Solutions to Exercise Session
Filtering

Note on filter masks

The filters that you will see in this exercise can in general be divided into two classes:

- integrating filters (averaging filters, smoothing filters, low-pass filters)
- differential filters (derivative filters, high-pass filters, band-pass filters)

While the components of the second class of convolution filters usually sum up to 0 (i.e. their corresponding MTF is 0 at the origin), the components of smoothing filters are usually chosen s.t. their components sum up to 1 (i.e. the average grey value of the image is not altered by the convolution with such a filter). However, for the sake of readability the integrating filters in this exercise are not normalised. You can transform them into normalised filters by simply dividing all components by their sum (spatial domain) or dividing by the value at the origin (frequency domain).

1 Noise filtering

Suppose you get an image that contains lot of noise. Smoothing the image is one option to reduce the noise. We're going to apply two filters (separately) in the frequency domain:

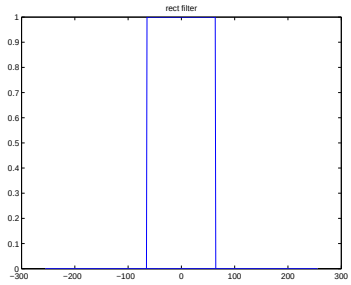
1. a block filter (rectangular disk MTF)
2. a Gaussian filter (Gaussian MTF)

Which do you think will have the nicest results? What is the undesired side-effect of the other one?

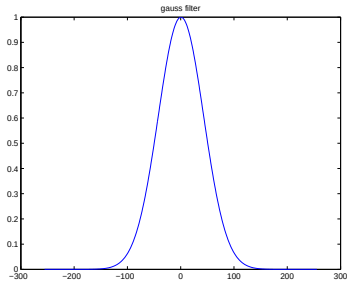
The block filter corresponds to a sinc convolution mask in the spatial domain whereas the Gaussian filter corresponds to another Gaussian. While the Gaussian decreases smoothly to zero the sinc function exhibits negative values. Negative values cause undesired ripples in the block-filtered image whereas the Gaussian-filtered image looks much smoother (cf. Figure 1).

2 Edge detection

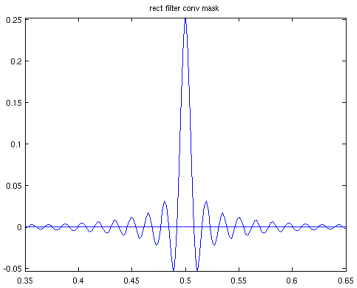
There exist many methods for edge detection. In the following we will consider a simple linear filter.



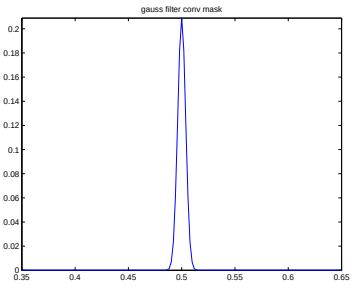
(a) Block filter (frequency domain)



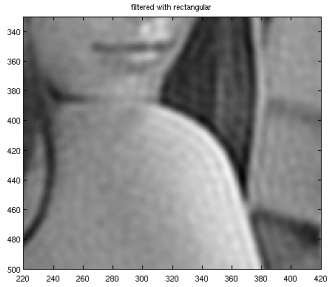
(b) Gaussian filter (frequency domain)



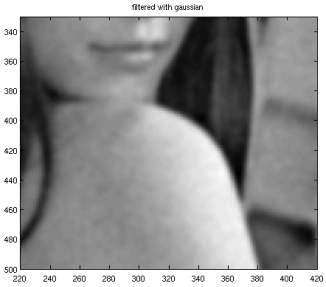
(c) Convolution mask of block filter (spatial domain)



(d) Convolution mask of Gaussian filter (spatial domain)



(e) Close-up of block-filtered image



(f) Close-up of Gaussian-filtered image

Figure 1: *Image filtering using block filter and Gaussian filter.*

- Starting from the Gaussian filter from exercise 1, how could you turn it into an edge-filter?

