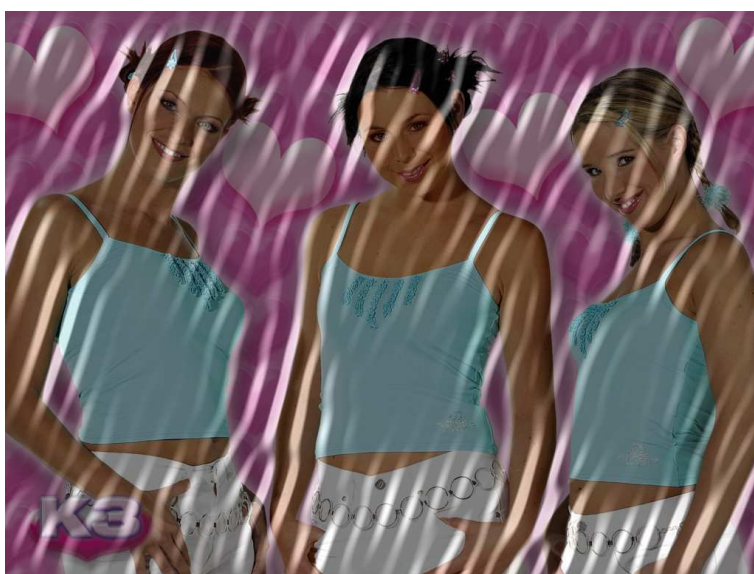


Retinotopic Gabor image recognition, first step in building up Arleo's framework

Richèl Bilderbeek

1st April 2005



Supervised by

Christian Igel (Christian.Igel@neuroinformatik.ruhr-uni-bochum.de)

Ioannis Iossifidis (Ioannis.Iossifidis@neuroinformatik.ruhr-uni-bochum.de)

Contents

1	<i>Abstract</i>	2
2	<i>Introduction</i>	3
2.1	<i>Neural Network</i>	3
2.2	<i>Visual information processing</i>	3
2.3	<i>Idiothetic information processing</i>	4
2.4	<i>Hippocampal Place Cells</i>	5
2.5	<i>Preprocessing snapshots with Gabor filters</i>	6

2.6	<i>View cell recognition</i>	8
2.7	<i>This research</i>	10
3	<i>Research questions</i>	10
4	<i>Materials and methods</i>	10
4.1	<i>Obtaining the retinotopic grid and Gabor filter parameters</i>	10
4.2	<i>Image processing</i>	11
4.3	<i>View cell recognition</i>	12
4.4	<i>View cell recognition after translation</i>	13
4.5	<i>Software implementation</i>	13
5	<i>Results</i>	13
5.1	<i>Obtaining the Gabor filter parameters</i>	13
5.2	<i>Image processing</i>	15
5.2.1	<i>Difference between OpenCV and self-written convolution</i>	15
5.2.2	<i>The effect of histogram equalization on feature detection</i>	17
5.3	<i>View cell recognition</i>	18
5.4	<i>View cell recognition after translation</i>	20
5.5	<i>Software implementation</i>	21
6	<i>Conclusion</i>	27
7	<i>Discussion</i>	27
8	<i>Future work</i>	28

1 Abstract

Producing a robot that is able to learn and navigate autonomously is a difficult task. In May 2004 an article was published (Arleo *et al.*, 2004) describing an experimental setup that has been successful. This setup uses Gabor filter operations, a neural network and reinforcement learning. The described architecture can serve as a reference and a starting point for improvements later. Therefore the goal was to transfer it to the infrastructure of the institute. While implementing the framework many programs available at the institute have been greatly improved in functionality and debugging facilities. Some unmentioned parameters in the paper have been extracted indirectly from it. A convolution algorithm that can process at specific spots has been developed successfully. Although having not completely implemented the architecture, interesting experiments can be performed on single view cell recognition. The retinotopically Gabor filtering system used seems very promising, because it is found to be able to successfully remember a location and able to distinguish it from other, but also the recognition has some flexibility in horizontal translation. This has been concluded by testing on difficult testing images. In the reference, also an image was histogram equalized before being retinotopically Gabor processed. Clues are found that this operation is actually decreasing performance of the visual system.

2 Introduction

In our high-tech society, letting an agent explore and remember a new environment will sound like an easy-to-take hurdle. It is... in simulated environment with a simulated agent. Crossing the gap between theory and practice is still not possible. However recently, Arleo and co-workers produced a biologically-inspired experiment setup to let a Kephra robot explore a new environment and remember the locations previously visited.

2.1 Neural Network

The skeleton of the architecture consists of a multi-layered neural network (see figure 1 below). The network has one pillar with neurons recognizing a location visually (called visual place cells), the other neurons recognize a location from self-motion information (called idiothetic place cells) provided by the odometer. The visual place cells do not recognize a location by sight itself, but are connected to view cells that do. Both of these inputs alone are insufficient for unambiguous location recognition. Because of this, their inputs are combined and processed by a layer of neurons (called Hippocampal place cells) downstream to identify each location unambiguously. When due to this layer a location has been recognized, learning can begin. The Hippocampal place cells' output are converted to motor commands by another layer of neurons, called action cells. Except for the latter, these cell types are described in more detail below, followed by a more detailed section on the processing of the image to make it suitable as input for the visual place cells.

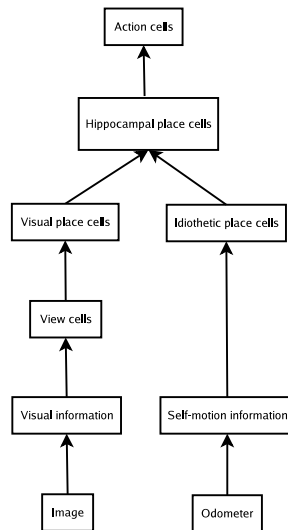


Figure 1: *Schematic overview of the architecture of the program. Adapted from Arleo et al. (2004).*

2.2 Visual information processing

A visual place cell is a neuron that remembers a certain location by visual information. Visual place cells do not obtain their information directly from sight, but from view cells that do. To be

more precise, visual place cells recognize a location by recognizing the responses of the view cell layer. Unlike view cells, they also contain a (hypothetical) relative Cartesian coordinate.

View cells are initialized by extracting values obtained from Gabor filtering from an image taken by a camera. At every (new) input, it measure the difference between the actual Gabor values and the values it is initialized on. The lower this difference, the higher its response (although the response is not directly coupled to the input, see equations 6,7,8).

Location recognition of visual information alone will unavoidably lead to the problem of aliasing. Aliasing means that two different locations are visually identified as more-or-less the same. This problem is illustrated below in figure 2:

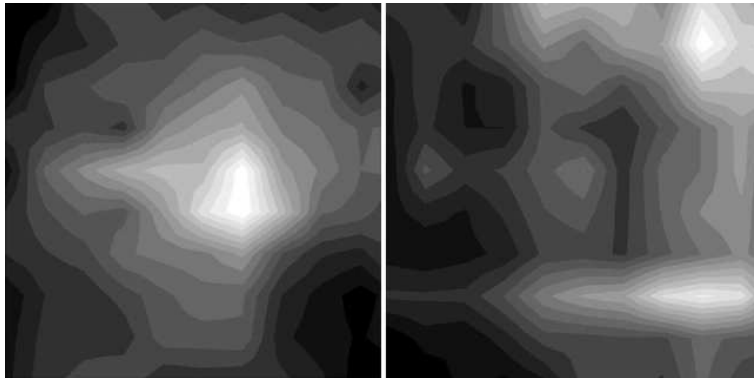


Figure 2: *Recognition of a location by a visual place cell field. The box denotes the arena (80 times 80 centimeter). The color denotes the visual recognition of a certain location, where white means a strong recognition and black a low recognition. Left) Unambiguous recognition of the middle of the area. Right) Indistinct recognition between top-right and bottom-right location. From Arleo et al.(2004).*

This figure, which is actually a result of Arleo *et al.* (2004), shows the visual recognition of a location in an explored area. The area has been stored in multiple visual place cells, their place fields covering it entirely. In the left figure, the visual place cell(s) in the middle recognize the location unambiguously. In the right figure, the location of the robot cannot be determined, because both visual place cells at the top-right and bottom-right recognize the current location.

2.3 Idiothetic information processing

The other pillar of the neural network, are the idiothetic place cells. These are initialized to fire on a certain location, where the location is concluded from measuring the movement of the robot internally. Both wheels of the robot are attached to an odometer, which keeps track of the wheel position. Using their information, a position can be estimated by path integration.

