Adaptive Filtering
Filter optimization, tensor representation of local structure and adaptive image enhancement

Mats Andersson Anders Eklund Hans Knutsson
Department of Biomedical Engineering, Linköping University

5 oktober 2016

Contents
The purpose of this project is to give a deeper understanding of how parameters such as the frequency response of the filter, the spatial support, the image spectrum and the SNR (signal to noise ratio) is included in the design of filters for multi-dimensional signal processing. You will learn how to use the kernel optimizer krnopt. The optimized filters are in the next step used to compute a local structure description using a tensor representation of the image. In the last part of the project, this tensor data is used to control a local adaptive image enhancement algorithm. The mini project consists of two classes, three laborations and a written report.

Preparations
The theory behind this project is covered in chapter 4-5, chapter 6.1-6.5 (the 2D part) and chapter 10 in Signal Processing for Computer Vision by Granlund and Knutsson, and chapter 1-5 in the Advanced Filter Design paper. The project will be performed in Matlab. A Matlab library for filter optimization (kerngen.zip, for kernel generator) can be downloaded from the course home page http://www.imt.liu.se/edu/courses/TBMI02/miniproject.html.

1 Filter optimization
All filter optimization is a compromise between conflicting demands, which have to be considered in relation to each other for an optimal result. The filter optimizer solves a weighted least squares problem. In this project we focus on how the input functions and parameters to krnopt should be adapted to the image features.
The inputs to the kernel optimizer consists of

**Frequency ideal function** A description of the ideal frequency function in the Fourier domain (FD).

**Frequency weight function** This weight function is closely related to the spectrum of the image and the spectrum of the noise.

**Spatial mask** A binary mask that limits the spatial support (size) of the filter.

**Spatial weight function** The purpose of this weight function is to further limit the spatial support (size) of the filter in the optimization, by introducing a cost for large filter coefficients close to the border of the spatial support. This requirement is most simple to formulate in the spatial domain (SD).

**Spatial ideal function** The most compact spatial filter function there is (a Dirac impulse in the center of the filter).

### 1.1 The filter optimizer krnopt

Both the input and output of krnopt are multi-dimensional arrays (filters and weight functions) in the spatial- and frequency-domain. To simplify the handling of multiple domains, the multi-dimensional arrays are embedded in a Matlab struct called *wa* (weight array). The *wa* contains the multi-dimensional array, a domain specific coordinate system and a domain flag. To generate an empty *wa* we type in

```matlab
>> wa([n,n], 0) % empty (n x n) wa in SD
>> wa([N,N], 1) % empty (N x N) wa in FD
```

**Question 1:** Why do we usually need more samples in the FD? What happens if we optimize a filter using the same number of samples in both domains?

You cannot mix spatial and frequency *wa* in the same Matlab expression. You can add a *wa* and a Matlab array of the same size. The result will be a *wa*. Most Matlab functions work for *wa*. Some exceptions are ‘*’ and ‘/’ but ‘.*’ and ‘./’ work. You can import/export between *wa* and Matlab arrays using *getdata* and *putdata*. To plot a *wa* use the function *wamesh*. Other useful functions are *dft*, *getgrid*, *getcoord* and *putorigo*. Use help

```matlab
>> help function_name
```

to find out more. There exists a wide range of scripts that generate ideal and weight functions in *wa* -format e.g. *quadrature*, *F_weightgensnr* and *goodsw*. krnopt has the following in and out parameters (all of them are *wa*).

```matlab
>> [f, F] = krnopt(Fi, Fw, fm, fi, fw);
```
Krnopt returns the optimized filter in both the spatial domain and in the frequency domain. The input arguments are: frequency ideal function, frequency weight function, spatial mask, spatial ideal function and spatial weight function.

1.2 Optimization of quadrature filters

The first part of this project is to write two Matlab scripts. One for optimization of a quadrature filter set and one for optimization of an image enhancement filter set. The quadrature filters are used to estimate the structure tensor, which will control the enhancement filters. Start by defining the number of filters to use, the corresponding filter directions, the bandwidth \( B \) and center frequency \( u_0 \) of the quadrature filters, the size of the filter in the SD and in the FD.