  Edges are regions in the image with high-frequency content. If we could remove all low frequencies from an image, the edges would remain. Since the previous smoothing filter did the exact opposite, one solution would be to subtract the smoothed image from the original one. That would leave only high-frequencies, which includes the edges.
  This is equivalent to subtracting the Gaussian spectrum from a flat one, i.e. in 1D it would look like this:

$$H_u(u) \quad = \quad 1 - G(u, \sigma_f) \quad \text{with e.g. } \sigma_f = 10 \tag{1}$$

  That filter is shown in Figure 2(a).

- How does the filter's spatial convolution mask look like?

  The inverse Fourier transformation of $H_u(u)$ in one dimension gives:

$$\begin{aligned} h_x(x) &= \mathcal{F}^{-1}\{H_u(u)\} \\ &= \mathcal{F}^{-1}\{1\} - \mathcal{F}^{-1}\{G(u, \sigma_f)\} \\ &= \delta(x) - G(x, \frac{1}{2\pi\sigma_f}) \end{aligned}$$

  That convolution mask is shown in Figure 2(b).

- What is its response to an abrupt edge?

  To see this intuitively, we approximate the convolution mask by a difference of Gaussian filters (commonly known as DOG filter):

$$\begin{aligned} h_x(x) &= \delta(x) - aG(x, \sigma_s) \\ &\approx G(x, \sigma_{\text{small}}) - G(x, \sigma_{\text{wide}}) \end{aligned}$$

  That means, a strongly smoothed (σ_{wide}) version is subtracted from a lightly smoothed (σ_{small}) version of the image. Figure 3 shows the response in case of an abrupt edge and how this effect is well visible at Lena's hat.

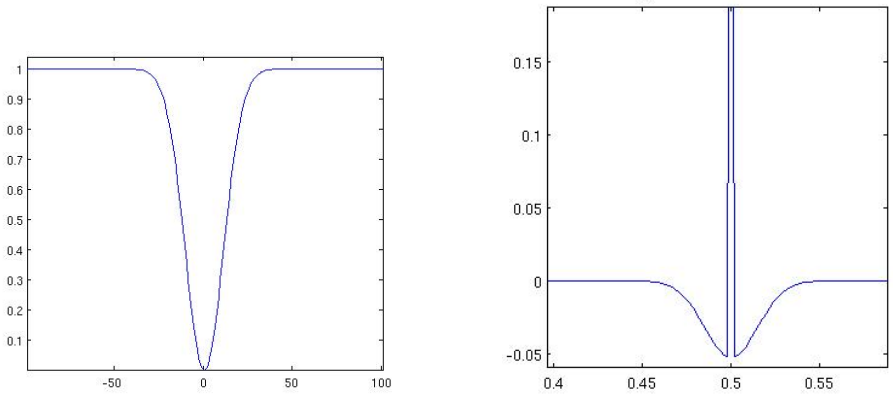
The noise inherent in the image affects the edge detection.

- Explain why.

  Edges are of course not the only structures residing in the high-frequency part of an image. There reside other thin image structures like hair or grass etc. Also our assumption of additive white noise means, that noise within the high-frequency part will not be filtered out.

- Although it's intrinsic to the problem, you can reduce the effects. How?

  One could smooth the image before edge detection. That way one filters out a certain amount of noise (depending on the amount of smoothing) while still keeping most of the edges. Instead of two subsequent operations it is also possible to do both steps in one using a band-bass instead of a high-pass filter. Keep in mind that subtracting a wide Gaussian from a small Gaussian in the spatial domain corresponds to subtracting a small Gaussian from a wide Gaussian in the frequency domain.