If the recording of self-motion was perfect, path-integration would be accurate. In practice, the accuracy of this technique decreases in time, as shown by figure 3 below:

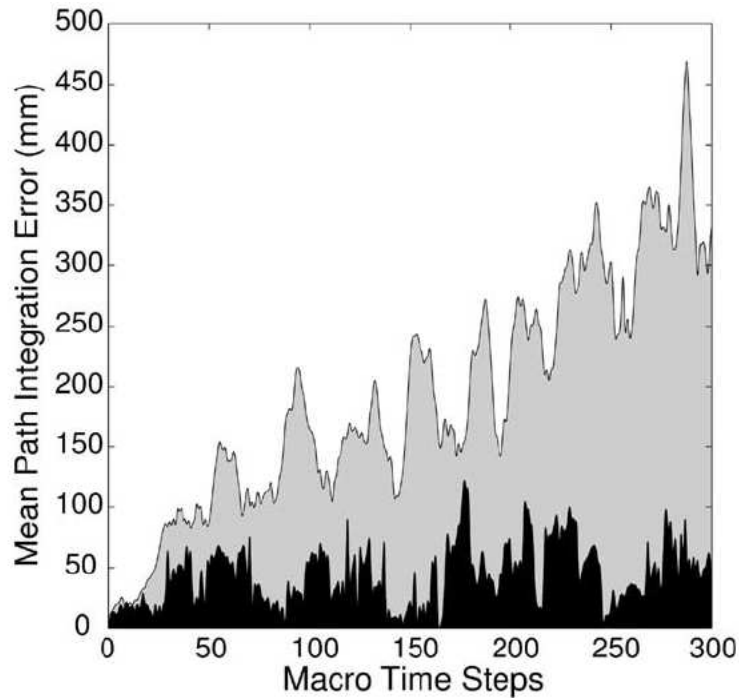


Figure 3: *Accumulation of the error from path integration. The grey line (above the grey area) denotes the estimation error of a position by path integration in time. The black line (above the black area) denotes the same error, except when the position estimation gets reset to the position known for sure by the visual place cells. A macro time step denotes a movement of 5 centimeter in a direction. From Arleo et al.(2004).*

This figure, again a result of Arleo *et al.*(2004), shows that the error by path integration alone (the grey line) accumulates in time. Note that it is also possible for this error to decrease sometimes. This happens when e.g. the robot drives backwards, creating estimation error increase in the other direction. The black line in the figure shows the result of a setup in which the estimated position from idiothetic place cells is adapted to the position estimated by the visual place cells. This means, that when the visual place cells recognize the location unambiguously and strong, this knowledge is passed to the path integrator. This recalibration has not been done in this research, therefore see Arleo *et al.*(2004) for more details.

2.4 Hippocampal Place Cells

While both types of place cells have their problems, the responses of layers of both types are used to determine the position of the robot. The cells in this layer, the Hippocampal place cells, succeed in doing this in Arleo *et al.*(2004). The figure below illustrates this.

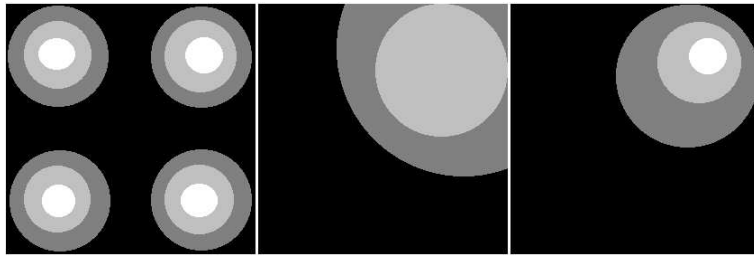


Figure 4: *Example responses when robot is in top-right of area. Left) Ambiguous visual recognition of a location by Visual Place Cells. Middle) Location probable following Idiothetic View Cells (with low precision). Right) Location concluded by Hippocampal Place Cells, without aliases and higher accuracy. White denotes the highest recognition/probability/'belief' of being at a certain location.*

This figure shows at the left the accurate but ambiguous recognition of the location by a layer of visual place cells. The robot is in a corner of the area, but these four are indistinguishable. The middle picture shows the location probable from employing path integration. The accuracy is lower than from the visual place cells, but is unambiguous. The Hippocampal place cells, right picture, combines this information to conclude unambiguously with higher accuracy the actual location of the robot.

2.5 Preprocessing snapshots with Gabor filters

The view cells do not recognize a location directly, but use values resulting from a Gabor filter operation. Gabor filters are frequency and orientation selective filters, that have the same behaviour as so-called simple cells in the brain (Daugman, 1980). There are many qualitatively equivalent ways to model these (Arleo *et al.*, 2004, J. Yang *et al.*, 2003, B.S. Manjunath W.Y. Ma, 1996, D.J.Field, 1987 and various Internet websites). In its easiest form, a one dimensional Gabor filter can be viewed as a cosine in a Gauss normal curve envelope. Arleo *et al.* chose to use one of the more complex ones, an image showing the complex-real part of the Gabor filter is shown below:

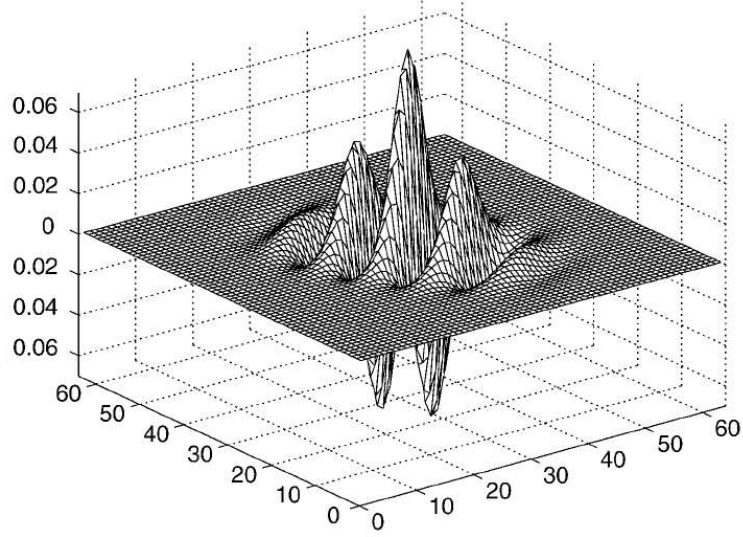


Figure 5: *The real part of the complex sinusoidal wave of a Gabor filter used by Arleo et al.(2004).*

Arleo then also modified this Gabor filter in the Log-Polar frequency plane. This modification reduces overlap in the low-frequency side of the Gabor filter recognition. For this a logarithmic transformation was used.

In this research no complex waves nor Log-Polar modifications were used. Instead the more intuitive form is used, derived from the one-dimensional form:

$$G(D) = \cos(f \cdot D) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{D^2}{\sigma^2}} \quad (1)$$

In which D is the distance, f is the frequency of the filter. σ determines the speed at which the Gabor normal curve reaches zero. The first part denotes the frequency-dependent cosine wave, the latter the Gauss normal curve.

Transforming it to 2 dimensions, makes it possible for the filter being selective for a certain angle ϕ :

$$G(x, y) = \cos(f_x \cdot x + f_y \cdot y) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{D^2}{\sigma^2}} \quad (2)$$

$$f_x = f \cdot \cos(\phi) \quad (3)$$

$$f_y = f \cdot \sin(\phi) \quad (4)$$

f_x and f_y are the separated frequencies in horizontal and vertical direction respectively, calculated from the frequency f and angle ϕ . Again, D is the distance from $(0,0)$ and σ determines the speed at which the Gabor normal curve reaches zero. Note that the second part of equation 2, the normal Gauss curve, remains the same.

The maximum, G_{max} , of both equations 1 and 2 are at $D = 0$:

$$V_{t+\Delta t} = V_t + \frac{\Delta t}{\tau}(-V_t + I) \quad (5)$$

Figure 6 below, obtained by plotting equation 2, shows the qualitative resemblance with figure 5:

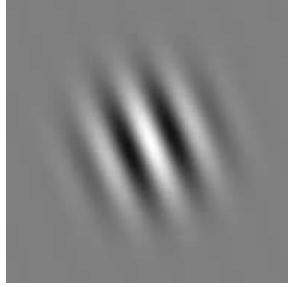


Figure 6: *The Gabor filter function used. Black denotes a highly negative value, white a high positive and grey equals a value of zero.*

Before image processing, the image is histogram equalized (see e.g. Pratt, 1978) and rescaled from (pixel)values of [0,255] to values of [-1,1].

Instead of performing image processing on the entire image, Arleo and co-workers chose to use only certain locations. Like the human retina, most information is extracted from the middle of the image. These 31 image processing locations are show in figure 7. On each of these locations, 3 different-sized Gabor filters were applied, for 8 different angles. These 744 values are used by the view cells.