Question 2: How many quadrature filters are required to estimate the structure tensor in 2D? Motivate your answer.

Note that krnopt follows the Matlab convention using \( y \) (the row index) as the first index. Use the function goodsw to compute the spatial weight function

\[
\text{goodsw}(sz_s, rexp);
\]

\[
\text{wamesh}(fw0) \quad \% \text{plot fw0}
\]

where \( sz_s \) is the size of the filter in the spatial domain (e.g. \([11 \ 11]\)).

Question 3: How does \( rexp \) affect the spatial weight function? What is the difference between using the spatial weight function to limit the spatial support as to choose a smaller spatial size? What happens for very large \( rexp \)?

Use \( \text{F_weightgensnr} \) to generate the frequency weight function (same for all quadrature filters). The most important parameters are \( rexp \) and \( \text{SNRest} \), which represent the spectrum of the image and the SNR.

\[
\text{F_weightgensnr}(sz_F, rexp, cosexp, SNRest);
\]

where \( sz_F \) is the size of the filter in the frequency domain (e.g. \([23 \ 23]\)). The frequency ideal function (different for each direction \( \text{dir}(k) \)) is defined using the function \( \text{quadrature} \). The last argument is a vector that defines the direction of the filter, \([10]^T\) for example gives a filter in the y-direction.

\[
\text{quadrature}(sz_F, u0, B, \text{dir}(k));
\]

Create an ideal function for each quadrature filter \( k \), for example using a for-loop. To facilitate the choice of center frequency, \( u0 \), and bandwidth, \( B \), you can use the Matlab script \( \text{lift (local interactive Fourier transform)} \). Type \( \text{lift(image name)} \) and study the local Fourier transform of some interesting neighborhoods in your image.

Time to optimize some filters, use no spatial weight the first time. Note that you need to do
the optimization separately for each filter $k$, for example using a for-loop.

```matlab
>> fw_amp = 0; % defines the amount of spatial weight
>> fw = fw_amp.*fw0;
>> [f{k}, F{k}] = krnopt(Fi{k}, Fw, fm, fi, fw);
```

Plot the optimized filters in both domains

```matlab
>> figure(1), wamesh(f{1})
>> figtitle('real part f{1}')
>> figure(2), wamesh(imag(f{1}))
>> figtitle('imaginary part f{1}')
>> figure(3), wamesh(F{1})
>> figtitle('F{1}')
```

Use the function `filterr` to estimate the errors in the optimized filter.

```matlab
>> filterr(F{1}, Fi{1}, Fw, fi, fw, 1);
```

`Filterr` computes many error measures. One of the most important is the error in the FD (space 2). Note that there are weighted and unweighted errors.

**Question 4:** Why is the weighted FD error usually larger then the unweighted?

Increase the spatial weight gradually and observe what happens to the filter. The weighted error in the FD should in this case not exceed 0.2 - 0.25.

Try to optimize using other center frequencies and bandwidths. Try to get a feeling for the required spatial size from $u_0$ and $B$. How much larger should you make the spatial size if $u_0$ is decreased by half an octave?

The difference between the center frequencies $\pi/2$ and $\pi/4$ is one octave, the difference between $\pi/2$ and $\pi/8$ is two octaves. Half an octave is defined using the factor $\sqrt{2}$.

### 1.3 Optimization of enhancement filters

This filter set consists of one isotropic LP-filter and anisotropic BP-filters in the same directions as the quadrature filters. Use the same frequency weight function as for the quadrature filters, but the spatial support of the enhancement filters may differ from the quadrature filters. The frequency ideal functions are generated by the functions `enhlp` and `enhp`.

```matlab
>> H0i = enhlp(sz_F, wlp); % Ideal LP-filter H0i
>> Hi{k} = enhhp(sz_F, dir{k}, wlp); % Ideal BP-filter Hi{k}
```

Optimize the filters in the same way as you optimized the quadrature filters. Start by a small spatial weight which is gradually increased. Keep track of the filter shape and the FD error. Note that all these filters should only contain real valued filter coefficients, since all the filters
are even. The filter optimization may result in some very small imaginary filter coefficients, they can be removed by

```matlab
>> filter = real(filter);
```

Plot the optimized filters, what is the difference from the quadrature filters? Plot the sum of the LP filter and one of the BP-filters? Do they fit together?

