recognizing is 0.5. In this report the chance was taken at 0.78. At this value the acuity of the person will be determined.

A Visual Programming System for software development called 'Khoros' was used to design the model for the eye. Khoros is a public domain software package created at the university of New Mexico.
2. Model input and experimental results

2.1 ACUITY

Visual acuity is the ability to see fine details in a scene. It is the most important single variable determining reading rate according to Aberson and Bouwhuis: "Silent reading as determined by age and visual acuity". With good acuity for example you can discriminate two black dots placed close to each other on a white background instead of one blurred object. Several methods of measuring acuity are available. All involve a description of the amount of space occupied by the target object called Visual angle. Visual angle means the size of the angle formed by extending two lines from your eye to the outside edges of the target.

The definition of acuity used in the experiments is:

\[ acuity = \frac{1}{\text{visual angle (minutes)}} \]  

1 degree (°) equals 60 minutes (') equals 3600 seconds (")

2.2 VISUAL STIMULI

To be able to design a model of the eye visual stimuli were created in the form of landolt-rings. A landolt ring looks like the letter 'C'. Its dimensions (size, width of the opening, width of the ring) are defined in figure 1. The visual angle is determined by the size of the opening and not the size of the ring (acuity = 1/visual angle).

E.g. if the size of the ring (diameter) is 100 mm, the size of the opening is 20 mm and the width of the ring is also 20 mm.

17 different rings were created digitally in an image of 900x900 dots. The sizes in dots (number of pixels on computer screen) of the rings are:

<table>
<thead>
<tr>
<th>Size in dots:</th>
<th>475</th>
<th>375</th>
<th>300</th>
<th>240</th>
<th>190</th>
<th>150</th>
<th>120</th>
<th>95</th>
<th>75</th>
<th>60</th>
<th>45</th>
<th>40</th>
<th>30</th>
<th>25</th>
<th>20</th>
<th>15</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size in mm:</td>
<td>145</td>
<td>115</td>
<td>92</td>
<td>73</td>
<td>58</td>
<td>46</td>
<td>37</td>
<td>29</td>
<td>23</td>
<td>18</td>
<td>14</td>
<td>12</td>
<td>9.2</td>
<td>7.6</td>
<td>6.1</td>
<td>4.6</td>
<td>3.1</td>
</tr>
</tbody>
</table>

The monitor that was used has a resolution of 83 dpi. 
1 inch = 2.54 cm = 25.4 mm : 475 dots = (475/83)*25.4 mm ≈ 145 mm.
All stimuli will be shown to normal sighted observers. The distance between the observers and the computer monitor is 6 m. The acuity must be calculated with the size of the opening (one fifth of the ring size). At a great distance the visual angle is the inverse tangent of the size of the opening on the screen divided by the distance (multiply by 60 to get the angle in minutes):

\[
\arctan \left( \frac{\text{ringsize}}{5} \cdot \frac{25.4}{83} \right) \cdot 60
\]

The visual acuity can be calculated using (1).

In order to obtain more parameters to create a model more stimuli were created. First the contrast of the rings is changed. Both negative and positive contrast rings were created (negative means: ring is darker than its background, positive means background is darker). Another stimulus was created using blur filters. The image is blurred with digital filters of different sizes.

In this report only 13 ring sizes were used:

**Size in dots:** 475 375 300 240 190 150 120 95 75 60 45 30 25

### 2.2.1 Negative contrast rings

Four different types of negative contrast rings were created. The contrast measure used in the experiments is the Michelson contrast, which is defined as:

\[
\frac{L_\text{m} - L_\text{s}}{L_\text{m} + L_\text{s}}
\]

$L_\text{m}$ is the minimum luminance level (cd/m²) and $L_\text{s}$ is the maximum luminance level. Background and foreground grey-values (0 means black, 255 means white) are values in the image. If you display a ring on the monitor the luminance (cd/m²) can be measured. On a logarithmic scale, the distance between the contrast values is equal.
Chapter 2

Model input and experimental results

Table 1: values for negative contrast rings

<table>
<thead>
<tr>
<th>type of stimuli</th>
<th>background value</th>
<th>foreground value</th>
<th>Luminance background</th>
<th>Luminance foreground</th>
<th>Michelson contrast</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg. cont. 1</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
<td>0.986</td>
</tr>
<tr>
<td>neg. cont. 2</td>
<td>255</td>
<td>141</td>
<td>97.2</td>
<td>54.00</td>
<td>0.286</td>
</tr>
<tr>
<td>neg. cont. 3</td>
<td>255</td>
<td>215</td>
<td>97.2</td>
<td>82.30</td>
<td>0.083</td>
</tr>
<tr>
<td>neg. cont. 4</td>
<td>255</td>
<td>244</td>
<td>97.2</td>
<td>92.60</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Figure 2: negative contrast rings 1 to 4 (size = 60 dots)

2.2.2 Positive contrast rings

The values of the positive contrast rings are given in the following table:

Table 2: values for positive contrast rings

<table>
<thead>
<tr>
<th>type of stimuli</th>
<th>background value</th>
<th>foreground value</th>
<th>Luminance background</th>
<th>Luminance foreground</th>
<th>Michelson contrast</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos. cont. 1</td>
<td>20</td>
<td>255</td>
<td>7.71</td>
<td>97.20</td>
<td>0.853</td>
</tr>
<tr>
<td>pos. cont. 2</td>
<td>20</td>
<td>43</td>
<td>7.71</td>
<td>16.50</td>
<td>0.363</td>
</tr>
<tr>
<td>pos. cont. 3</td>
<td>20</td>
<td>27</td>
<td>7.71</td>
<td>10.51</td>
<td>0.156</td>
</tr>
<tr>
<td>pos. cont. 4</td>
<td>20</td>
<td>23</td>
<td>7.71</td>
<td>8.82</td>
<td>0.067</td>
</tr>
</tbody>
</table>

Figure 3: positive contrast rings 1 to 4 (size = 60 dots)
2.2.3 Blurred rings

The third set of stimuli was generated with digital filters of different sizes. To create blurred rings a binomial filter was used.

A binomial filter with length m and n=m+1 elements is defined by:

$$\frac{1}{2^m} \left( \binom{m}{0} \binom{m}{1} \binom{m}{2} \cdots \binom{m}{m} \right)$$  \hfill (4)

The area of this filter is set to 1.

E.g. a binomial filter with length 4 gives the following filter kernel:

$$\left[ \frac{1}{16}, \frac{4}{16}, \frac{6}{16}, \frac{4}{16}, \frac{1}{16} \right]$$  \hfill (5)

and has 5 elements.

The next step is to compute the convolution of the images with the filters. 5 different filters were used:

**Table 3:** values for blurred rings

<table>
<thead>
<tr>
<th>Type of stimuli</th>
<th>Filter length</th>
<th>Background grey value</th>
<th>Foreground grey value</th>
<th>Luminance background</th>
<th>Luminance foreground</th>
</tr>
</thead>
<tbody>
<tr>
<td>blur 1</td>
<td>50</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
</tr>
<tr>
<td>blur 2</td>
<td>100</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
</tr>
<tr>
<td>blur 3</td>
<td>200</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
</tr>
<tr>
<td>blur 4</td>
<td>400</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
</tr>
<tr>
<td>blur 5</td>
<td>5000</td>
<td>255</td>
<td>0</td>
<td>97.2</td>
<td>0.69</td>
</tr>
</tbody>
</table>

**Figure 4:** Blurred rings 1 to 5 (size = 60 dots)
2.3 RESULTS OF ACUITY TEST

Acuity values for a number of observers were determined. The results of these experiments performed by Roelofs are:

<table>
<thead>
<tr>
<th>type of stimuli</th>
<th>average acuity</th>
<th>ringsize (dots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>negcont 1</td>
<td>1.03</td>
<td>27.68</td>
</tr>
<tr>
<td>negcont 2</td>
<td>0.75</td>
<td>38.02</td>
</tr>
<tr>
<td>negcont 3</td>
<td>0.45</td>
<td>63.37</td>
</tr>
<tr>
<td>negcont 4</td>
<td>0.19</td>
<td>150.09</td>
</tr>
<tr>
<td>poscont 1</td>
<td>0.92</td>
<td>31.60</td>
</tr>
<tr>
<td>poscont 2</td>
<td>0.66</td>
<td>43.21</td>
</tr>
<tr>
<td>poscont 3</td>
<td>0.4</td>
<td>71.29</td>
</tr>
<tr>
<td>poscont 4</td>
<td>0.29</td>
<td>98.33</td>
</tr>
<tr>
<td>blur 1</td>
<td>0.8</td>
<td>35.65</td>
</tr>
<tr>
<td>blur 2</td>
<td>0.78</td>
<td>36.56</td>
</tr>
<tr>
<td>blur 3</td>
<td>0.71</td>
<td>40.16</td>
</tr>
<tr>
<td>blur 4</td>
<td>0.57</td>
<td>50.03</td>
</tr>
<tr>
<td>blur 5</td>
<td>0.24</td>
<td>118.82</td>
</tr>
</tbody>
</table>

The ringsize (at threshold size) was computed with (2) and (1).
3. The Vision model

A model for the eye was created using the software-package Khoros. This program uses a workspace. On this workspace icons called 'glyphs' are placed. Each glyph performs a mathematical operation on input-data. The output-data of one glyph can be the input-data of another one. Combining a number of glyphs creates one large program. The workspace of our vision model is:

![Figure 5: Khoros workspace of the vision-model](image)

The Khoros environment works with a standard file format called 'viff-format'. A large number of standard operations (matrix algebra, image processing) is available.