(a) Highpass (edge) filter (b) Convolution mask

Figure 2: *Highpass filter*

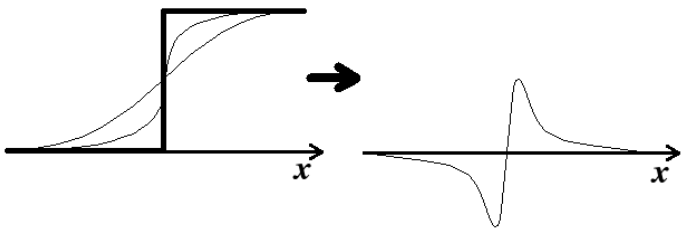


Figure 3: *Approximation of an abrupt edge using two differently smoothed versions.*

3 Linearity and shift-invariance

Which of the following operators are linear and shift-invariant? Could you see this intuitively?

$$x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y} \tag{2}$$

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{3}$$

$$\left(\frac{\partial}{\partial x}\right)^2 + \left(\frac{\partial}{\partial y}\right)^2 \tag{4}$$

A short reminder:

$$\begin{aligned} O\{\cdot\}\text{is linear} &\iff O\{af(x,y)+bg(x,y)\} = aO\{f(x,y)\}+bO\{g(x,y)\} \\ O\{\cdot\}\text{is shift}-\text{invariant} &\iff O\{f(x-x_0,y-y_0)\} = O\{f(u,v)\}|_{u=x-x_0,v=y-y_0} \end{aligned}$$

Operator 1: $\mathbf{x}\frac{\partial}{\partial \mathbf{x}} + \mathbf{y}\frac{\partial}{\partial \mathbf{y}}$

$$\begin{aligned} O\{af+bg\} &= x\frac{\partial}{\partial x}(af+bg) + y\frac{\partial}{\partial y}(af+bg) \\ &= x(af_x+bg_x)+y(af_y+bg_y) \\ &= a(xf_x+yf_y)+b(xg_x+yg_y) \\ &= aO\{f\}+bO\{g\} \\ &\Longrightarrow \textbf{linear} \\ O\{f(x-x_0,y-y_0)\} &= x\frac{\partial}{\partial x}\left((f(x-x_0,y-y_0)\right)+y\frac{\partial}{\partial y}\left((f(x-x_0,y-y_0)\right) \\ &= xf_x(x-x_0,y-y_0)+yf_y(x-x_0,y-y_0) \\ &\neq (x-x_0)f_x(x-x_0,y-y_0)+(y-y_0)f_y(x-x_0,y-y_0) \\ &\Longrightarrow \textbf{not shift}-\textbf{invariant} \end{aligned}$$

Operator 2: $\frac{\partial^2}{\partial \mathbf{x}^2} + \frac{\partial^2}{\partial \mathbf{y}^2}$ *(Laplacian* $\longrightarrow \nabla^2$*)*

$$\begin{aligned} O\{af+bg\} &= \frac{\partial^2}{\partial x^2}(af+bg) + \frac{\partial^2}{\partial y^2}(af+bg) \\ &= (af_{xx}+bg_{xx})+(af_{yy}+bg_{yy}) \\ &= a(f_{xx}+f_{yy})+b(g_{xx}+g_{yy}) \\ &= aO\{f\}+bO\{g\} \\ &\Longrightarrow \textbf{linear} \\ O\{f(x-x_0,y-y_0)\} &= \frac{\partial^2}{\partial x^2}\left(f(x-x_0,y-y_0)\right)+\frac{\partial^2}{\partial y^2}\left(f(x-x_0,y-y_0)\right) \\ &= f_{xx}(x-x_0,y-y_0)+f_{yy}(x-x_0,y-y_0) \\ &= O\{f(u,v)\}|_{u=x-x_0,v=y-y_0} \\ &\Longrightarrow \textbf{shift}-\textbf{invariant} \end{aligned}$$