2.6 View cell recognition

When a view cell is created to remember a certain image, it is initialized on the 744 retinotopically Gabor filter values. When it is asked to recognize a (new) set of values, then first its input I is calculated as follows:

$$I = \frac{1}{N} \sum |r_G - w| \quad (6)$$

In which N is the amount of values it needs to process (in this case 744), r_G is a new value obtained from retinotopically Gabor filter processing and w the value (of an earlier r_G) it is initialized on. Note that for a perfect recognition ($r_G = w$ for all 744 values), I equals 0. This input changes a view cell's membrane potential in the following way:

$$V_{t+\Delta t} = V_t + \frac{\Delta t}{\tau}(-V_t + I) \quad (7)$$

in which V denotes the membrane potential, I the input, t time and τ a time constant.

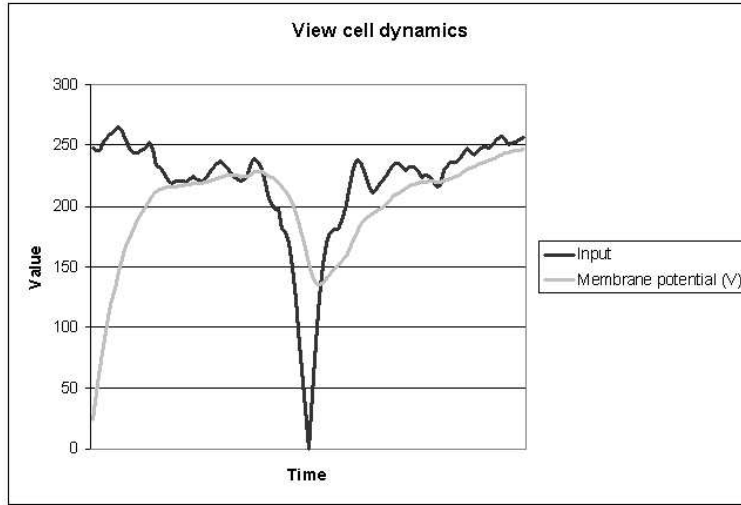


Figure 7: *View cell dynamics.* Plotted in time is a certain input and its resulting membrane potential. An input of 0 means a perfect recognition of the image initialized on. Note that if input is low, membrane potential starts to approach to zero.

After having changed its membrane potential, a view cell responds according to

$$V_{t+\Delta t} = V_t + \frac{\Delta t}{\tau}(-V_t + I) \quad (8)$$

Below is a plot (figure 8) illustrating this response behaviour:

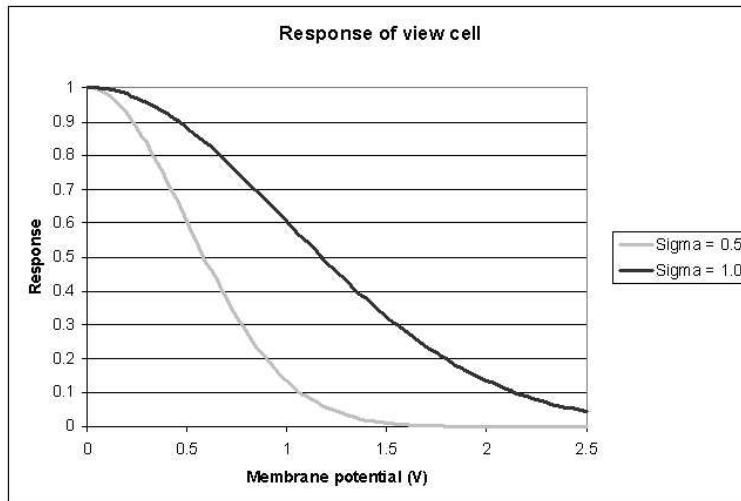


Figure 8: *Response of a view cell.* A response is dependent on the view cell's membrane potential and its sigma. A response of 1 is the consequence of a low membrane potential. A low membrane potential can be reached by perfect recognition, i.e. input is zero.

Note that for multiple perfect recognition ($r_G = w$ for all 744 values), r will approach 1, due to V approaching 0.

2.7 This research

The focus of this research is to set up the same framework as Arleo and co-workers. However, the environment at the institute might be different than at Arleo's. He uses the same standard robot as at the institute, a Kephra miniature robot (distributed by K-Team S.A.).¹ Although not mentioning it explicitly, it is assumed that the same standard Kephra camera is used.

At the institute, many software, written in C++, was already available to control the hardware of the robot and camera. Much of this hardware control was also transformed to use Parallel Virtual Machine (PVM). PVM enables multiple programs to run parallel on different computers and is available freely.² When a program uses PVM, it means it behaves like a server, sending and receiving messages. At the institute, for both the physical robot and camera, a server was already programmed. Goal is to use these institute's server, improving their functionality where needed and perhaps creating more.

3 Research questions

Can the framework of Arleo be transferred to the Kephra system using PVM at the institute? Do some parameters need to be adopted to the institute's setup? What are the parameters Arleo used for the Gabor filters? What is the effect of view cell recognition with and without first histogram equalizing the input image? How does a view cell perform in distinguishing the location it is initialized to on from others? How does a view cell perform in recognizing its own image after a translation?

4 Materials and methods

4.1 Obtaining the retinotopic grid and Gabor filter parameters

In the article (Arleo *et al.*, 2004) a camera snapshot (fig. 9) was depicted, in which the retinotopic grid was shown and also the supports of a view filters. The image was stated to have a resolution of 768 times 576 pixels. The 31 positions on the retinotopic grids were measured and their positions calculated. The circles indicated the filter support were measured and their diameter calculated. The 'support' of a filter has been interpreted as the distance of which the Gauss envelope of the Gabor filter has such a low value, that a pixel won't have effect on the processing outcome.

¹www.k-team.com

²http://www.csm.ornl.gov/pvm/pvm_home.html

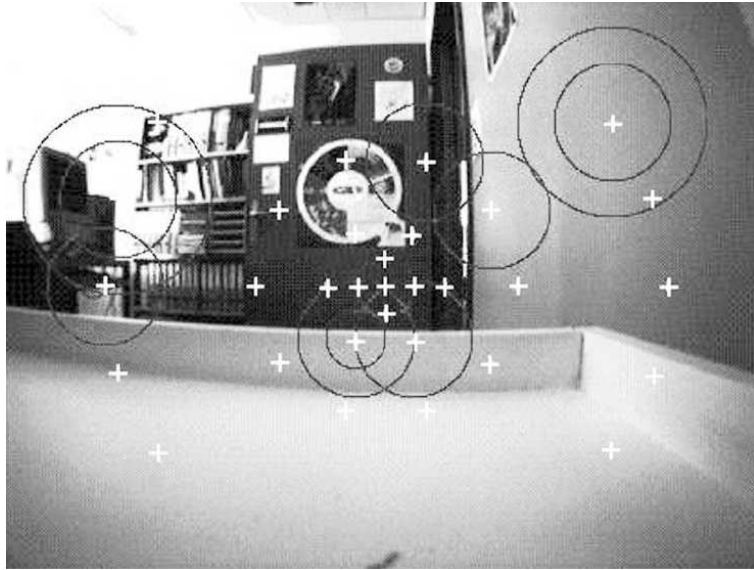


Figure 9: Camera snapshot (768 times 576 pixels), in which the retinotopic grid is shown with white crosses and the support of some filters in dark-gray circles. From Arleo et al.(2004).

4.2 Image processing

For image processing, OpenCV was used. It is an Open Source Computer Vision library for C++.³ However, the exact functioning of the image convolution can not be found (only developers are allowed to view the algorithm's code).

Instead of performing feature detection on entire images, this was necessary at the 31 specific positions only. OpenCV cannot be told to process only one pixel. It can be told to process a ROI (Region of Interest), but setting a ROI to only 1 pixel 31 times did not seem a good idea. To have control, an algorithm has been written being able to perform convolution on any pixel. The self-written convolution has for a certain pixels at (x, y) the result $F(x, y)$ using

$$F(x, y) = \sum_{x'=x-\frac{1}{2}size}^{x'+\frac{1}{2}size} \sum_{y'=y-\frac{1}{2}size}^{y'+\frac{1}{2}size} f(x' - x + \frac{1}{2}.size, y' - y + \frac{1}{2}.size)p(x', y') \quad (9)$$

in which $f(x', y')$ equals the value in the filter value matrix, $p(x', y')$ the converted pixel value of the original image to $[-1, 1]$.

The image processing algorithm has been tested on a testing image and on a 'real' image (using the project 'ImageFilterTest'). OpenCV's convolution is assumed to be a correct algorithm. If the self-written convolution detects the same features as OpenCV's, the self-written algorithm is assumed to be correct. Also, the effect of first histogram equalizing the image has been researched by processing an image with and without first equalizing the image.

³<http://www.Intel.com/research/mrl/research/opencv/>

4.3 View cell recognition

When the standard Kephera camera takes two images of the same location, these are not exactly identical, e.g. due to subtle lighting changes and/or camera adaptation to these. To test this, a dataset has been created by letting the robot take 50 pictures of the same location in 2 different settings in either light or dark conditions. One setting had many objects in it, in the other the camera only photographed a uniformly white paper (see figure 10).