3.1 PHYSICAL BACKGROUND

The input-data to the model are the stimuli discussed before. Intensity changes in the images are important to the model. With stimuli like negative contrast rings the contour of the ring is very clear. With blurred rings the contour is not clear at all. If you want to detect the edges of a blurred image you have to use some kind of edge detection. A theory of edge detection was presented by Marr and Hildreth. Intensity changes, which occur in a natural image over a wide range of scales, are detected separately at different scales. An appropriate filter for this purpose at a given scale is found to be the second derivative of a Gaussian. Intensity changes at a given scale are best detected by finding the zero values of: \( \nabla^2 G(x,y) * I(x,y) \) for image \( I \), where \( G(x,y) \) is a two-dimensional Gaussian distribution and \( \nabla^2 \) is the Laplacian. The intensity changes thus discovered in each of the channels are then represented by oriented primitives called zero-crossing segments.

In this report only one channel will be used namely the channel that detects the smallest details in an image.

A value for the size of the filter \( G(x,y) \) is discussed in Marr (Vision). A \( \sigma \)-value for a filter can be computed: \( \sigma = 3.798 \) dots at a distance of 6 m. See Appendix 7.9 for more details about how this value was computed. The chosen value for the filter will be used from here on.
3.2 ZERO CROSSINGS

To detect intensity changes zerocrossings must be calculated. A sudden intensity change will give rise to a peak or trough in the first derivative or, equivalently, to a zerocrossing in the second derivative.

![Figure 6: An intensity change (a) and its first (b) and second derivative (c)](image)

If zerocrossings of an image with sharp intensity changes are computed, this will result in large peeks in the second derivative. To limit the rate at which intensities can change, we first convolve the image $I$ with a two-dimensional Gaussian operator $G$. The size of this filter is a measure how much detail the observer can see in the image. After blurring the image the second derivative of the image is taken. To do this the Laplacian operator is used. This operator is an orientation-independent differential operator.

A one-dimensional definition of a Gaussian distribution is:

$$G_1(x) = \frac{1}{\sqrt{\pi \sigma}} e^{-\frac{x^2}{2\sigma^2}}$$  \hspace{1cm} (6)

This definition of the filter was chosen so that the relation between this filter and the binomial filter used to create blurred rings becomes:

$$\sigma = \sqrt{\frac{m}{2}} \quad \text{or} \quad \sigma = \sqrt{\frac{n-1}{2}}$$  \hspace{1cm} (7)

The error made in this approximation of a binomial filter goes to 0 for $m \to \infty$ (this was proven by Jean-Bernard Martens, *The Hermite Transform-Theory* (1990)).
Chapter 3

The Vision model

A two-dimensional version of the blur filter is:

\[ G_2(x, y) = G_1(x)G_1(y) = \frac{1}{\pi \sigma^2} e^{-\frac{(x^2 + y^2)}{\sigma^2}} \]  \hspace{1cm} (8)

The next step is to convolve the image with this filter and then use the Laplacian operator on the image. It is also possible to create a filter that performs these two operations in one step:

\[ \nabla^2 G_2(x, y) = \frac{\partial^2 G_2(x, y)}{\partial x^2} + \frac{\partial^2 G_2(x, y)}{\partial y^2} \]  \hspace{1cm} (9)

using the following relation:

\[ (\nabla^2 G) * I = \nabla^2 (G*I) \]  \hspace{1cm} (10)

Because \( G_2(x,y) \) was created using two one-dimensional filters this filter can also be written as:

\[ \nabla^2 G_2(x, y) = \frac{\partial^2 (G_1(x)G_1(y))}{\partial x^2} + \frac{\partial^2 (G_1(x)G_1(y))}{\partial y^2} = \frac{d^2 G_1(x)}{dx^2}G_1(y) + G_1(x) \frac{d^2 G_1(y)}{dy^2} \]  \hspace{1cm} (11)

Convolution of the image with (9):

\[ (\nabla^2 G_2(x, y)) * I(x, y) \]  \hspace{1cm} (12)

Convolution with (11):

\[ \left[ \frac{d^2 G_1(x)}{dx^2}G_1(y) + G_1(x) \frac{d^2 G_1(y)}{dy^2} \right] * I(x, y) = \]  \hspace{1cm} (13)

This equation can also be written as:

\[ G_1(y) * \left( \frac{d^2 G_1(x)}{dx^2} * I(x, y) \right) + G_1(x) * \left( \frac{d^2 G_1(y)}{dy^2} * I(x, y) \right) \]  \hspace{1cm} (14)

A convolution of a 2-dimensional filter with an image is rewritten as four convolutions of
Chapter 3

The Vision model

a 1-dimensional filter with an image. This has great computational advantages (much faster).

Equation (11) needs the second derivative of a one-dimensional Gaussian:

$$\frac{d^2 G_i(x)}{dx^2} = \frac{2}{\sqrt{\pi} \sigma^3} e^{\left(-\frac{x^2}{\sigma^2}\right)} \left(1 - \frac{x^2}{\sigma^2}\right)$$  \hspace{1cm} (15)

An example of a 1-D Gaussian and its second derivative:

Figure 7: A Gaussian ($\sigma = 10$ dots) and its second derivative

Examples of convolutions with landolt-rings:

Figure 8: (a): landolt ring, (b): blurred ring (binomial filter-length=100, size = 60 dots), (c): convolution of (b) with $\nabla^2 G$ ($\sigma=3.798$ dots (§3.1))
In figure 8c you can see the zerocrossings: grey values are 0, dark values are negative and white values are positive. Zerocrossings are at positions in figure 8c where a grey value turns from positive to negative (or from negative to positive).

Figure 9: (a): blurred ring (binomial filter-length=400, size = 60 dots), (b): convolution of (a) with $V^2G$ ($\sigma$=3.798 dots)

The opening of the ring in figure 8c was still open (no zerocrossings at the opening). In figure 9b the opening is closed. The blur factor of that ring caused that effect. To analyze what happens at the opening (when does it close, how strong are the slopes of the zerocrossings) the outer (and inner) envelop (and values of the slope of the zerocrossings at the envelop) of a landolt ring will be determined. For that reason a coordinate transformation will be done (§3.3).

3.2.1 Definition of the slope value

The sharpness of the second derivative at the zerocrossing (slope) must be defined. In figure 13 an example of a zerocrossing is shown. $a_1$ to $a_4$ are values of the 2nd derivative.

Figure 10: example of a zerocrossing
Definition of the slope:

$$\text{slope} = \frac{|a_1 + a_2| + |a_3 + a_4|}{6}$$ (16)

### 3.2.2 Convolution of images with large blur filters

Blur stimuli were convolved with binomial filters. A $\sigma$-value for these filters can be computed using the relation (5): $\sigma = \sqrt{m/2}$:

<table>
<thead>
<tr>
<th>type of stimuli</th>
<th>filter length</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>blur 1</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>blur 2</td>
<td>100</td>
<td>7.071</td>
</tr>
<tr>
<td>blur 3</td>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>blur 4</td>
<td>400</td>
<td>14.142</td>
</tr>
<tr>
<td>blur 5</td>
<td>5000</td>
<td>50</td>
</tr>
</tbody>
</table>

Numerical problems arose computing the second derivative (with a small filter: $\nabla^2 G$, $\sigma = 3.798$) of blurred stimuli (blur 5: $\sigma = 50$). Because the blurred stimuli were stored as BYTE-images (only grey values from 0 to 255 are allowed), computing the second derivative of such an image resulted in very noisy pictures. A solution to this problem is to combine the blur filter of the stimuli with the filter of the eye, compute $\nabla^2 G$ for this large combined filter and then compute the convolution. The only disadvantage of this solution is that all blur-stimuli have to be recalculated.

The combination of two (Gaussian) blur filters is explained using the Fourier-transform of the Gaussian distribution.

The Fourier transform of a Gaussian (6) has a very simple expression:

$$\mathcal{F} \left[ \frac{1}{\sqrt{\pi \sigma}} e^{-\left( \frac{x}{\sigma} \right)^2} \right] = e^{-\frac{\omega^2 \sigma^2}{4}}$$ (17)

Convolution in the time-domain is equal to multiplication in the frequency domain.

The combination of two blur filters:

$$e^{-\frac{\omega^2 \sigma_1^2}{4}} \cdot e^{-\frac{\omega^2 \sigma_2^2}{4}} = e^{-\frac{\omega^2}{4} (\sigma_1^2 + \sigma_2^2)}$$ (18)
Chapter 3

This gives a simple expression for the combined $\sigma$:

$$\sigma_3 = \sqrt{\sigma_1^2 + \sigma_2^2} \quad (19)$$

A table with all $\sigma$ values for the five blur stimuli (also values for distances 12 m and 18 m are given):

<table>
<thead>
<tr>
<th>distance</th>
<th>$\sigma_{\text{eye}}$</th>
<th>blur 1</th>
<th>blur 2</th>
<th>blur 3</th>
<th>blur 4</th>
<th>blur 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 m</td>
<td>3.798</td>
<td>6.2789</td>
<td>8.0264</td>
<td>10.6970</td>
<td>14.6431</td>
<td>50.1440</td>
</tr>
<tr>
<td>12 m</td>
<td>7.596</td>
<td>12.5578</td>
<td>16.0529</td>
<td>21.3939</td>
<td>29.2862</td>
<td>100.2881</td>
</tr>
<tr>
<td>18 m</td>
<td>11.394</td>
<td>18.8368</td>
<td>24.0793</td>
<td>32.0909</td>
<td>43.9294</td>
<td>150.4321</td>
</tr>
</tbody>
</table>

In the next chapter (§3.3) the reason of adding $\sigma$-values at 12 m and 18 m will be given.
3.3 CARTESIAN TO POLAR COORDINATES

The original images are 900 x 900 pixels. The center of the image is at coordinate (450.5,450.5) (between pixels). If a polar representation is computed, the maximum radius will be 450.

\[
(x, y) \Rightarrow (\varphi, r)
\]

\[
r = \sqrt{(x-x_c)^2 + (y-y_c)^2}
\]

\[
\varphi = \tan^{-1}\left(\frac{y-y_c}{x-x_c}\right)
\]

Most interesting are zero-crossings in radial direction. A number of angles for the transformation must be chosen. If 48 are taken, 3 angles go through the opening. All 450 radial positions are used. The size of the transformed image will be 48 x 450 values.