**Question 5:** How does the parameter $w_{lp}$ change the filter shape? How would you expect a change in $w_{lp}$ to affect the result of the adaptive filtering?

**Question 6:** These filters are all even. How come only even filters are sufficient for adaptive filtering?

## 2 Tensor representation of local structure

Next the quadrature filter responses will be used to generate a local structure tensor for each neighborhood in the image. Use the test image 'ploop' to verify that the results are correct.

```matlab
>> im = ploopgen512;
>> gimage(im, 1); % display test image
```

Compute the filter responses using convolution. To reduce the edge effects in the filtering we use a convolver which performs edge extraction (pixels outside the image are set to have the same value as the closest pixels inside the image), this is achieved with the option 'replicate'.

```matlab
>> q{k} = abs(imfilter(im, getdata(f{k}),'replicate','conv'));
```

Do the convolution for all quadrature filters $k$, using a for-loop. Note that the function `imfilter` by default performs correlation, it is therefore necessary to add the option 'conv' (for convolution). For symmetric filters, e.g. a Gaussian lowpass filter, it does not matter if we use correlation or convolution. Quadrature filters are, however, not symmetric.

**Question 7:** Why are we using the function `abs()` here?

**Question 8:** Look at the filter responses of the ploop using `gimage`, how can you tell if the magnitude is phase independent?

The local tensor $T$ is computed as:

$$
T = \begin{pmatrix} t_1 & t_2 \\ t_2 & t_3 \end{pmatrix} = \sum_k |q_k| M_k \\
M_k = \ldots
$$

Compute the 3 independent tensor components $t_1$, $t_2$ and $t_3$ separately as 3 images.
Look at the result using \texttt{mygopimage}.

\texttt{>> mygopimage(t1, 1)}

The function \texttt{mygopimage} is used to visualize signed and complex valued images. The argument is coded in the colour. Positive and real values are shown as green. Negative and real values are shown as red. Imaginary numbers correspond to blue and yellow.

Can you make an intuitive interpretation of the diagonal tensor components? How about the off-diagonal term $t_2$?

Although the local tensor contains more information it is sometimes convenient to map the tensor to one complex valued image, the so called \textit{gop} or \textit{double argument} representation.

\texttt{>> gop = t3 - t1 + sqrt(-1)*2*t2;}
\texttt{>> mygopimage(gop, 1);}

\textbf{Question 9}: How does the gop-representation work? Show that we get a continuous representation of orientation from a rank 1 tensor using the gop mapping.

Using \texttt{gopval} you can study the magnitude and argument for the gop mapping interactively. How many laps does the vector turn when you move the pointer on turn?

\texttt{>> gopval(gop)}

Finally the local tensor is averaged using a small LP-kernel

\texttt{>> t1 = aver2(t1, 5);}
\texttt{>> t2 = aver2(t2, 5);}
\texttt{>> t3 = aver2(t3, 5);}

\texttt{aver2} performs a separable averaging using a Gaussian kernel.

\textbf{Question 10}: Why is it beneficial to use a small averaging kernel here? Why don’t we skip the averaging and instead use a quadrature filter with a larger spatial support?

\section{Local adaptive image enhancement}

The next step is to create the control tensor, $C$, which controls the adaptive filtering. We start by disassembling the local tensor, $T$, into eigenvalues and eigenvectors.

$$ T = \begin{pmatrix} t_1 & t_2 \\ t_2 & t_3 \end{pmatrix} = \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T \quad \lambda_1 > \lambda_2 > 0 $$

For a second order 2D tensor $\lambda_1$ and $\lambda_2$ can be computed explicitly.

$$ T = \begin{pmatrix} t_1 & t_2 \\ t_2 & t_3 \end{pmatrix} \quad \lambda_1 = \frac{1}{2} (t_1 + t_3 + \ldots) \quad \lambda_2 = \frac{1}{2} (t_1 + t_3 - \ldots) $$
Usually $\lambda_2 \geq 0$. As a precaution against numerical problems we use the absolute value.

$$ >> \text{12 = abs(12); } \quad \% \text{ make sure lambda2} > 0 $$

**Question 11:** Generate a random $2 \times 2$ symmetric matrix ($\text{test} = \text{randn(2,2)}; \text{test} = \text{test} + \text{test'}$). Calculate the eigen values using your own formulas and using the Matlab function `eig`. Do the results match? Why don’t we simply loop over all the pixels and call the function `eig` for each pixel?