Operator 3: $\left(\frac{\partial}{\partial x}\right)^2 + \left(\frac{\partial}{\partial y}\right)^2$ ***(square of gradient-magnitude*** $\longrightarrow \|\nabla\|^2$***)***

$$\begin{aligned}
O\{af+bg\} &= \left(\frac{\partial}{\partial x}(af+bg)\right)^2 + \left(\frac{\partial}{\partial y}(af+bg)\right)^2 \\
&= (af_x + bg_x)^2 + (af_y + bg_y)^2 \\
&\neq a(f_x^2 + f_y^2) + b(g_x^2 + g_y^2) \\
&\Longrightarrow \textbf{not linear} \\
O\{f(x-x_0, y-y_0)\} &= \left(\frac{\partial}{\partial x} f(x-x_0, y-y_0)\right)^2 + \left(\frac{\partial}{\partial y} f(x-x_0, y-y_0)\right)^2 \\
&= \left(\frac{\partial}{\partial (x-x_0)} f(x-x_0, y-y_0)\right)^2 + \left(\frac{\partial}{\partial (y-y_0)} f(x-x_0, y-y_0)\right)^2 \\
&= O\{f(u,v)\}|_{u=x-x_0, v=y-y_0} \\
&\Longrightarrow \textbf{shift} - \textbf{invariant}
\end{aligned}$$

The fact that the first operator is not shift-invariant should be obvious, since it depends on the location (multiplication with x resp. y). The non-linearity of operator 3 can also be seen directly: linear operators (first derivatives in x and y) are squared and then added. Note that this is different from operator 2, which is the sum of two linear operators (second derivatives).

4 Median filtering

Smoothing an image looses a lot of its edge information. Median filtering can overcome this problem to a certain extent, since it removes noise, while preserving edges.

- Is median filtering a linear operation?

  No, because:

  $$\text{median}\{af(x) + bg(x)\} \neq a \cdot \text{median}\{f(x)\} + b \cdot \text{median}\{g(x)\}$$

  Consider the following example:
  $a = 1$, $b = 3$, $f(x) = [1\,3\,2]$ *and* $g(x) = [4\,5\,6]$*:*

  $$\begin{aligned}
  \text{median}\{[1\,3\,2] + 3 \cdot [4\,5\,6]\} &= \text{median}\{[13\,18\,20]\} = 18 \\
  \text{median}\{[1\,3\,2]\} + 3 \cdot \text{median}\{[4\,5\,6]\} &= 2 + 3 \cdot 5 = 17 \neq 18
  \end{aligned}$$

- Will running a 3x3 median operation twice yield the same result as one 5x5 median operation? Give an example (1D).

  No, see Figure 4 for a simple counterexample. In real images this effect is less noticable, however, applying a small window median operator very often to the same image is indeed very different from applying a big median operator only once, as shown in Figure 5.

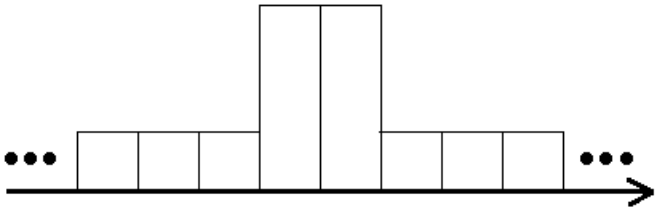


Figure 4: *This input pattern will remain unchanged by applying a* 3×3 *median operator twice, but will be flattened by a* 5×5 *median operator.*

Figure 5: *A small median operator applied repeatedly to the same image (left) yields different results than applying a big median operator once (right).*

5 Matched filters

Suppose the following convolution mask:

$$F = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 2 & 4 & 0 & -4 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -4 & 0 & 4 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} \tag{5}$$

- Is this filter separable? Try to separate it as much as possible.