Figure 10: *The 4 types of images used. All images had size 768 times 576 pixels. Note that the images taken in the dark (left) have increased noise.*

In 16 trials, a view cell has been initialized on an image of any of these four settings and then faced with 50 images of the same or the other four settings. This should show, that a view cell recognizes the location it is initialized on better than on other locations. These four different settings are a very stringent test. Only the lively picture in light condition is hypothesized to be recognized successfully. These tests are also done with/without first histogram equalizing the image before retinotopically Gabor processing it.

From looking at the bottom-left image of figure 10, it can be concluded that the camera itself performs some adaptation, because this image was taken in pitch-black conditions. It is hypothesized that the camera does most optimization in the dark. Also it is hypothesized that the camera images taken from a uniform white surface differ more from each other than from a lively scene.

Another program makes it possible to test live camera images, but can also request a testing set. This testing set contains reproducible artificial values. When a view cell is initialized on this testing set and receives it again, its recognition is always 100%. Taking live images can show the qualitative view cell behaviour, but unless its images are saved and thus making the experiment reproducible, no comparisons can be made. But it is great for demonstration purposes.

4.4 View cell recognition after translation

To measure the recognition of a slight translated image, a view cell is initialized on the retinotopic Gabor values of one of the four standard images. Then, the same image is loaded, but first translated horizontally, before retinotopic Gabor processing it. All horizontal translations between -57 to 57 pixels were performed. The absolute maximal translation of 57 was chosen, because else the support of the filter would go beyond the image's surface.

4.5 Software implementation

Programming such a complex network is a tedious task. Therefore, all programs needed to be thoroughly tested, before serving as a building block. The possibility to test already-programmed classes was low, e.g. only by entering a single command in the start-up command line. Therefore, all existing programs were thoroughly rewritten as also their testing programs.

One important way to perform testing, is writing a program that performs a functionality on a user's key press. This was not implement and will be. Also, the amount of direct visual feedback and output will be increased.

5 Results

5.1 Obtaining the Gabor filter parameters

The image from the article had width 8.4 cm. (768 pixels) and height 6.3 cm (576 pixels). The 'support' of a filter has been interpreted as the distance of which the Gauss envelope of the Gabor filter has such a low value, that a pixel won't have effect on the processing outcome.

The boundary value of 0.5/255 of the maximum was chosen, because its multiplication with the maximal pixel color value (of 255) after rounding off will result in 0, due to the formula 9. However, G_{max} is dependent on σ (see formula 5). To make all supports equal, instead the algorithm below is chosen:

$$G(D) = \frac{0.5}{255} G_{max} \quad (10)$$

Filling in $G(D)$ and G_{max} from equations 2 and 5 yields

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{D^2}{\sigma^2}} = \frac{0.5}{255} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \quad (11)$$

resulting in

$$D = \sqrt{-\ln\left(\frac{0.5}{255} \cdot 2\pi\sigma^2\right)} \quad (12)$$

The measuring and the calculation resulted in the following table:

Filter	Size(mm)	Support(pixels)	Size(pixels)	Wavelength(pixels)	Sigma
Small	6.7	32	64x64	8	9.1
Middle	13.0	64	128x128	16	18.2
Large	21.0	96	192x192	32	27.2

Table 1: *Parameters of the three Gabor filters used in this research.*

For checking these parameters, a surface plot was made of these filter values, resulting in the figures below:



Figure 11: *the smallest-size Gabor filter used. Left: standardized surface plot. Middle: standardized vertical cut. Right: standardized horizontal cut. Parameters: size: 64x64 pixels, wavelength: 8 pixels, sigma: 9.1. Maximum: 0.0438.*

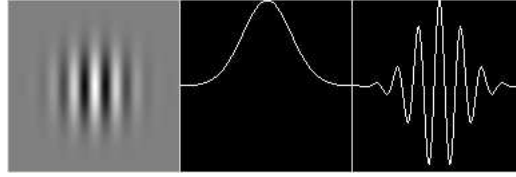


Figure 12: *the middle-size Gabor filter used. Left: standardized surface plot. Middle: standardized vertical cut. Right: standardized horizontal cut. Parameters: size: 128x128 pixels, wavelength: 16 pixels, sigma: 18.1. Maximum: 0.0220.*

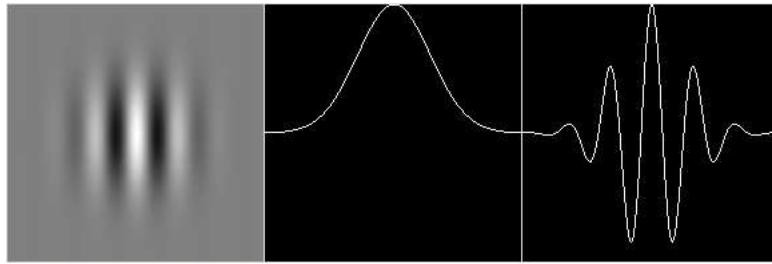


Figure 13: *the largest-size Gabor filter used. Left: standardized surface plot. Middle: standardized vertical cut. Right: standardized horizontal cut. Parameters: size: 192x192 pixels, wavelength: 32 pixels, sigma: 27.2. Maximum: 0.0147.*

As can be seen, the value of $G(D)$ at the edges is indeed zero.

5.2 Image processing

5.2.1 Difference between OpenCV and self-written convolution

The self-written convolution performs quantitatively equal as OpenCV in detecting the right features. An artificial image was created to sharply detect both convolution's performances. It consists of alternate-colored concentric rings, each being 8 pixels broad. Therefore, it has a wavelength of 16 pixels and due to its rotation-symmetric nature, the angle-specific nature of Gabor filtering should be seen clearly. Figure 14 and 15 show this testing image, and the result of a convolution with three different filters. The parameters are the same as in table 1, which are of the three Gabor filters used throughout this research.

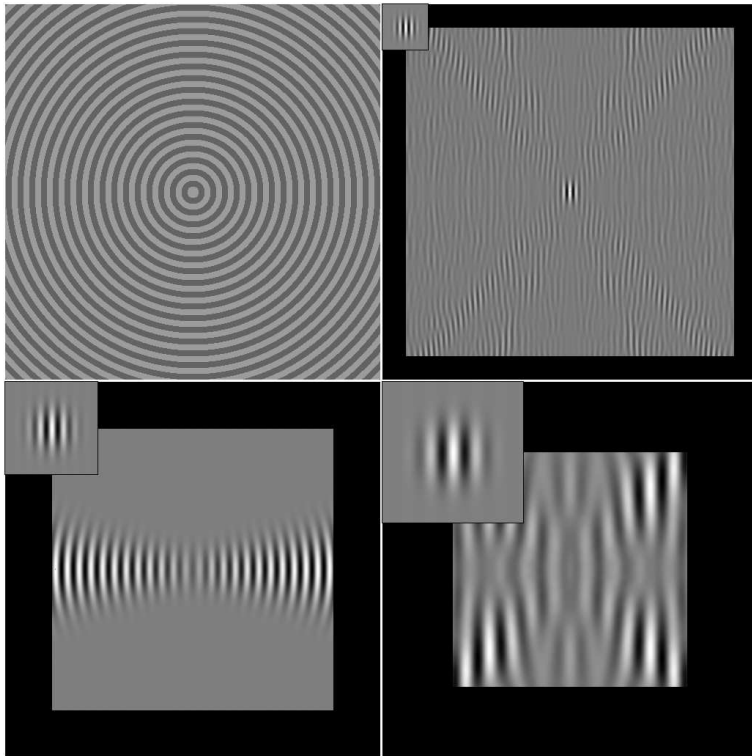


Figure 14: *The result of the self-written convolution on a testing image. Top row: Left) original image with size 512x512 pixels. Right) image after convolution with the smallest Gabor filter. Bottom row: Left) image after convolution with the middle Gabor filter. Right) image after convolution with the largest Gabor filter. Note the black edges of the resulting images. See Table 1 for the exact Gabor filter parameters. The three insets shows the filter used.*