The transformation procedure goes as follows: For each point \((X_i, Y_i)\) in the image \(r\) and \(\varphi\) are computed (figure 11). One value in the \(\varphi, r\)-table occupies a number of points in the image (area \(A_{\varphi, r}\) \(A\) in figure 11). The center of \((X_i, Y_i)\) lies in this area the value in the image at this position will be added to the position in the \(\varphi, r\) table. This is done for all points in the image. Finally all values in the \(\varphi, r\)-table are divided by the number of points in area \(A_{\varphi, r}\).

At small ringsizes, this transformation will not be accurate. E.g.: a \(\varphi, r\) transformation of a ring with 10 pixels with 48 angles is not possible. There are not enough image-points to create an accurate transformation. The solution to this problem is to increase both the ringsize and the distance the observer (read: computer model) looks at the ring. By increasing the ringsize, the transformation will be more accurate, and by increasing the distance the visual angle will be the same. The filter of the eye and all blur filters also have to be changed. If \(\sigma\) was 2 pixels at 6.0 m, \(\sigma\) will be 4 pixels at 12.0 m. The eye filter itself does not change. Increasing \(\sigma\) is only necessary to obtain the same results at 6.0 m and 12.0 m (or 18.0 m).

If the ringsize gets smaller then 75 the distance will be increased. E.g.: a ring of 60 dots at 6.0 m (\(\sigma= 3.798\)) becomes a ring with 120 dots at 12.0 m (\(\sigma = 7.596\)).
3.4 ENVELOP DETECTION

In the transformed image zerocrossings must be detected. To get a useful parameter for the sharpness of the zerocrossing, the slope at the position of the zerocrossing will be calculated. The envelop will be detected by looking in the $\varphi,r$-table from $r=450$ to $r=0$. If a zerocrossing at $r(i)$ is found, the slope at that position can be determined.

![Graphs showing envelops for negative contrast 1-rings at different sizes.](image)

**Figure 12:** example of envelops for negative contrast 1-rings at different sizes.

The opening in the envelop of negative contrast rings in **figure 12** does not close. At angle 23-25 the radius is 0, this means there was no zerocrossing at these angles (See e.g. **figure 8c**).
Chapter 3

The Vision model

Using large blur filters in figure 13 (σ = 50 dots) the opening is closing at size 150. At size 75, the opening is closed. At size 25 the opening is hardly visible.

3.5 SUMMARY

All the steps in the analysis of stimuli were:

- 13 different types of stimuli were created:
  - 4 types of negative contrast rings
  - 4 types of positive contrast rings
  - 5 types of blurred rings
- 13 (of the 17) different ringsizes per type were used (total 13 x 13 = 169 rings)
- Compute a filter for the eye and take its second derivative
- Compute convolutions of all stimuli with the 2nd derivative of the eye filter
- Transform Cartesian- to Polar coordinates
- Find radial zerocrossings and compute the envelop
4. Zerocrossings of a ring-segment

4.1 SECOND DERIVATIVE OF A RING SEGMENT

In this chapter a 1-dimensional version of a ring: a ring segment is analyzed to get a better idea what happens with the second derivative of a landolt-ring. Although a ring segment has no opening, the effect of what happens with the slope with a large blur filter will be easier to see compared to the 2-dimensional version.

![Diagram of ring-segments and its 2nd derivative](image)

**Figure 14:** ring-segments and its 2nd derivative (ringsize 120 dots $\sigma=10$)

In **figure 14** the peaks of the 2nd derivative lie far apart. What happens if the rings get smaller? The peaks will probably interact with each other. To analyse this, the second derivative of a number of ringsizes (475, 240, 190, 60 dots) will be computed. All rings were computed at a distance of 18 m so the actual sizes must be multiplied by a factor 3. In **figure 14** 900 dots for the segment were used, in **figure 15,16** $3 \times 900 = 2700$ dots were used (distance=18).
Result of these computations:

Looking at figure 15,16 it can be seen that the shape of the 2nd derivative changes for small ringsizes.
4.2 SLOPE OF 2ND DERIVATIVE

As discussed before the slope of the 2nd derivative (at the position of a zerocrossing) was a value we were interested in. It indicates the sharpness of the zerocrossing. The slope of the 2nd derivative at the outer zerocrossing (e.g. position 625 for ring 475 in figure 15) will be computed for all ringsizes (475 to 25 dots).

For large rings (475 to 300) this slope value is constant. For smaller rings this is not the case as can be seen in figure 17.
5. Analysis of all stimuli

Looking at ring-segments gave us an idea what happened with zerocrossings when the ringsize decreases. We also saw that the opening of a landolt-ring closes when using large blur filters. Things to look at are:

- The depth of the opening (when does it close)
- Zerocrossings at the envelop (mean value)
- Area of the opening

![Figure 18: parameters to extract from the envelop of one ring](image)

Zerocross values at the envelop are denoted as 'x'-marks in figure 18.
5.1 DEPTH OF THE OPENING

In figure 16 the depth of the opening is defined as the maximum radius minus the minimum radius.

5.1.1 Depth of opening for negative contrast rings

For all 4 negative contrast types the depth of the opening is the same and linear with the ringsize. The x-marks in the figures are ringsizes at threshold size (table 4), 1-4 are the functions for negative contrast rings 1 to 4.

5.1.2 Depth of opening for positive contrast rings

Same results as for negative contrast rings.
5.1.3 Depth of opening for blurred rings

For blur rings the opening closes at a certain ringsize. Looking at the smallest ringsizes the observers could see ('x'-marks in figure 21), it can be seen that for blur 2 to 5 the position of this ringsize lies after closing of the ring. This is not the case for blur 1. A relation between the ringsize at closing and the \( \sigma \)-value can be determined and is shown in figure 22. The ringsizes at closing of the opening were plotted against \( \sigma \).

A linear fit between the combined \( \sigma \)-value (eye filter + blur filter) and the ringsize at closing can be found (dashed line in figure 22):
Since the $\sigma$-values contain the stimulus-$\sigma$ as well as the model (eye)-$\sigma$, this fit also gives a prediction for different model-filters.

E.g.: $\sigma_{\text{eye}} = 5$, $\sigma_{\text{blur}} = 10 \Rightarrow \sigma_{\text{combined}} = \sqrt{(\sigma_{\text{blur}}^2 + \sigma_{\text{eye}}^2)} = 11.18$. Using these filters the opening of the ring will close at size $3.7 \times 11.18 + 5 = 46.36$.

5.2 MEAN ZEROCROSSINGS OF THE ENVELOP

A mean value of all slopes at the zero-crossings is computed for all rings.

5.2.1 Mean zero-cross slope for negative contrast rings

The shape of figure 23 (1 to 4 means curves for neg. cont. 1 to 4) is the same for each type of negative contrast. For large ringsizes, the slope is constant, for small ringsizes, the slope has a maximum-value. This ringsize of this maximum is the same for each type.
5.2.2 Mean zero-cross slope for positive contrast rings

![Figure 24: mean zc-slopes for pos. cont.](image)

5.2.3 Mean zero-cross slope for blurred rings

![Figure 25: mean zc-slopes for blur](image)

The maxima of the curves in **Figure 25** vary with the blur-types. A relation between these maxima and the σ-value of the combined filter (eye + blur) can be determined (**Figure 26**).
Figure 26: ringsize at maximum zc-slope

B1 to B4 in figure 26 are the ringsizes at the maxima of blur1 to blur4-types, N1 is the maximum of negcont1 ($\sigma_{\text{blur}} = 0$).

A linear fit between the combined $\sigma$-value (eye filter + blur filter) and the ringsizes at the maxima can be found (dashed line in figure 26):

$$\text{Size}_{\text{max}} = 6.0\sigma + 10 \quad (22)$$

Since the $\sigma$-values contain the stimulus-$\sigma$ as well as the model (eye)-$\sigma$, this fit also gives a prediction for different model-filters.

E.g.: $\sigma_{\text{eye}} = 5$, $\sigma_{\text{blur}} = 10 \Rightarrow \sigma_{\text{combined}} = \sqrt{(\sigma_{\text{blur}}^2 + \sigma_{\text{eye}}^2)} = 11.18$. Using these filters the maximum-slope of the ring will be at size $6.0 \times 11.18 + 10 = 77.08$. 
5.3 AREA OF ENVELOP OPENING

Another value which could be an important parameter for the model is the area of the opening of the envelop. The idea behind this parameter is that if the area is small, it is more difficult to identify the opening.

For closed envelops, the area is defined in figure 27 as 'Area 2'. For open envelops the area is defined as 'Area 1'. This was done under the assumption that only the area of the opening in the landolt-ring is important to the model.