The control tensor $C$ has the same eigen system as $T$. There is no need to compute $e_1$ and $e_2$, it is sufficient to compute $e_1^T e_1$ and $e_2^T e_2$. To make a robust and efficient computation of $e_1^T e_1$ and $e_2^T e_2$ we make use of the following facts:

$$ T = \lambda_1 e_1^T e_1 + \lambda_2 e_2^T e_2 \quad I = e_1^T e_1 + e_2^T e_2 $$

$$ >> \text{ee1}_1 = \ldots; \\
>> \text{ee1}_2 = \ldots; \\
>> \text{ee1}_3 = \ldots; \\
>> \text{norm} = \text{sqrt(ee1}_1.^2 + 2*\text{ee1}_2.^2 + \text{ee1}_3.^2); \\
>> \text{norm} = \text{1./(norm+eps)}; \\
>> \text{ee1}_1 = \text{norm.*ee1}_1; \\
>> \text{ee1}_2 = \text{norm.*ee1}_2; \\
>> \text{ee1}_3 = \text{norm.*ee1}_3; $$

Finally compute $e_2^T e_2$ as

$$ e_2^T e_2 = I - e_1^T e_1$$

### 3.1 Mapping of the control tensor $C$

The control tensor has the same eigen system as $T$, but the eigen values are different.

$$ C = \begin{pmatrix} c_1 & c_2 \\ c_2 & c_3 \end{pmatrix} = \gamma_1 e_1^T e_1 + \gamma_2 e_2^T e_2 $$

The eigen values of $C$, $\gamma_1$ and $\gamma_2$, are mappings of the eigen values of $T$, $\lambda_1$ and $\lambda_2$. We use two mapping functions, `m_func` and `mu_func`, which correspond to the $m$- and $\mu$-functions on p. 317 in the course book.

![Figur 1: Example of m_func and mu_func mappings.](image-url)
The \texttt{m_func} defines how much of the BP-filter content that will be used in the result image.

\begin{equation}
m(x, \sigma, \alpha, \beta, j) = \left( \frac{x^\beta}{x^\beta + \alpha \beta + \sigma \beta} \right)^{1/j}
\end{equation}

The position dependent parameter \( x \) is expected to be in the interval \([0, 1]\) and is computed from the magnitude of \( T \).

\begin{verbatim}
>> x = sqrt(11.^2 + 12.^2);
>> x = x/max(x(:));
\end{verbatim}

The parameter \( \sigma \) is a soft noise threshold. The parameter \( \alpha \) defines an overshoot which improves visibility of small structures just above the noise threshold (use with care!). A good way to experience the \texttt{m_func} is to look at the curve and \( \gamma_1 \)-image while testing different parameters.

\begin{verbatim}
>> m_s = ...; % m-sigma
>> m_a = ...; % m-alpha
>> m_b = ...; % m-beta
>> m_j = 1; % m-j
>> gam1 = m_func(x, m_s, m_a, m_b, m_j, 1); % Compute gamma1 and plot curve
>> figure(10), gimage(gam1,-1) % display gamma1 image
\end{verbatim}

The \texttt{gam1}-image should be bright where the original image has apparent local structure, but dark in flat areas or areas containing plain noise.

**Question 12:** What should the \texttt{gam1}-image look like for the ploop test image? What does it look like in your case?