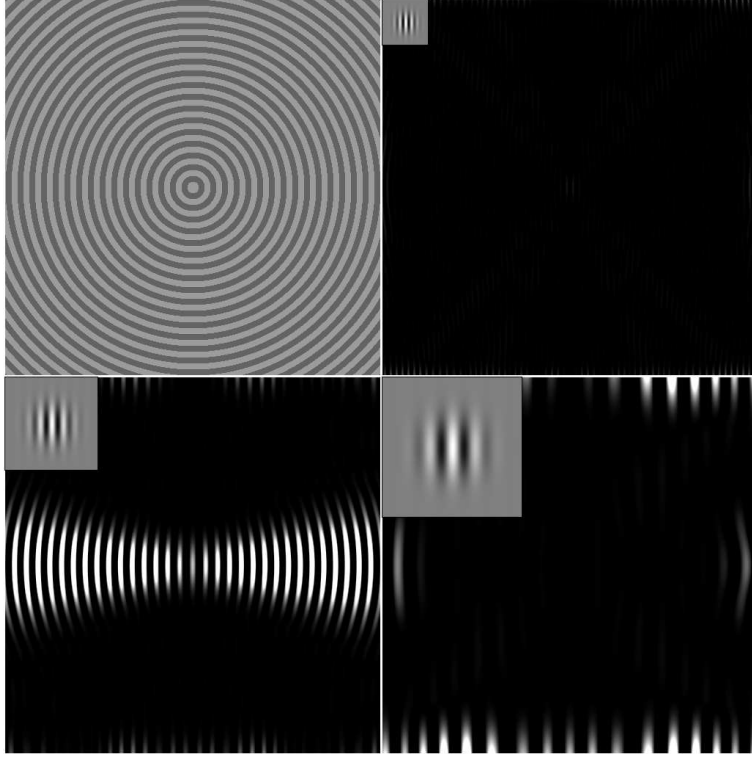


Figure 15: *The result of the OpenCV convolution on a testing image. Top row: Left) original image with size 512x512 pixels. Right) image after convolution with the smallest Gabor filter. Bottom row: Left) image after convolution with the middle Gabor filter. Right) image after convolution with the largest Gabor filter. Note the black edges of the resulting images. See Table 1 for the exact Gabor filter parameters. The three insets shows the filter used.*

Expected is that the middle Gabor filter, with the same wavelength as the repetition in the image, yields the sharpest recognition in the recognition angle of the filter. Both self-written and OpenCV clearly have this as a result, but on different scales. The self-written convolution has both negative and positive values as a filter operation result, which are drawn black and white respectively. A detection of zero is therefore drawn as grey, which is the dominant color. The OpenCV's convolution clearly draws a black pixel for a recognition of zero. It is hypothesized that OpenCV only draws white when a recognition is positive, which is more probable than that it takes the absolute value, because the result of the middle filter has as much white rings as the self-written convolution (instead of twice as much). Also, in the OpenCV's convolution, the highest recognition is not scaled as the color white. It is hypothesized that this convolution scales to maximal possible recognition. The maximal possible result of a filter can be calculated from summing all absolute values of the matrix:

$$F_{\max} = \sum_{x=0}^{x=size} \sum_{y=0}^{y=size} |f(x, y)| \quad (13)$$

Also, the self-written convolution always gives black borders on the resulting image. It does so when it cannot process that location, due to the filter needing pixel values from beyond the

image. Unlike OpenCV, the self-written convolution does not employ any processing to solve this. This does have no effect on the rest of the research at all, because for a view cell, an image is only processed retinotopically. These 31 retinal points do not lie at the edges of the images, therefore no need exists for some special edge processing.

5.2.2 The effect of histogram equalization on feature detection

Histogram equalization seems to increase Gabor filter responses in the OpenCV's convolution only. In the self-written convolution, peaks sometimes seem to fade away (as in fig 16). This is probably because the self-written convolution scales the image produced to pixel values of [0,255]. In an image with low feature detection, multiple small peaks get amplified to [0,255]. When due to histogram equalization, which is a non-linear image processing algorithm, one of these locations increases strongly, it will outrun the other peaks. As a result that the other peaks seem to vanish, but in fact are just adapted to the scale of the highest peak. In figure 16 this effect can be seen at the top right of the image: without histogram equalization, 5 peaks are seen. After histogram equalization, 4 peaks get less strong. This is probably because the only peak that remained, has become amplified stronger than the rest.



Figure 16: *The difference between OpenCV's and the self-written convolution, with and without first histogram equalizing the initial picture. Top row: Left) original image. Mid) image after OpenCV's convolution. Right) image after self-written convolution. Bottom row: Left) original image after histogram equalization. Mid) image after OpenCV's convolution Right) image after self-written convolution. The filter used was the smallest Gabor. size 32×32 , wavelength 8 pixels, sigma 9.1.*

5.3 View cell recognition

Expected is that the images of the same scenario as the one initialized on, are best recognized, i.e. to have the lowest input value for the view cell. View cell recognition was measured on either images that were or were not histogram equalized. The recognitions were different as can be concluded from fig 17 and 18. For the recognition on histogram equalized pictures (figure 17), the average input values of the scenery initialized one, were always lowest. Also, the one-but-lowest input value is always from the same scenery, but in the other lighting condition. The distinction between low input on the scenery initialized on and high input on the other settings is greatest on the lively image in lighted conditions. But also this recognition of the lively environment in the dark was also very good. Astonishingly, even a view cell initialized on the plain white paper can distinguish its own scenery, albeit that the distinction between dark and light cannot be made that clearly. That is, on average the input of the scenery initialized on is lower, but when looking at the standard deviation bars, some inputs of the other environment are lower than this average.

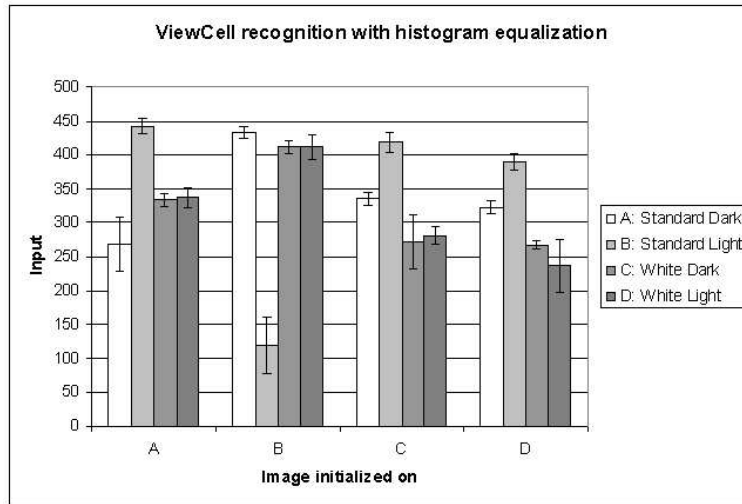


Figure 17: View cell recognition with first using histogram equalization. Under each experiment, a view cell was initialized on one of the four environments and then the response of 50 different pictures from the same or other environment is measured. A low value of input (see formula 6), denotes a high recognition. Bars denote standard deviation (N=50).

This again shows that the effect of histogram equalization is as expected: the variability of the input is higher then without. When histogram equalization, recognition in the dark is very complicated, because images can get deformed extensively. When looking at view cell recognition on images that were not histogram equalized, the variability is much lower. Removing histogram equalization makes the view cells do much better. A view cell initialized on a white paper in the dark can tell the difference between the same paper in light and dark (see figure 18).

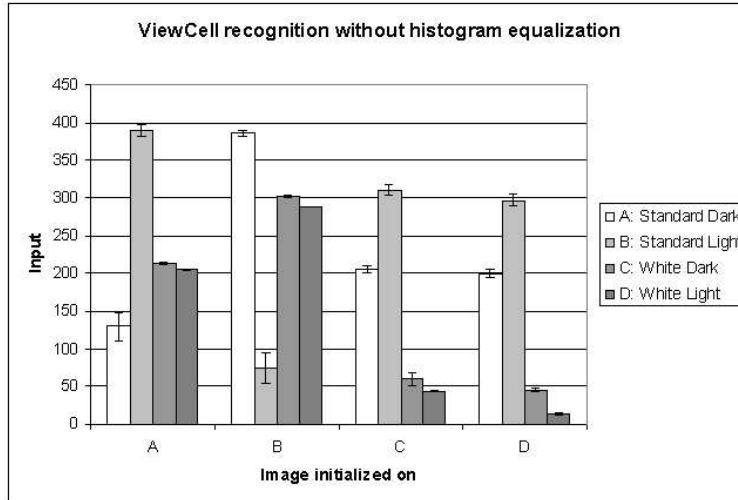


Figure 18: View cell recognition without first using histogram equalization. Under each experiment, a View cell was initialized on one of the four environments and then the response of 50 different pictures from the same or other environment is measured. A low value of input (see formula 6), denotes a high recognition. Bars denote standard deviation ($N=50$). Note that the standard deviation is sometimes too small for a bar to be drawn.

Also, without histogram equalization, recognition is always statistically improved, i.e. the chance an other scenery's input gets below the input of the scenery initialized on is lower. There is one exception: a view cell initialized on a white paper in the light, recognizes the same paper in the dark better. The irony is, that it will consequently give the scenery it is initialized on a higher value then the same scenery in dark, and these two input ranges do nearly not overlap.