5.3.1 Area for negative contrast rings
Chapter 5

Analysis of all stimuli

5.3.2 Area for positive contrast rings

Figure 29: area for pos. cont. rings

5.3.3 Area for blurred rings

Figure 30: area for blurred rings
6. Conclusions

Only parameters for the model were analysed in this report, the model itself is not finished. The input of the model (blur, neg.cont., pos.cont.-stimuli) is the same as used in the experiments with the observers. The results of the experiments have to be fitted to the model (e.g. the model predicts the acuity of normal sighted observers). In this report some parameters that could be used to define this model were analysed.

- A computer model to analyse images (computing the 2nd derivative) was created
- Problems with computing the 2nd derivative of stimuli with large blur filters were solved (increasing the distance) and combining $\sigma$-value for the eye and the $\sigma$-value for the blur filters.
- Three different parameters were analysed: depth, mean zero-cross-slope and area of the envelop.
- A relation between $\sigma$ and the ringsize at closing of the opening was found.
- A relation between $\sigma$ and the maximum-slope of the zero-crossings was found.
Chapter 7

7 Appendices

7.1 GAUSS FILTER

A khoros program to create a gaussian filter kernel using the following formula:

\[ G_f(x) = \frac{1}{\sqrt{\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \]

Further information about this routine: read mkgauss.prog.

7.1.1 mkgauss.pane

```bash
-P 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System'
cantata
-^ 1 0 100x80+10+20 +29+1 'Create Signals (1D)' createID
-^ P 0 80x38+22+2 +18+0 'Gauss filter' mkgauss
-0 1 0 0 1 0 1 50x1=7+2 =10+0 =0+0 =0+0 3 5 =1 0 0 1 0 40x1+2=1 =0+0 3 5 =1 'Number of points' m
-0 1 0 0 1 0 40x1+2=5 =0+0 0 0 0 1 =1 'Sampling frequency' s
-0 1 0 0 1 0 40x1+2=7 =0+0 0 0 0 1 =1 'Standard deviation sigma' 
-sigma
-0 1 0 1 13x2+1=10 =1 'Execute' =1 'do operation' =1 /home/soede/khoros/dev/mkgauss/mkgauss
-H 1 13x2+39+10 'Help' =1 'man page for mkgauss' =1 KHOROS_HOME/doc/manpages/dgsin.1
-B
-R

7.1.2 mkgauss.prog

```bash
-AUTHORS
Michiel Soede
-AUTHORS_END

-SHORT_PROG_DESCRIPTION
Compute one-dimensional gauss-filter
-SHORT_PROG_DESCRIPTION_END

-SHORT_LIB_DESCRIPTION
Compute one-dimensional gauss-filter
-SHORT_LIB_DESCRIPTION_END

-MAN1_LONG_DESCRIPTION
Compute one-dimensional gauss-filter.
The formula used is:

\[ \frac{1}{\sqrt{\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \]

this is not the standard formula for a gaussian distribution.

SYNTAX:

mkgauss -o filter -n 31 -sigma 3 -s 1

- `o`: filename of the filter data
- `n`: length of array to store data-points
  must be odd in order to get a center
  at a pixel position (and not between to
  points).
- `sigma`: value in the formula above.
Chapter 7

-s : sample frequency. e.g. if this parameter is set to 2 you get filter points at 0, 0.5, 1, 1.5 etc.
(sample distance between to points = 1/sample_frequency).

MAN1_LONG_DESCRIPTION

MAN1_EXAMPLES

MAN1_RESTRICTIONS

MAN1_SEE_ALSO

MAN1_LONG_DESCRIPTION_END

MAN1_SEE_ALSO_END

MAN1_SEE_ALSO

MAN1_RESTRICTIONS

MAN1_RESTRICTIONS_END

MAN1_SEE_ALSO

MAN1_SEE_ALSO_END

MAN1_SEE_ALSO

MAN1_SEE_ALSO_END

MAN3_RESTRICTIONS

MAN3_RESTRICTIONS_END

MAN3_SEE_ALSO

MAN3_SEE_ALSO_END

MAN3_RESTRICTIONS

MAN3_SEE_ALSO

MAN3_SEE_ALSO_END

USAGE_ADDITIONS

USAGE_ADDITIONS_END

INCLUDE_INCLUDES

INCLUDE_INCLUDES_END

INCLUDE_ADDITIONS

INCLUDE_ADDITIONS_END

INCLUDE_MACROS

INCLUDE_MACROS_END

MAIN_VARIABLE_LIST

struct xvimage *polydf;

char comment[200];

MAIN_VARIABLE_LIST_END

MAIN_LIBRARY_CALL

sprintf(comment,"gauss filter generated by mkgauss");
polydf = createimage(
       (unsigned long) mkgauss->n_int, /* number of rows */
       (unsigned long) mkgauss->n_int, /* number of columns */
       (unsigned long) VFF_TYP_FLOAT, /* data storage type */
       (unsigned long) 1, /* num_of_images */
       (unsigned long) 1, /* num_data_bands */
       comment, /* comment */
       (unsigned long) 0, /* map_row_size */
       (unsigned long) 0, /* map_col_size */
       (unsigned long) VFF_MS_NONE, /* map_scheme */
       (unsigned long) VFF_MAP_TYP_NONE, /* map_storage_type */
       (unsigned long) VFF_LOC_IMPLICIT, /* location type */
       (unsigned long) 0 ); /* location_dim */

MAIN_LIBRARY_CALL_END

if ( ( (mkgauss->n_int) %2) == 0) /* (void)fprintf(stderr,"no even filters allowed! \n'n' must be odd.");

exit(l);)
if(! lmkgauss(polydf,mkgauss->n_int,mkgauss->s_float,mkgauss->sigma_float))
{ (void)fprintf(stderr, "lmkgauss Failed\n");
exit(l);

MAIN_LIBRARY_CALL_END
7.2 SECOND DERIVATIVE OF GAUSS FILTER

The second derivative of the gauss-formule is:

$$\frac{\partial^2 G(x)}{\partial x^2} = \frac{1}{\sqrt{\pi} \sigma^3} e^{-\frac{x^2}{\sigma^2}} \left( 1 - 2 \frac{x^2}{\sigma^2} \right)$$

Information about the parameters: read zcfilter.prog

7.2.1 zcfilter.pro
Chapter 7

7.2.2 zcfilter.prog

AUTHORS
Michiel Soede

SHORT_PROG_DESCRIPTION
Compute one-dimensional 2nd derivative of a gauss-filter

SHORT_LIB_DESCRIPTION
Compute one-dimensional 2nd derivative of a gauss-filter

MAN1_LONG_DESCRIPTION
Compute one-dimensional 2nd derivative of a gauss-filter. The gauss formula is:
\[
\frac{1}{\sqrt{\pi} \sigma} e^{-\left(\frac{x}{\sigma}\right)^2}
\]

The second derivative of this formula is:
\[
\frac{1}{\sqrt{\pi} \sigma} e^{-\left(\frac{x}{\sigma}\right)^2} \left(1 - 2\left(\frac{x}{\sigma}\right)^2\right)
\]

SYNTAX:
zcfilter -o filter -n 31 -sigma 3 -s 1 -area 1

-o : filename of the filter data
-n : length of array to store data-points must be odd in order to get a center at a pixel position (and not between to points).
-sigma : value in the formula above.
-s : sample frequency. e.g. if this parameter is set to 2 you get filter points at 0, 0.5, 1.5 etc. (sample distance between points= 1/sample_frequency).
-area : set the area of the filter (integral of positive and negative area’s) to this value.

MAN1_RESTRICTIONS

MAN1_EXAMPLES

MAN1_RESTRICTIONS_END
Chapter 7

struct xvimage *polydf;
char comment[200];

sprintf(comment, "gauss filter generated by zcfilter");
polydf = createimage(
  (unsigned long) 1, /* number of rows */
  (unsigned long) zcfilter->n_int, /* number of columns */
  (unsigned long) VFF_TYP_FLOAT, /* data storage type */
  (unsigned long) 1, /* num_of_images */
  (unsigned long) 0, /* num_data_bands */
  comment,
  (unsigned long) 0, /* comment */
  (unsigned long) 0, /* map_row_size */
  (unsigned long) 0, /* map_col_size */
  (unsigned long) VFF_MS_NONE, /* map_scheme */
  (unsigned long) VFF_MAPTYPE_NONE, /* map_storage_type */
  (unsigned long) VFF_LOC_IMPLICIT, /* location type */
  (unsigned long) 0 ); /* location_dim */

if ((zcfilter->n_int) %2) == 0) { (void) fprintf(stderr,"no even filters allowed! ‘n’
  must be odd.");
      exit(1);}

    if(!
lzcfilter(polydf,zcfilter->n_int,zcfilter->s_float,zcfilter->sigma_float,zcfilter->area_float)
    )
    {
        (void) fprintf(stderr, "lzcfilter Failed
" );
        exit(1);
    }

writeimage(zcfilter->o_file,polydf);
freeimage(polydf);
Chapter 7

7.3 CONVOLUTION OF IMAGES

This routine computes the convolution of an image with a second image: the filter kernel. Edge problems (if the routine needs a value of the image outside the image boundaries) are handled by taking a constant value called 'pixval' as a value of the image.