The next mapping function is \texttt{mu_func} or the \( \mu \)-function.

\begin{equation}
\mu_2(x, \alpha, \beta, j) = \left( \frac{(x (1 - \alpha))^\beta}{(x (1 - \alpha))^\beta + (\alpha (1 - x))^\beta} \right)^{1/j}
\end{equation}

This function defines in what orientation(s) the BP-filter content will contribute to the result image, i.e. the degree of anisotropy in the adaptive filtering. The most important parameters are \( \alpha \) and \( \beta \) which control the position and steepness of the transition band in the \texttt{mu_func} curve. The position dependent parameter \( x \) is the relation between \( \lambda_2 \) and \( \lambda_1 \)

\begin{equation}
x = \frac{\lambda_2}{\lambda_1}
\end{equation}

\begin{verbatim}
>> x = 12./(11+eps);
>> mu_a = ...; % mu-alpha
>> mu_b = ...; % mu-beta
>> mu_j = 1; % mu-j
>> mu2 = mu_func(x, mu_a, mu_b, mu_j, 1); % Compute mu2 and plot curve
\end{verbatim}
The \textit{mu2}-image only depends on the tensor shape and is independent of the magnitude. The \textit{mu2}-image is dark where the local orientation is well defined and bright where the orientation is ambiguous such as in plain noise.

Finally $\gamma_2$ is computed as

$$\gamma_2 = \gamma_1 \mu_2$$

to make $\gamma_2$ dependent both on the tensor magnitude and the local energy distribution (anisotropy). Note that the multiplication between $\gamma_1$ and $\mu_2$ is elementwise (i.e. .* and not *).

Question 13: What should the \textit{gam2}-image look like for the ploop test image? What does it look like in your case? Can you think of a case where the \textit{gam2}-image will be bright?

Since the \texttt{m_func} and \texttt{mu_func} are non-linear a small component wise averaging of $C$ may be appropriate.

### 3.2 Compute filter weights

Finally the control tensor

$$C = \begin{pmatrix} c_1 & c_2 \\ c_2 & c_3 \end{pmatrix}$$

and the projection tensors $M_k$ are used to compute a weight for each pixel and for each BP-filter. The weight in each pixel for filter $k$, $w_k$, is computed as a scalar product between the control tensor and each of the projection tensors.

$$w_k = C \cdot M_k$$

### 3.3 Compute the enhanced image

Apply the optimized isotropic LP-filter and the optimized anisotropic BP-filters to the (noisy) image, i.e.

$$\texttt{imlp} = \text{imfilter}(\texttt{im}, \text{getdata(hlp)},\text{'replicate','conv'});$$

$$\texttt{imbp\{k\}} = \text{imfilter}(\texttt{im}, \text{getdata(hbp\{k\})},\text{'replicate','conv'});$$

Perform the convolution for all bandpass filters $k$, using a for-loop. Finally compute the enhanced image $\texttt{im}_{\text{enh}}$ by summing the LP-filter response $\texttt{im}_{\text{lp}}$ with the weighted BP-filter.
responses $w_k \cdot im_{bp}(k)$. Note that the multiplication between the weights and the BP-filter responses is elementwise (i.e. .* and not *).

$$im_{enh} = im_{lp} + \sum_k w_k \cdot im_{bp}(k)$$

Compare the resulting image to the LP-filtered original image. Did the result meet your expectations? If not try to analyze what went wrong and make another try. Check list:

- Is there relevant structure information present in $T$? Look at the GOP-representation of $T$. If not adjust center frequency and/or bandwidth of the quadrature filters.
- How much noise is present in the LP-image? Is the borderline between LP and BP right?
- Is the noise suppressed in flat areas or do we enhance noise induced structures? Check the noise threshold $\sigma$ in m_func.
- For which edges or lines do we have maximal anisotropy for the BP-filters? Look at the mu2-image and change the mu_func parameters.
- Look at the GOP-representation of $C$ to further verify your m_func and mu_func parameters.
- If you optimize new filters, don’t forget to check the distortion and adjust the spatial support.

### 3.4 Image enhancement

Start by using the ploop as test image with a small amount of noise. When you are satisfied by the result use a natural image or a medical image. Do at least two image enhancements of your image, using SNR=20dB and SNR=5dB.

### 3.5 Two noise exercises

1. Compute the control tensor for a natural image without noise. Add a fair amount of noise to your image and run the adaptive filters, but modify the the image enhancement algorithm such that the filtering will occur in an orientation orthogonal to the one specified by the local tensor. What effect will you get?

2. Compute the control tensor for a natural image without noise. Use the control tensor to enhance an image containing just noise. Describe how you see the effect of the adaptive filters in the shape of the resulting noise image.