5.4 View cell recognition after translation

When translating an image, recognition should be maximal at a translation of 0, i.e. the image is the same as initialized on, resulting in an input value of 0. For a self-recognition to be a bit flexible, recognition should deteriorate after translation of a single pixel. Instead, the recognition should only gradually decrease. The effect of translation on recognition has been tested with and without first histogram equalizing an image (figure 19 and 20). Under all four circumstances, recognition seems to be able to cope with small translations. Always, recognition of small translations yield higher recognition then for high translations. This trend continues until a translation of 24 pixels, in which recognition gets better. From 36 pixels on, recognition decreases again. Why this pattern emerges is unknown, but it is clearest in the lively photo in light. There is a big difference in effect between with and without first histogram equalizing the image on the scenery consisting of only a white paper. When histogram equalization this scenery, recognition decreases more extensively. It can also be seen, that translating the white paper in light condition, has least effect, especially when this image is not histogram equalized. This might give rise to the hypothesis that this scenario has the least visual cues to rely on.

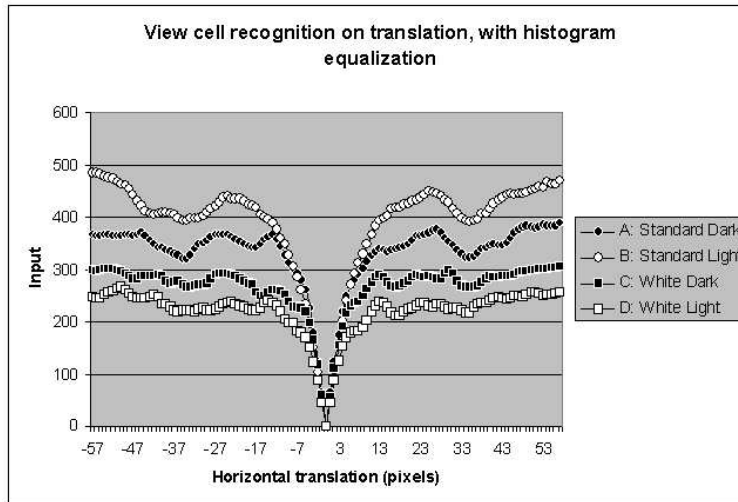


Figure 19: View cell recognition on translation, with first performing histogram equalization. A view cell is initialized on a histogram equalized image (with translation 0), then the recognition is measured for translations from -57 to +57 pixels. A high recognition denotes a low value for input. In this image this can be seen clearly at the translation of 0, where the initialized view cell sees exactly the image it is initialized on.

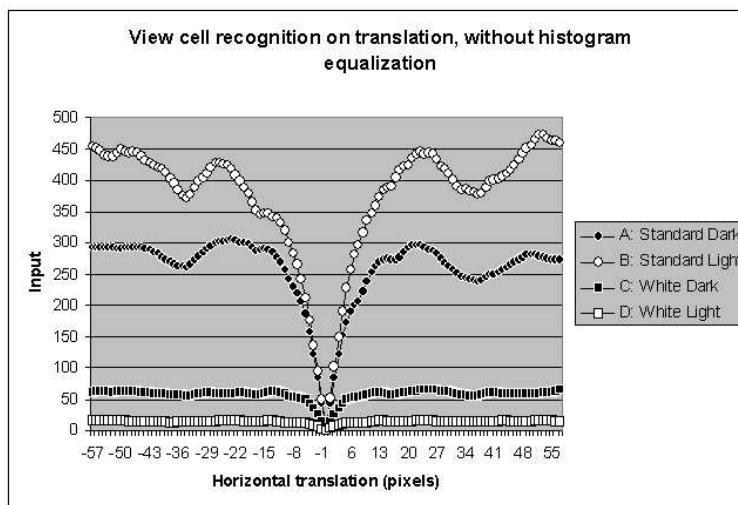


Figure 20: View cell recognition on translation, without first performing histogram equalization. A view cell is initialized on a histogram equalized image (with translation 0), then the recognition is measured for translations from -57 to +57 pixels. A high recognition denotes a low value for input. In this image this can be seen clearly at the translation of 0, where the initialized view cell sees exactly the image it is initialized on.

5.5 Software implementation

All programs used were thoroughly looked at, largely rewritten, tested and often debugged. For all C++ classes a testing program has been written with a very high level of control, which is

control e.g. by key press.

All functionality of the class `KRobotLib`, controlling the physical robot, has been tested using `KRobotLibTest`. This program has put all the robot's functionality somewhere on the keyboard. It is now possible to drive the robot, use its gripper and see the infrared sensor values (used for obstacle detection) on the screen. As gateway for the user input to the program, a `CvTextUnit` was developed (see below). To view the infrared sensors on the screen, `CvPlotSensorsUnit` was written (see below).

The class `KRobotLib` has a PVM equivalent called `KRobotServer`. This server has been greatly improved. When testing it, using `KRobotServerTest`, also here all functionality can be tested on a key press. The amount of different messages that can be sent to it has doubled (for a Kephera robot). Although the debugging of programs using parallel processing is more difficult than in non-parallel programs, there are no difference in robot control when using either the direct `KRobotLib` or the `KRobotServer`. Because debugging parallel processes is hard, many debugging functionality has been added, mostly uncommenting a single line resulting in a (more helpful) error message. For user input, the self-made `CvTextUnit` was used.

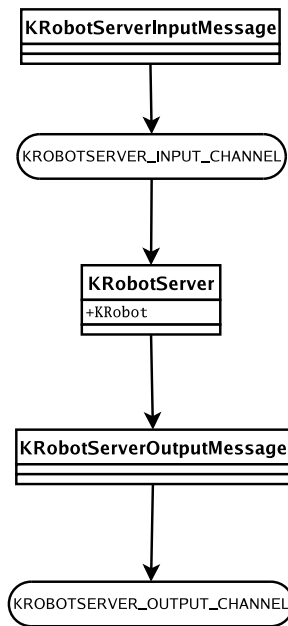


Figure 21: Overview of `KRobotServer`. It has a single input message type `KRobotServerInputMessage`. It reads from channel `KROBOTSERVER_INPUT_CHANNEL`. `KRobotServer` manages a `KRobot` class. It has a single output message type `KRobotServerOutputMessage`. This message is only sent to `KROBOTSERVER_OUTPUT_CHANNEL`.

The class for controlling the camera, `KoCameraLib`, was written very sloppy. Restyling the code resulted in solving a few memory leaks and also making it more exception-safe. `KoCameraLib` was given more functionality in showing the image on screen autonomously, storing the image in memory, in resizing the capture size and/or color channels and also the option to directly save

the grabbed image. KoCameraLib can be tested using KoCameraLib, easily showing all above-described functionality. The ability of KoCameraLib to show its image on screen autonomously was reached using the self-written CvImageUnit (see below). Again, user input was heard using CvTextUnit.

The PVM equivalent of KoCameraLib is KImageServer. This server has had the greatest improvement. At the start, it could only grab an image on request and send it to another process. Now it can display anything grabbed or processed on the screen directly. The amount of different messages it can handle has tripled, making it very lightweight to handle. KImageServer has also become the PVM server for the ImageFilter class (described below), enabling it to do image processing operations. It can directly show any processed image and also an image of the filter ImageFilter is initialized with. To let PVM communicate with the ImageFilter contained in KImageServer, a new type of PVM message was designed. The most important function of this message (called KImageServerInputFilterMessage) is to contain the filter used for image processing. The KImageServer can also use its ImageFilter to do retinotopic processing, sending 31 values obtained from image processing using any filter. These values are sent to a newly-designed server, called ViewCellServer (see below), using a new type of message (NeuralNetServerInputMessage). This new message type can contain any amount of values, therefore rendering it very flexible. Because Arleo and co-workers used 24 different filters, KImageServer also got to manage a class called ImageFilterManager (see below), containing all 24 filters. KImageServer can be requested to process an image retinotopically with these Gabor filters and then sends the resulting 744 values, using the flexible NeuralNetServerInputMessage, to the ViewCellServer. Additionally, KImageServer is also used to process files loaded from the hard-disc. This makes it possible to perform all this functionality on the same dataset. For this server to reach this high amount of functionality it was not only practical but crucial to increase the debugging facilities in it. As in the KRobotServer, only uncommenting a single line of code and recompiling results in a program that mostly gives a clear error message, instead of just ending. For visualization of the original image, processed image and filter, the self-written CvImageUnit was used.