7.3.1 convolXY-pane

- F 4 2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
- N 1 0 100x4+10+20 +37+1 '2D Spatial Domain Filters' convolXY
- F 1 0 80x38+22+2 +7+0 'Two-D Spatial Convolution of Two Images' convolXY
- I 1 0 0 1 0 1 50x1+2+2 +0+0 'Input Image' 'first input image' i1
- I 1 0 0 1 0 1 50x1+2+3 +0+0 'KHOROS_HOME/data/kernels/' 'Input Kernel' 'second input image (filter kernel)' i2
Chapter 7

Appendices

7.3.2 `convolXY.prog`

-AUTHORS
Michiel Soede
-AUTHORS_END

-SHORT_PROG_DESCRIPTION
Compute two-dimensional convolution of a pair of images
-SHORT_PROG_DESCRIPTION_END

-SHORT_LIB_DESCRIPTION
Compute two-dimensional convolution of a pair of images
-SHORT_LIB_DESCRIPTION_END

-MAN1_LONG_DESCRIPTION
`.I convolXY
performs a two dimensional convolution of an image a specified kernel image.
`.LP
The second input image should be smaller than the first input image as it is treated as the filter kernel or weights.
`.LP
-MAN1_LONG_DESCRIPTION_END

-MAN1_EXAMPLES
convolXY -il feath.xv -i2 laplace3x3.xv -o result.xv -pixval 0 -kdots 1
`.LP
Convolves the laplacian3x3 kernel with "feath.xv" and leaves the result in "result.xv". pixval is a value outside the image. kdots is the dotsize of the filter kernel.
-MAN1_EXAMPLES_END

-MAN1_RESTRICTIONS
`.I convolXY will not operate on VFF_TYP_COMPLEX images.
Accumulator error may occur when convolving two large images.
-MAN1_RESTRICTIONS_END

-MAN1_SEE_ALSO
lconvolXY(3), vfft(l)
-MAN1_SEE_ALSO_END

-MAN3_LONG_DESCRIPTION
`.I lconvolXY computes the two-dimensional spatial convolution of two images.
`.LP
Edge problems are handled by using a value outside the image. If the filter kernel gets outside the image (there is no image data available at that point), the value at that position will be 'pixval'.
`.LP
img1 is a pointer to an xvimage structure containing the image to be convolved. img1 is also used to hold the result of the convolved image. This is done to save space, but you must be careful not to overwrite important data.
-MAN3_LONG_DESCRIPTION_END

-MAN3_RESTRICTIONS
`.I convolXY will not operate on VFF_TYP_COMPLEX images.
Accumulator error may occur when convolving two large images.
Chapter 7

```c
#define CHECKINPUT(program, img1, img2) 
    (void) proper_num_images(program, img1, TRUE); 
    (void) match_num_images(program, img1, img2, TRUE); 
    (void) proper_num_bands(program, img1, TRUE); 
    (void) match_num_bands(program, img1, img2, TRUE); 
    (void) proper_map_enable(program, img1, VFF_MAP_OPTIONAL, TRUE); 
    (void) match_map_enable(program, img1, img2, TRUE) 

#include "xvimage.h"

struct xvimage *img1, *img2, *readimage();

int lconvolXY(img1, img2, pixval, kdots)
struct xvimage *img1, *img2;
float pixval;
int kdots;
int one;
int t, f;

one = 1.0;
t = 1;
```

Appendices 38
f = 0;
/* checking for proper type inputs */
if (propertytype("lconvolXY",imgl,VFF_TYP_COMPLEX, FALSE) ||
    propertytype("lconvolXY",img2,VFF_TYP_COMPLEX, FALSE))
{
    (void) fprintf(stderr, "lconvolXY does not operate on VFF_TYP_COMPLEX.");
    return(0);
}
/* Call convolution routine based on image and kernel data types */
/* Convert to float */
if (imgl->data_storage_type != VFF_TYP_FLOAT)
    (void) Ivconvert(imgl,vFF_TYP_FLOAT,f,t,one,one,f);
if (img2->data_storage_type != VFF_TYP_FLOAT)
    (void) Ivconvert(img2,VFF_TYP_FLOAT,f,t,one,one,f);
(float_convolve(img2,imgl,pixval,kdots);
return(1);
float_convolve(kern,img,pval,kdot)
struct xvimage *kern, *img;
float pval;
int kdot;
{
    float a;
    float *result;
    float *image, *kernel, *revkern;
    int rows,cols,krows,kcols;
    int i,j,k,l,x,y,nx,ny;
    rows = img->col_size;
    cols = img->row_size;
    krows = kern->col_size;
    kcols = kern->row_size;
    /* Make room for the resulting image */
    result = (float *)malloc(cols*rows*sizeof(float));
    if(result == NULL) {
        (void) fprintf(stderr,"lconvolXY: insufficient memory available
"),
        return(O);
    }
    image = (float *)img->imagedata;
    kernel = (float *)kern->imagedata;
    /* Build an X- and Y-reversed kernel */
    revkern = (float *)malloc(krows*kcols*sizeof(float));
    if(revkern == NULL) {
        (void) fprintf(stderr,"lconvolXY: insufficient memory available
"),
        return(O);
    }
    for (i=0; i<kcols; i++)
        for (j=0; j<krows; j++)
            revkern[(krows-1-j)*kcols+(kcols-1-i)] = kernel[j*kcols+i];
    /* Set the entire result array to 0 */
    bzero((char *)result,rows*cols*sizeof(float));
    /* compute center of filter kernel
      e.g. if kcols = 5 the center is at 2 (array starts with 0)*/
    nx=(kcols-1)/2;
    ny=(krows-1)/2;
    /* Compute convolution of kernel with image and place in result.
     Edges are handled by truncating the kernel */
    for(i=0;i<kcols;i++)
        for(j=0;j<rows;j++)
            a = 0.0;
            for(k=0;k<kcols;k++)
                for(l=0;l<krows;l++)
                {
/* X,Y is location of pixel in image under the convolution mask */
x = i-(nx-k)*kdot; y = j-(ny-l)*kdot;
if (x >= 0 && /* Left edge */
x < cols && /* Right edge */
y >= 0 && /* Top edge */
y < rows) /* Bottom edge */
{a += revkern[l*kcols+k]*image[y*cols+x];}
else
{a += revkern[l*kcols+k]*pval;}
}
result[j*cols+i] = a;
}
free((char *)revkern);
free((char *)image);
img->imagedata = (char *)result;
return(1);
-LIBRARY_CODE_END

-LIBRARY_MODS

-LIBRARY_MODS_END
7.4 CARTESIAN TO POLAR TRANSFORMATION

Compute a transformation from cartesian coordinates to polar coordinates. A 900 x 900 dots image will be converted to a 48 x 450 dots matrix (φ x r): φ = 0 means 0°(degrees) φ = 12 means: (12/48)*360°. r = 0: center of the image.

7.4.1 phi_r.pane

```
-r 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System'
cantata
-M 1 0 100x40+10+20 +33+1 'Unary Arithmetic Operations' unary
-I 1 0 50x1+22+2 +6+0 'Determine envelop of radius as function of angle representation' phi_r
-J 1 0 0 1 0 1 50x1+22+2 +0+0 'Input Image' 'input image'
-O 1 0 0 1 0 1 50x1+22+4 +0+0 'Output Image' 'resulting output image'
-H 1 0 1 13x2+1+8 'Execute' 'execute operation' /home/soede/khoros/dev/phi_r/phi_r
-H 1 13x2+39+8 'Help' 'man page for vscale' KHOROS_HOME/doc/manpages/vscale.1
-K
-D
-E
```

7.4.2 phi_r.prog

```
/* The following define checks for proper values of:
   num_of_images = 1
   map_enable = VFF_MAP_OPTIONAL
*/
#include_additions
#includeDefines

#define CHECKINPUT(program, img1) \( (void) proper_num_images(program, img1,1,TRUE); \)
```
\(\text{(void)}\) proper_map_enable(program, imgl, VFF_MAP_OPTIONAL, TRUE)

-include_macros_end

-main_variable_list
  struct xvimage *image, *readimage();
-main_variable_list_end

-main_before_lib_call
  /* if (check_args()) exit(1); */
  image = readimage(phi_r->i_file);
  if ((image == NULL) exit(1)); /* Quit if error reading the image */
  CHECKINPUT(program, image); /* Verify input image */

-main_before_lib_call_end

-main_library_call
  if (! lphi_r(image))
  { (void) fprintf(stderr, "lphi_r Failed

  MAIN_LIBRARY_CALL_END

-main_after_lib_call
  writeimage(phi_r->o_file, image);

-main_after_lib_call_end

-library_includes
  #define nr_of_angles 48
-library_includes_end

-library_input
  -library_input_end

-library_output
  -library_output_end

-library_def
  int lphi_r(img)
  struct xvimage *img;
  -library_def_end

-library_code
{
  int row, row2, col, xend, yend, nc, nr, index, diameter_nr,
  lineangle_nr, x_coord, y_coord, x_c, y_c, circle_x,
  circle_y, found, increment, diameter, angle,
  angle_nr, radius_nr, min_angle_nr,
  i, j;
  float minimum, maximum, mean, rowval,
  /* points image row */
  /* points image row search */
  /* step in img up to here */
  /* step in img up to here */
  /* column size of image */
  /* row size of image */
  /* mean value of phi_r */
  /* value in phi_r matrix */
Chapter 7

`*ptr, /* pointer to the image data char */
*result; /* pointer to resulting image char */

double min_angle, val;
double pl, size_mm, eccentricity, dpi, dir_avg[4], min;
double sum_sq, rms, square_dist, rms_values[1024][2], middle_x, middle_y;
double diff_x, diff_y, phi_r[nr_of_angles][450];
int NOP[nr_of_angles][450];

char *program; /* program */

program = "lp_hil_r";

property_type(program, img, VFF_TYP_FLOAT, l);

/* Allocate space for resulting image */
result = (float *)malloc((unsigned int)450*nr_of_angles*sizeof(float));
if(result == NULL) {
    (void) fprintf(stderr,"lphi_r: insufficient memory available\n");
    return(0);
}

/* Zero array for result */
bzero((float *) result, (int)1*nr_of_angles * sizeof(float));