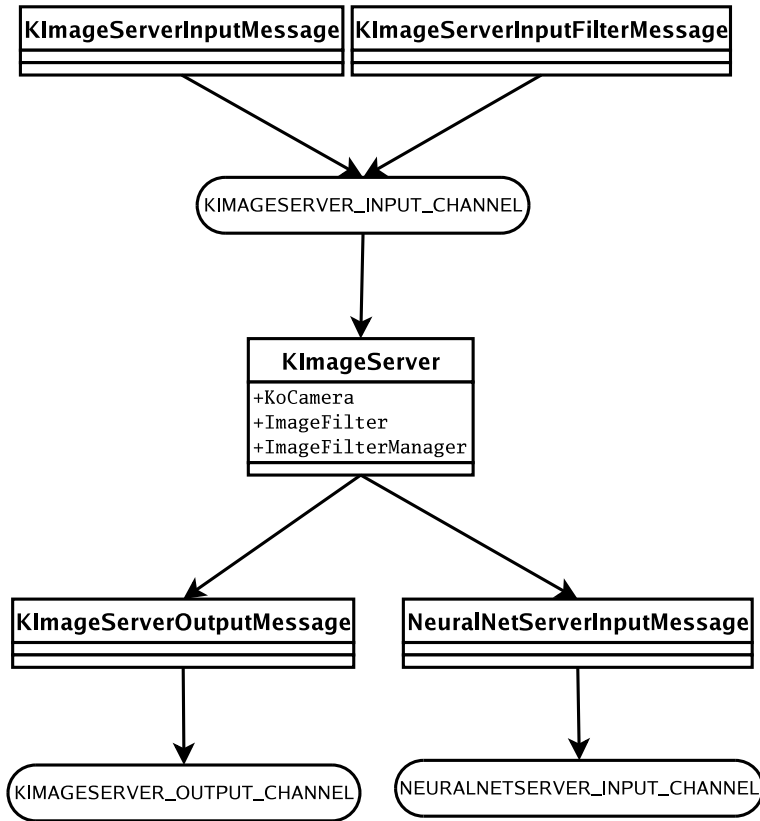


Figure 22: Overview of *KImageServer*. It has a two input message type *KImageServerInputMessage* and *KImageServerInputFilterMessage*. It reads from channel *KIMAGESERVER_INPUT_CHANNEL*. *KImageServer* manages a *KoCamera*, *ImageFilter* and *ImageFilterManager* class. It has a two output message types *KImageServerOutputMessage* (for sending either unprocessed image from the *KoCamera* or processed images from the *ImageFilter*) and *NeuralNetServerInputMessage* (for sending retinotopic value from either *ImageFilter* or *ImageFilterManager*). The *KImageServerOutputMessage* is only sent to *KIMAGESERVER_OUTPUT_CHANNEL*. The *NeuralNetServerInputChannel* is only sent to *NEURALNETSERVER_INPUT_CHANNEL*.

The class used for image processing, *ImageFilter*, needed to be developed from scratch. It can initialize its filter matrix in many ways. One way is to call a Gabor filter initialization, after which the *ImageFilter* calculates the filter matrix from these parameters. The *ImageFilter* can process images with both OpenCV's convolution and the self-written convolution, called the 'MatrixMultiplication' convolution. Because the *ImageFilter* also needs to run in PVM environment, it was made compatible with it by adding functions that can be logged in this environment. This way also in PVM errors can be easily and quickly spotted. The *ImageFilter* was tested using *ImageFilterTest*. It enables the user to select an image filter operation from the command line and view the result from an image that is both processed by OpenCV and the self-written convolution.

Building on the success of the self-written *ImageFilter*, *ImageFilterManager* was developed. This class manages the 24 Gabor filters used by Arleo, using the parameters concluded from this

research. Because ImageFilter was debugged so thoroughly, there has not been much need for adding PVM debugging functionality. ImageFilterManager is tested by ImageFilterManagerTest. This program performs the 24 Gabor filter processes on any image and then saves the results and filters used to disc.

A C++ analogy of a view cell was created, conveniently called ViewCell. Because the property of making its response dependent on its membrane potential, instead of its input, was found interesting for investigation, the ViewCell can be initialized to do one of them both. To make it function in PVM environment it also has entries for logging and debugging.

When a retinotopic filter operation is done by KImageServer, these values are send to the ViewCellServer. The body of the server is a single ViewCell. The server has to be initialized by a message, where after the ViewCell itself can get initialized. The messages it receives are called NeuralNetServerInputMessages with eye on expansion: a server of multiple ViewCells will be called NeuralNetServer. But with both servers having the same message format, there was no need to rename the input message for the ViewCellServer. Although the ViewCellServer manages only a single ViewCell, it has many output windows. It plots the actual incoming input values and all viewcell's input (eq. 6), membrane potential (eq. 7) and response (eq. 8). It does so using CvBarChartUnits and CvLineChartUnits (see below). Because the ViewCellServer always operates in a PVM environment with at least two other processes, the necessity of strong debugging possibilities is strong, using the same techniques as KRobotServer and KImageServer

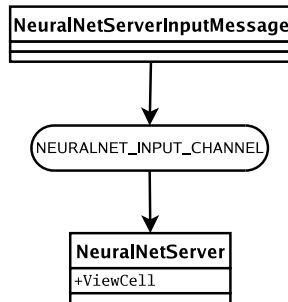


Figure 23: Overview of NeuralNetServer. It has a single input message type NeuralNetServerInputMessage. It reads from channel NEURALNET_INPUT_CHANNEL. NeuralNetServer manages a ViewCell class. It has no output messages or channels.

The ViewCellServer is tested by actually two programs. The first is ViewCellServerTest. It is a PVM program that also needs KImageServer and ViewCellServer running. It initializes both these servers. Then the self-written CvTextUnit shows the user its options from pressing a key. The user can select in which way the ViewCell in the ViewCellServer gets initialized. It can be initialized directly by sending one of non-random and random input values, the first values being always the same. In this way, it is possible to see that if the ViewCellServer is initialized on the non-random values, it will recognize these totally, because on purpose no noise is put into these values. This way, it can be seen that a perfect recognition denotes a value of input I of 0, membrane potential going to 0 and response climbing to 1. Instead of sending these values to the

NeuralNetServer directly, which was needed for debugging purposes, it is also possible to request KImageServer to send these either non-random or random data. Although the message pathway is then extended by one server in between, the same can be concluded. Also, it is possible to request KImageServer to grab an image from the camera, retinotopically process it using the ImageFilter and send these 31 values to the ViewCellServer. This way, view cell recognition can be measured live, for any desired filter. Additionally, it is possible to request the KImageServer to grab an image, process it retinotopically using the same 24 filters as Arleo and send these 744 values to ViewCellServer. This way, view cell recognition using the method of Arleo can be viewed live.

The ViewCellServerTest is also tested using ViewCellTest. This program only has user input via configuration files. This program also functions in PVM environment and uses both KImageServer and ViewCellServer. It initializes both KImageServer and ViewCellServer and then sends a message to KImageServer to process image files stored on the hard-disc. This way it is possible to compare results from exactly the same dataset.

GaborFilter was one of the earliest projects. It makes it possible to manipulate Gabor filter parameters by keyboard. A CvTextUnit is used for receiving the keypresses as well as showing the user all possibilities. Multiple kinds of Gabor filters can be viewed on screen. By key press, the Gabor filter selected is used to process an image, showing the result on screen using a CvImageUnit.

All graphical units have been developed from scratch. Every one has a very distinct function and uses the OpenCV library. Every graphical unit has been made flexible accessible in PVM environments to easy debugging, and all units have a testing program, showing their functionality on keypress.

CvWindowUnit is the simplest, containing only information to show a window, that needs an extern image to have something to show. Although being the simplest, it uses all window functionality OpenCV has to offer. Providing an image to a CvWindowUnit is exactly what CvImageUnit likes to do. The CvImageUnit manages an image and offers many image operation functionality. It can do all low-level image processing like resizing and changing its color channels. Also, it can save and load its images. Because CvImageUnit is used heavily in PVM for all kinds of visualization, it has very high PVM logging functionality.

The CvTextUnit is some kind of CvImageUnit, because it also handles an image, but does not provide any low-level image operation of its own. CvTextUnit is just a replacement for a console window. This was needed very heavily, because in PVM, there are no functional console windows possible. It is used for displaying text and receiving keyboard input.

For plotting data, CvLineChartUnit, CvBarChartUnit and CvPlotSensorsUnit have been developed, plotting a line chart, a bar chart and the infrared sensor values of the Kephera robot respectively. The latter class having only simple functionality, the first two excel in being flexible with data. Therefore, these are used extensively by ViewCellServer. Because many things can go wrong in PVM environments with also many data, especially CvBarChartUnit is made PVM logging-friendly.

6 Conclusion

A founded approximation of the Gabor filter parameters used has been made.

The ImageFilter class embodying the self-written convolution functions very good. Its convolution seems qualitatively to have the same expert behaviour as the OpenCV's algorithm, but additionally enables to do retinotopic processing also.

The effect of histogram equalization before image processing reduces the self-recognition. When a view cell was faced with 50 times a slightly different image of the same environment, a higher input value was measured. This is due to that fact that when taking two natural image, they are not totally identical. Histogram equalization only decreases the likeliness of two such photos, because it tends to enlarge the difference. It seems likely that a simple way to improve the framework of Arleo is to remove this histogram equalization.