/* Assign image data address to ptr */
ptr = (float *) img->imagedata;

/* New representation is being computed; radius as a function of nr_of_angles angles.
After that the number for each radius in each direction is weighted to represent
the size invariance of the visual system. */
pi=3.141592654;
dpi=83;

for (i=0; i<nr_of_angles; i++){
    for (j=0; j<450; j++){
        phi_r[i][j]=0; NOP[i][j]=0;
    }
}
middle_x = nc/2;
middle_y = nr/2;

for (j=0; j<nr; j++){
    for (i=0; i<nc; i++){
        index=(j*nc+i);
        
        diff_x=i-middle_x;
        diff_y=middle_y-j;
        radius_nr=(int)(floor((double)sqrt((double)(diff_x)*diff_x)+
        (double)(diff_y)*diff_y)));
        if (radius_nr<450){
            if (diff_x!=0){
                angle=(int)((360/(2*pi))*atan(diff_y/diff_x));
            }
            else {
                angle=0;
            }
            if (i<middle_x)
            
            angle=angle+180;
            
            if (i==middle_x && j==middle_y)
            
            angle=angle+360;
        }
        
        angle_nr=(int)(floor(((double)angle)/(double)(360)/(double)(2)*(double)(nr_of_angles)))/{{
Chapter 7

```c
double (360)/(double) (nr_of_angles));
if (angle_nr==nr_of_angles)
{
    angle_nr=0;
}
phi_r[angle_nr][radius_nr]=phi_r[angle_nr][radius_nr]+ptr[index];
NOP[angle_nr][radius_nr]++;
}
```

```c
/* Weighing the array to represent the size invariance of the */
/* visual system. */

for (row = 1 ; row < 450; row++) {
    for (col = 0 ; col < nr_of_angles; col++) {
        if (NOP[col][row]!=0)
        {
            result[row*nr_of_angles+col]=(float) (phi_r[col][row]/NOP[col][row]);
        }
    }
}
```

```c
img->col_size=450;
img->row_size=nr_of_angles;
free (img->imagedata);
img->imagedata = (float *)result;
return(1);
```

7.5 ENVELOP DETECTION

Short description of this routine:

Compute zerocrossing of phi_r matrix (created with phi_r.prog). When a zerocrossing is found compute the slope value and store the result in a new matrix.
Next compute the outer envelop:
Look in this zerocrossings matrix from the outside (r=449) to the center until a value above the threshold-parameter is found. Do this for every angle.
Also compute the inner envelop:
Look in the matrix from the center (r=0) to the outside until a value above the threshold is found.

7.5.1 envelop.pane

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System'
cantata
-N 1 0 100x40+10+20 +23+1 'Unary Arithmetic Operations' unary
P 1 0 80x38+22+2 +6+0 'Determine envelop of radius as function of angle representation'
envelop
-I 1 0 0 1 0 1 50x1+2+2 +0+0 'Input Image' 'input image' i
-O 1 0 0 1 0 1 50x1+2+4 +0+0 'Output Image' 'resulting output image' o
-O 1 0 0 1 0 1 50x1+2+5 +0+0 'Zerocross Image' 'resulting zerocross image' phir
-f 1 0 0 1 0 50x1+2+8 +0+0 0 100 6 'Reading distance' 'reading distance' distance
Chapter 7

7.5.2 envelop.prog

/* The following define checks for proper values of:
num_of_images = 1
map_enable = VFF_MAP_OPTIONAL */
clude_additions_end
clude_macros
#define CHECKINPUT(program, img1) \
(\void) proper_num_images\((program,\img1,1,TRUE)\); \
(\void) proper_map_enable\((program,\img1,VFF_MAP_OPTIONAL,TRUE)\)
clude_additions_end
clude_macros_end
main_variable_list
struct xvimage *image, *pr_image, \readimage();
char comment[200];
main_variable_list_end
main_before_lib_call
/* if (check_args()) exit(1); */
image = \readimage(empenal->i_file);
if (image == NULL) exit(1); /* Quit if error reading the image */
CHECKINPUT\((program,image)\); /* Verify input image */
sprintf(comment,"phi_r matrix generated by envelop");

/* creates empty structure to store phi_r matrix */

pr_image = createimage(
    (unsigned long) 1,  /* number of rows */
    (unsigned long) 1,  /* number of columns */
    (unsigned long) 1,  /* data storage type */
    (unsigned long) 1,  /* num of images */
    (unsigned long) 1,  /* num data bands */
    comment, /* comment */
    (unsigned long) 0,  /* map_row_size */
    (unsigned long) 0,  /* map_col_size */
    (unsigned long) VFF_MS_NONE, /* map_scheme */
    (unsigned long) VFF_TYP_FLOAT, /* data storage type */
    (unsigned long) VFF_MAPTYP_NONE, /* map_storage_type */
    (unsigned long) VFF_LOC_IMPLICIT, /* location type */
    (unsigned long) 0); /* location_dim */

lenvelop(image,pr_image,envelop->threshold_float,envelop->distance_float,envelop->ZCacc_float)

fprintf(stderr,"lenvelop Failed\n")

return 0;

int lenvelop(img,pr_img,threshold,distance,ZCacc)

struct xvimage *img,*pr_img;

int row, col, nc, nr, i, j, ZCrangle;

float rowval,*ptr,*result,*resultpr;

double val,
ZCenvelop[nr_of_angles], phi_r[nr_of_angles][450], zerocross[450],*zp;

char *program; /* contains the library name */
Chapter 7

program = "lenvelop";

propertype(program, img, VFF_TYP_FLOAT, 1); /* check img. type */

nr = img->col_size;
nc = img->row_size;

/* allocate space for resulting image */
result = (float *)malloc((unsigned int)l*nr_of_angles*sizeof(float));
if(result == NULL) {
(void) fprintf(stderr,"lenvelop: insufficient memory available\n");
return(0);
}

/* allocate space for resulting phi_r image */
resultpr = (float *)malloc((unsigned int)450*nr_of_angles*sizeof(float));
if(resultpr == NULL) {
(void) fprintf(stderr,"lphi_r: insufficient memory available\n");
return(0);
}

/* assign image data address to ptr */
ptr = (float *)img->imagedata;

/* print("put values below accuracy-level to zero\n"); */

/* values below the accuracy in the phi_r matrix will be put to zero*/
for (row = 0 ; row < 450 ; row++)
   for (col = 0 ; col < nr_of_angles ; col++)
      if ((val>0)&&(val<ZCacc)
          &&(val<O)&&(val>-ZCacc))
         phi_r[col][row]=0;

/* compute zerocrossings for each column */
ZCrange=(int)((distance)/6);
if (ZCrange==0) ZCrange=1;
/*for (row=0;row<450;row++) printf("%.6f *,phi_r[0][row]); */
for (col = 0 ; col < nr_of_angles ;col++)
{
   ZP=zerocross; for (i=0; i<450; i++) ZP[i]=0;
   for (row = ZCrange*2; row< (450-ZCrange*2); row++)
      if ( (phi_r[col][row]<0) && (phi_r[col][row+1]>0)
          || (val<0)&&(val>-2Cacc))
         {zerocross[row] = (-phi_r[col][row-ZCrange]+phi_r[col][row+ZCrange]-phi_r[col][row-ZCrange*2]+phi_r[col][row+ZCrange*2])/6;
          if (phi_r[col][row]<0) && (phi_r[col][row+1]>0)
         }
   if (phi_r[col][row]<0) && (phi_r[col][row+1]>0)
      {zerocross[row] = (phi_r[col][row-ZCrange]-phi_r[col][row+ZCrange]+phi_r[col][row-ZCrange*2]-phi_r[col][row+ZCrange*2])/6;
          if ( (phi_r[col][row]<0) && (phi_r[col][row+1]>0))
         }
   else phi_r[col][row]=val;
}

/* from the new angle - radius representation the envelop is taken,
which is equal to the outmost zero-crossing for each of the nr_of_angles directions. */

for (i=0;i<nr_of_angles;i++)
   for (col = 0 ; col < nr_of_angles ; col++)
      {rowval = 0;
       row = 449;
       while ( (row > 0) && (phi_r[col][row]<threshold))
         row--;
       result[i]=rowval;
result[col]=row;
ZCenvelop[col]=phi_r[col][row];
}

* the radius values of the envelop are normalized at a distance of 6.0 meters 
e.g. if the distance is 12.0 meters the radius-values will be multiplied by a factor 
6.0/12.0 = 0.5 
if the distance is 6.0 m the values remain unchanged 
*/
printf(" ");
for (i=0; i<nr_of_angles; i++) printf("%f ",result[i] *(6.0/distance));
printf(" ");
for (i=0 ; i<nr_of_angles; i++) printf("%f ", ZCenvelop[i]) ;

img->col_size=1;
img->row_size=nr_of_angles;
free (img->imagedata);

img->imagedata = (float *)result;

pr_img->col_size=450;
pr_img->row_size=nr_of_angles;
pr_img->imagedata=(float *)resultpr;

for (row = 0 ; row <450; row++)
{ for (col = 0 ;col < nr_of_angles; col++) {resultpr[row*nr_of_angles+col] = (float)phi_r[col][row];}
}
return(1);

/library_code_end
/library_mods
/library_mods_end

Chapter 7

7.6 UNIX BATCH FILE FOR CONVOLUTION

To compute convolutions of all the rings (13 sizes and 13 types) a batchfile was used. 'batmod' calls three other batchfiles namely:

- getfilt: computes all filter-kernels neccessary for the convolution.
- getring: get a ring with given size, back- and foreground greyvalue.
- convring: computes 2nd derivative of blurred image and transform to polar coordinates

After running this batchfile 13 x 13 q-r images are computed of 2nd derivatives of all stimuli.