However, because images get more contrast, histogram equalization decreases the image recognition after translation. This could have the benefit, that it could reduce translational aliasing, e.g. a false location recognition by translation.

If a good self-recognition is important (figure 18) or that it is important to have little translational aliasing (figure 19) determines the outcome whether histogram equalization is beneficial for the architecture. However, the first is important for sure, whereas the reduction of aliasing due to translational self-recognition might be a non-existing problem. To be sure, the interplay of multiple neurons must be analyzed while the robot is navigating.

Using retinotopic Gabor filtering seems to work very good for image recognition. A view cell cannot only distinguish between locations, but also between light and dark. And even in the near-dark it can recognize the location correctly. It saves a lot of time to do only retinotopic Gabor filtering, even for 24 small to large filters. It might be interesting to try other filters to compare because of the flexibility of ImageFilter it can be easily used to compare performance of other filter types, e.g. the modified Gabor filter type.

Measuring the input of view cells is a good way to measure its recognition of an image. In Arleo's framework however, they have an indirect response. Instead of directly responding to new inputs like equation 6, this response first changes the membrane potential (equation 7), the latter inducing a response (equation 8). The need of this construction does not seem necessary yet, because view cell discrimination is plentiful, but it might play a role when the entire neural network architecture is put downstream of the view cells. It might be needed to stabilize recognition when the robot is in motion, because view cell input is fluctuating even on the same location at a different time. However, perhaps this construction is needed in the setup of Arleo and co-workers, but is obsolete when removing histogram equalization.

7 Discussion

The recognition difference between Gabor filter values obtained by either the self-written and the OpenCV's convolution were not compared. Comparing these is not possible: the first uses both

negative and positive, broken values, whereas the second computes integer (pixel) values [0,255]. However, when the suggested convolution indeed matches OpenCV's, there would be nothing to compare. Then, the ImageFilter would contain both a good convolution function and the ability to perform retinotopic processing.

Visualization of any image is not possible. Only images having both length and height dividable by four can always be shown on screen successfully. Because the filters and images used by Arleo and co-workers (and thus this research) had this properties, no compromise needed to be made at all for this research. However, when performing convolution with the self-written algorithm (in ImageFilter), pay attention to this constraint. OpenCV's own convolution however, does not put constraints on the image filter size.

8 Future work

A more intuitive convolution, that might accidentally match OpenCV's, is suggested to function like:

$$F(x, y) \leq 0 \rightarrow P(x, y) = 0 \quad (14)$$

$$F(x, y) > 0 \rightarrow P(x, y) = 255 \cdot \frac{F(x, y)}{F_{\max}} \quad (15)$$

Where $P(x, y)$ is a pixel color value, $F(x, y)$ is the outcome of a filter operation, F_{\max} is the maximal value the filter operation can yield, $F(x, y)$ the filter value, $P(x, y)$ the converted original image pixel value in range [-1.1].

Implementing this will bring the preferred behaviour of getting nearly-black when a feature a filter selects for is not detected in an image. This way, the resulting image is scaled to the filter's maximum, instead that the resulting image itself is scaled to [0,255].

The implementation of the self-written convolution functions well, but can be made quicker. This can be done by, instead of performing all the multiplications every convolution, a lookup-table is created only once. This lookup-table contains for every filter index, all 256 possible multiplications outcomes. When this lookup-table has been created, a convolution can be done with only summations, probably making it much faster.

The implementation of a single view cell in a single direction has been done. However, Arleo *et al.* implemented that a view cell comprises a panoramic view, each therefore storing 4x744 values. The visual part of the neural net architecture having only a single view cell, the idiotic part has not been implemented at all. Dead-reckoning is not hard to implement, but in the multi-process architecture means that more PVM messages will have to be sent, increasing debugging complexity. Except for the difficulties of programming for a parallel processing environment, implementing the rest of the neural network will not be that difficult, because Arleo *et al.* published all neuron's properties. Implementing the learning of the neural network is not done and no estimation on its difficulty can be made.

A first nice experiment continuing on this research would be an extension of the program `ViewCellServerTest`. In this program, the robot takes images from the camera. The first image initializes the single view cell. The extensions needs to have the robot taking a quasi panoramic view. Then adding the possibility for the robot to move at random through the arena and recording the visual recognition. It would yield an image like figure 2.

Below is given a proposal for the final implementation. Note that `KImageServer`'s function `RequestArleo()`, means that there has to be a temporary cooperation between `KRobotServer` and `KImageServer` in such a way that `KImageServer` can take snapshots to the four wind direction. This cooperation can best be managed by `NeuralNetServerTest`, sending messages to `KRobotServer` and `KImageServer` respectively four times. After or before this takes place the robot's position is requested.

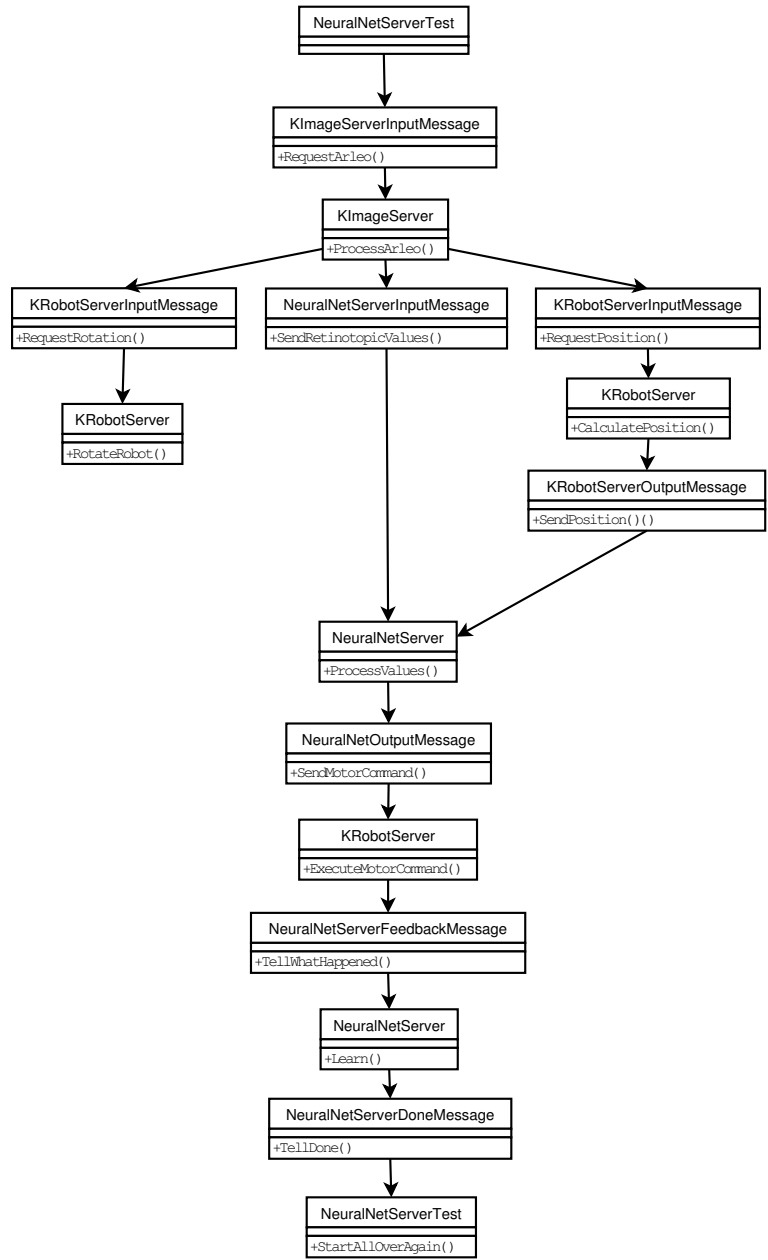


Figure 24: Proposed sketch of the final structure. Note that some classes appear multiple times. Time goes chronologically from up to down.

References

- [1] A. Arleo, F. Smeraldi and W. Gerstner, "Cognitive navigation based on non-uniform Gabor space sampling, unsupervised growing networks and reinforcement learning", *IEEE Transactions on Neural Networks*, vol. 15, pp. 639-652, 2004
- [2] J.G. Daugman, "Two-dimensional spectral analysis of cortical receptive field profiles", *Vision Research*, vol 20, pp. 847-856, 1980

- [3] D.J.Field, "Relations between the statistics of natural images and the response properties of cortical cells", *Journal of the Optical Society of America A*, vol 4-12, pp. 2381, 1987
- [4] B.S. Manjunath W.Y. Ma, "Texture features for browsing and retrieval of image data", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 18-8, pp 837, 1996
- [5] W.K. Pratt, "Digital image processing", Wiley, New York (1978)
- [6] J.Yang, L. Liu, T. Jiang, Y.Fan, "A modified Gabor filter design method for fingerprint image enhancement", *Pattern Recognition Letters*, vol 24, pp. 1807, 2003