7.6.1 batmod

```bash
#!/bin/sh
#
#

prnr='p10'
echo process number: $prnr
# parameter $1: process-number to create different outputs
# if you want to run more than one process at a time
echo 'KHOROS batch file'

prnr='p10'
echo process number: $prnr
# parameter $1: process-number to create different outputs
# if you want to run more than one process at a time

cd
cd khoros/dev
uncompress -c vision_model8.Z > temp

cd /home/soede/khoros
rm modout"$prnr"

#getfilt : compute gauss filters in X- and Y-direction and the 2nd derivative
# of a gauss filter in X- and Y-direction
SYNTAX: getfilt $prnr windowlength sigma
#
# getring : computes a landolt-ring for negcont and poscont
# SYNTAX: getring $prnr dir size
# dir: negcont1 .. 4 poscont1 .. 4
# size: if the ring gets to small you must increase the distance
# (see also addline.c)
#
# convring: computes zerocrossings and envelop of the ring
# SYNTAX: $prnr distance dir size
for dir in negcont1 negcont2 negcont3 negcont4 poscont1 poscont2 poscont3 poscont4

done
```

```bash
# RINGS AT DISTANCE 6.0
getfilt $prnr 31 3.799
for size in 475 375 300 240 190 150 120 095 075

do

gttering $prnr $dir $size
convring $prnr 6 $dir $size
done
```
# RINGS AT DISTANCE 12.0
getfi1t $prnr 63 7.596
for size in 120 090
do
  getring $prnr $dir $size
  convring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
getfi1t $prnr 93 11.394
for size in 090 075
do
  getring $prnr $dir $size
  convring $prnr 18 $dir $size
done

# RINGS AT DISTANCE 6.0
getfi1t $prnr 51 6.2789
for size in 475 375 300 240 190 150 120 095 075
do
  getring $prnr $dir $size
  convring $prnr 6 $dir $size
done

# RINGS AT DISTANCE 12.0
getfi1t $prnr 103 12.5578
for size in 120 090
do
  getring $prnr $dir $size
  convring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
getfi1t $prnr 155 18.8368
for size in 090 075
do
  getring $prnr $dir $size
  convring $prnr 18 $dir $size
done

# RINGS AT DISTANCE 6.0
getfi1t $prnr 65 8.0264
for size in 475 375 300 240 190 150 120 095 075
do
  getring $prnr $dir $size
  convring $prnr 6 $dir $size
done

# RINGS AT DISTANCE 12.0
getfi1t $prnr 131 16.0529
for size in 120 090
do
  getring $prnr $dir $size
  convring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
getfi1t $prnr 197 24.0793
for size in 090 075
do
  getring $prnr $dir $size
  convring $prnr 18 $dir $size
done

# RINGS AT DISTANCE 6.0
getfi1t $prnr 87 10.6970
for size in 475 375 300 240 190 150 120 095 075
do
gtring $prnr $dir $size
covring $prnr 6 $dir $size
done

# RINGS AT DISTANCE 12.0
gfilt $prnr 175 21.3939
for size in 120 090
do
gtring $prnr $dir $size
covring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
gfilt $prnr 263 32.0909
for size in 120 090
do
gtring $prnr $dir $size
covring $prnr 12 $dir $size
done

dir='blur4'
echo $dir

# RINGS AT DISTANCE 6.0
gfilt $prnr 119 14.6431
for size in 475 375 300 240 190 150 120 095 075
do
gtring $prnr $dir $size
covring $prnr 6 $dir $size
done

# RINGS AT DISTANCE 12.0
gfilt $prnr 239 29.2862
for size in 120 090
do
gtring $prnr $dir $size
covring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
gfilt $prnr 359 43.9294
for size in 090 075
do
gtring $prnr $dir $size
covring $prnr 18 $dir $size
done

dir='blur5'
echo $dir

# RINGS AT DISTANCE 6.0
gfilt $prnr 409 50.1440
for size in 475 375 300 240 190 150 120 095 075
do
gtring $prnr $dir $size
covring $prnr 6 $dir $size
done

# RINGS AT DISTANCE 12.0
gfilt $prnr 819 100.2881
for size in 120 090
do
gtring $prnr $dir $size
covring $prnr 12 $dir $size
done

# RINGS AT DISTANCE 18.0
gfilt $prnr 1227 150.4321
for size in 090 075
do
gtring $prnr $dir $size
covring $prnr 18 $dir $size
done
Chapter 7

7.6.2 getfilt

/home/soede/khoros/dev/mkgauss/mkgauss -o /home/plaatjes/soede/kernels/filtX0"$1" -n "$2" -s 1 -sigma "$3"
/home/soede/khoros/dev/zcfilter/zcfilter -o /home/plaatjes/soede/kernels/filtX2"$1" -n "$2" -s 1 -sigma "$3" -area 1

mtrans -i /home/plaatjes/soede/kernels/filtX0"$1" -o /home/plaatjes/soede/kernels/filtY0"$1" -c 0
mtrans -i /home/plaatjes/soede/kernels/filtX2"$1" -o /home/plaatjes/soede/kernels/filtY2"$1" -c 0

7.6.3 getring

bg='255'
fg='0'
if test $2 = negcont1
  bg='255'
  fg='0'
fi
#
if test $2 = negcont2
  bg='255'
  fg='141'
fi
#
if test $2 = negcont3
  bg='255'
  fg='215'
fi
#
if test $2 = negcont4
  bg='255'
  fg='244'
fi
#
if test $2 = poscont1
  bg='020'
  fg='255'
fi
#
if test $2 = poscont2
  bg='020'
  fg='043'
fi
#
if test $2 = poscont3
  bg='020'
  fg='027'
fi
#
if test $2 = poscont4
  bg='020'
  fg='023'
fi
#
/home/roelofs/khoros/dev/landolt/vglandolt -o /home/plaatjes/soede/temp/vring"$1" -dpi 83 -hor_resolution 900 -ver_resolution 900 -size $3 -direction 2 -backgr_value $bg -foregr_value $fg

7.6.4 convring

echo X2
/home/soede/khoros/dev/convolve/convolXY -il /home/plaatjes/soede/temp/vring"$1" -i2 /home/plaatjes/soede/kernels/filtX2"$1" -o /home/plaatjes/soede/temp/vringX2"$1" -pixval 255 -kdots 1
Chapter 7

Appendices 53

echo Y2
/home/soede/khoros/dev/convolve/convolveXY -i1 /home/plaatjes/soede/temp/vring"$1" -i2
/home/plaatjes/soede/kernels/filtY2"$1" -o /home/plaatjes/soede/temp/vringY2"$1" -pixval 255
-kdots 1
echo Y0
/home/soede/khoros/dev/convolve/convolveXY -il /home/plaatjes/soede/temp/vringX2"$1" -il
/home/plaatjes/soede/kernels/filtY0"$1" -o /home/plaatjes/soede/temp/vringXc"$1" -pixval 0
-kdots 1
echo X0
/home/soede/khoros/dev/convolve/convolveXY -i1 /home/plaatjes/soede/temp/vringY2"$1" -i2
/home/plaatjes/soede/kernels/filtX0"$1" -o /home/plaatjes/soede/temp/vringYc"$1" -pixval 0
-kdots 1
vadd -i1 /home/plaatjes/soede/temp/vringYc"$1" -i2 /home/plaatjes/soede/temp/vringYc"$1" -o
/home/plaatjes/soede/temp/vringXY"$1"
/home/soede/khoros/dev/envelop/envelop -i /home/plaatjes/soede/temp/vringXY"$1" -o
/home/soede/modeldata/"$3"_"$4"_"$2"
compress /home/soede/modeldata/*

7.7 UNIX BATCH FILE FOR ENVELOP DETECTION

For all the phi_r images created with 'batmod' the envelop will be calculated (for all sizes
and type 13 x 13 rings) using a batch-file. 'batenvelop' calls 'envelop.prog' to compute
the inner and outer envelop of one ring. The result will be stored in large matrix called
'modout"prnr"' using the C-program 'addline.c' ('prnr' is a batch variable).

7.7.1 batenvelop

#!/bin/sh
#
$prnr='p4'
echo process number: $prnr
# parameter $1: process-number to create different outputs
# if you want to run more than one process at a time
echo "Khoros batch file"
cd
cd /home/soede/khoros/dev
uncompress -c vision_modeI8.Z > temp
cd /home/soede/khoros

#########################################################################
# This batch program computes the envelop of all the rings (phi_r images) and stores it
# in a file called 'modout$prnr' with the command 'addline'. The envelop is computed
# with the program 'envelop' (Khoros). For information about the result file, see the
# comments in the source file 'addline.c'.
#########################################################################
rm modout"$prnr"
for dir in negcont1 negcont2 negcont3 negcont4 poscont1 poscont2 poscont3 poscont4 blur1 blur2 blur3 blur4 blur5
do
uncompress /home/soede/modeldata/"$dir"/*
# RINGS AT DISTANCE 6.0
# for size in 475 375 300 240 190 150 120 95 75
do
echo "$dir" "$size"
/home/soede/khoros/dev/envelop/envelop -i /home/soede/modeldata/"$dir"/"$dir"_"$size"_6 -o
/home/plaatjes/soede/temp/envel"$prnr" -phi /home/plaatjes/soede/temp/zerocr"$prnr" -distance
6 -threshold 0.01 -zcacc 0.01 > envout"$prnr"
addline "$dir" "$size" 6 envout"$prnr" modout"$prnr"
Chapter 7

# RINGS AT DISTANCE 12.0
for size in 120 090
done
echo "$dir" $size" 
/home/soede/khoros/dev/envelop/envelop -i /home/soede/modeldata/$dir"$size"_12 
-o /home/plaatjes/soede/temp/envl"$prnr" -phir /home/plaatjes/soede/temp/zerocr"$prnr" 
-distance 12 -threshold 0.01 -ZCacc 0.01 > envout"$prnr"
addline "$dir" "$size" 12 envout"$prnr" modout"$prnr"
done

# RINGS AT DISTANCE 18.0
for size in 090 075
done

7.7.2 addline.c

/* Author: M.J. Soede student TUE-W, id 307325
Copyright November 1994
Institute for Perception Research
Eindhoven, The Netherlands */

/* SYNTAX : ADDLINE type size distance env_output modresult

 type        : e.g. negcont1, poscont1 or blur1
 size        : size of landolt ring
 distance    : distance of the experiment

 if the size of the ring gets to small you can
 increase the distance, the size value in the outputfile
 will be multiplied by 6.0/distance.
 blur pictures must be modified at greater distances.
 the zerozero filter must also be modified.

 env_output  : filename to add a new line to.
 modresult   : filename to add a new line to.

 example structure of modresult:
 11 300 ... (12 floats with envelop values) ... (12 floats with zerozero values)
 11 : negcont1 (21 poscont1, 31 blur1, 32 blur2 etc.)
 300 : ringsize*6/distance

 the structure below is defined in envelop.prog!!!!
...
(12 fl): 12 radius values of the outside envelop of the landolt-ring
...
(12 fl): 12 zerozero values at the positions of the radius of ENV

*/

#include <stdio.h>

/* declaration of functions*/
main(argc,argv)
int argc; char *argv[];
{
   FILE *Fin,*Fout;

   int distance,
       size,
       typenr,
       com_cnt;

   char type[40],
env_output[200],
env_dat[2000],
modresult[200];

strcpy(type,**);
size=distance=0;
strcpy(env_output,**);
strcpy(modresult,**);
com_cnt=1;

if (argc==6)
{
strncpy(type,argv[com_cnt],40);
com_cnt++;
scanf(argv[com_cnt],"%d",&size);
com_cnt++;
scanf(argv[com_cnt],"%d",&distance);
com_cnt++;
strncpy(env_output,argv[com_cnt],200);
com_cnt++;
strncpy(modresult,argv[com_cnt],200);
com_cnt++;
}
else {printf("Wrong arguments!\n"); exit(1);}

/* maak van ‘negcontX’ -> 1x
‘poscontx’ -> 2x
’blurx’ -> 3x
*/
switch(type[0])
{
    case ’n’ : typenr=10; break;
    case ’p’ : typenr=20; break;
    case ’b’ : typenr=30; break;

    default : typenr=0;
}
typenr+=(type[strlen(type)-1]’0’);

/* add result in ‘modresult’ */

Fin=fopen(env_output,"r");
fgets(env_dat, sizeof(env_dat),Fin);
fclose(Fin);
Fout=fopen(modresult,"a");
fprintf(Fout,
"%d%3d %s\n", typenr, (int)(size*6.0/distance), env_dat);
fclose(Fout);

}/*main*/

7.8 C-PROGRAM FOR EXTRACTING PARAMETERS: TRANSFORM.C

The following program reads the file created with batenvolup and extracts the mean value of
the slope of the second derivative at the zerocrossing.

/*
   Author: M.J. Soede student TUE-W, id 307325
   Copyright October 1994
   Institute for Perception Research
   Eindhoven, The Netherlands
*/

/* load file with envelop and zerocross data generated by BATENVOLUP
and transform the envelop.
SYNTAX:
transform filename
structure of a line in filename:
Chapter 7

[ tp sz ... radius (48 values) (outer env.) .... zc (48 values) ... radius (48 values) 
(inner env.) ... zc (48 values) ]

tp: type 11= negcont1, 12 =negcont2, 21= poscont1, 31=blur1

equ: [12 90 45 46 45 ...

#include <stdio.h>
#include <math.h>
#define NR_OF_ANGLES 48
#define LINELENGTH 2+48*2

main(argc,argv)
    int argc; char *argv[];
{
    FILE *Fin,*Fout;
    int i,j,
    nlines,
    com_cnt,
    error,
    size_cnt,
    filter_arg,
    filterlength;

    float mean,
    size_mm,
    acuity,
    angle,
    pi,
    dpi,
    eccentricity;

    float pc,
    ic,
    fac1,
    fac2,
    fac3;

    char inputname[200],
    outputname[200],
    command_line[120];

    int type[221],
    size[221];

    float val,
    h,
    totval,
    zc,
    radius[221][NR_OF_ANGLES],
    zerocross[221][NR_OF_ANGLES],
    radius2[221][NR_OF_ANGLES],
    zerocross2[221][NR_OF_ANGLES];

    static int sizes[ ] = {475,375,300,240,190,150,120,95,75,60,45,30,25};
    int min_l,min_r,max_l,max_r;
    pi=3.141592654;
    dpi=83;
    /* Read the command line */
    error=0;
    strcpy(inputname,"");
    strcpy(outputname,"");
    /*default values of the parameters*/
    pc=0.5;
    ic=1;
Chapter 7

```c
fac1=fac2=fac3=1;
com_cnt=1;
if (argc>1) {
    strncpy(inputname,argv[com_cnt],sizeof(inputname));
    com_cnt++;
    /*
     * strncpy(outputname,argv[com_cnt],sizeof(outputname));
     * com_cnt++;
     */
    while (com_cnt<argc) {
        if (strcmp(argv[com_cnt],"-pc")==0) {
            sscanf(argv[com_cnt+1],"%f",&pc);
            com_cnt+=2;
        } else if (strcmp(argv[com_cnt],"-lc")==0) {
            sscanf(argv[com_cnt+1],"%f",&lc);
            com_cnt+=2;
        } else {printf("WRONG ARGUMENT: \"%s\n",argv[com_cnt]); exit(1);}
    }
    /*printf("lc: %f, pc: %f, fac1: %f, fac2: %f, fac3: %f\n",lc,pc,facl,fac2,fac3);
    */
    /* read modeldata-matrix (ascii) file */
    Fin=fopen(inputname,"r");
    if (Fin==NULL) {printf("Not able to open: \"%s\n",inputname);
    } else {
        i=0;
        while ( fscanf(Fin,"%d",&type[i]) !=EOF) && (i<22) ) {
            fscanf(Fin,"%d",&size[i]);
        /*read outer envelop*/
            for (j=0;j<NR_OF_ANGLES;j++)
                fscanf(Fin,"%f",&radius[i][j]);
        /*read inner envelop*/
            for (j=0;j<NR_OF_ANGLES;j++)
                fscanf(Fin,"%f",&zerocross2[i][j]);
        i++;
    }
    fclose(Fin);
}
nlines=i;
printf("\n"); for (i=0;i<13;i++) printf("%7d ",sizes[i]); printf("\n");
for (linenr=0; linenr<nlines;linenr++) {
    /* compute mean zerocross */
    zc=0;
    for (i=0;i<NR_OP_ANGLES; i++) zc+=zerocross[linenr][i];
    zc=zc/NR_OF_ANGLES;
    if ((type[linenr-1] != type[linenr]) || (linenr==0) ) printf("\n%7d ,type[linenr]);
    printf("%7.3f ",zc);
    printf("\n");
    /*for (i=0;i<NR_OF_ANGLES;i++) printf("%f ,radius0[i][i]; printf("\n");*/
    return(1);
}
/*main*/
```
7.9 COMPUTING THE EYE FILTER

A $\sigma$-value for the eye filter was computed with theory from D. Marr page 62-63. The diameter $\omega_{2D}$ of the central part of the receptive field should be about $1'20'' (=0.0222^\circ)$. 

![Figure 31: distance between zero crossings in 2nd derivative of gaussian](image)

This value for $\omega_{2D}$ is given in degrees (0.0222). This value must be converted to dots if we want to use it in our computations. In figure 31 'd' is defined in dots. This distance is the distance we want to know in dots but was defined in degrees. With the definition of the 2nd derivative of the gauss-formule and this value $\omega_{2D}$, $\sigma$ can be calculated.

- distance of the eye to the screen: $l$ = 6 m (=600 cm)
- resolution of the screen: 83 dpi
- 1 inch = 2.54 cm

So the resolution of the screen is also 32.667 dpcm (dots per centimeter). $\omega_{2D}$ was defined for the 2D gaussian (9): its second derivative is $(r^2 = x^2 + y^2)$:

$$\nabla^2 G(r) = \frac{4}{\pi \sigma^4} e^{-\frac{r^2}{\sigma^2}} \left(1 - \frac{r^2}{\sigma^2}\right)$$

This is the same formula as (9), but written in polar-coordinates. Zero's of the 2nd derivative of the gaussian are at $r = \sigma$ and $r = -\sigma$. $d (\omega_{2D}) = 2\sigma$ (or $\sigma = 0.5*d$).

$$\tan(\omega_{2D}) = \frac{d}{l} \Rightarrow d = \tan(\omega_{2D})l = \tan(0.0222\times600) = 0.2325cm$$

$\sigma = 0.5\times0.2325 \text{ cm} = 0.116 \text{ cm}$. In dots: $\sigma = 32.677\times0.116 = 3.798 \text{ dots}$.
8. References

• M. W. Matlin and Hugh J. Foley, "Sensation and Perception", third edition

• D. Marr, "Vision", 1982


• D.H.A. Aberson and D.G. Bouwhuis, "Silent reading as determined by age and visual acuity", Manuscript 547/II