  Yes, it is separable into 3 parts corresponding to smoothing, vertical and horizontal edge detection:

$$\begin{aligned} F &= \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 2 & 4 & 0 & -4 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -4 & 0 & 4 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 0 \\ -2 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}}_{\text{smoothing}} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}}_{\text{vertical edges}} * \underbrace{\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}}_{\text{horizontal edges}} \end{aligned}$$

- Looking at all the components, what is its MTF?

  The $[1 \quad 2 \quad 1]$ *part first component can either be seen as a 2D Gaussian or two symmetric Dirac pulses with a third central Dirac pulse. In the first case we would gain a Gaussian, in the latter case a cosine (shifted up in z-direction with a period exactly equal to the full frequency range). This will be multiplied with a vertical sine and a horizontal sine function, both also with a period equal to the frequency range. See Figure 6 (the fact that we see two peeks in each dimension comes from the use of the spectrum for visualisation, which does not allow the display of negative values).*

- What's the purpose of this filter?

  The first term will smooth the image, reducing the noise. next it will respond strongly to regions in the image with both vertical and horizontal edges. Hence, it will detect corners in the image. See Figure 7.

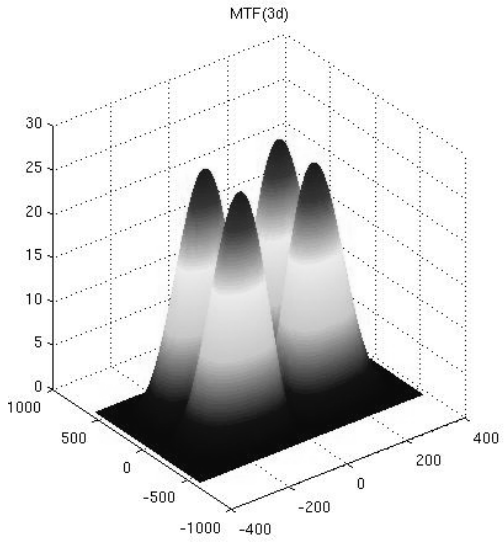


Figure 6: *MTF of matched filter from exercise 5.*

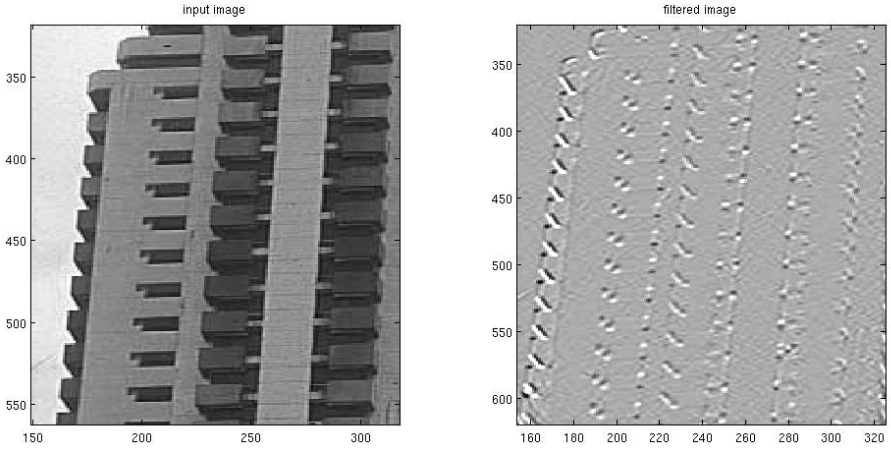
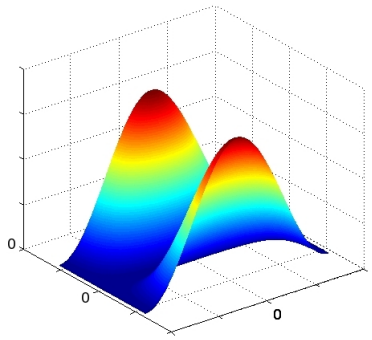


Figure 7: *Corner detection using matched filter.*

6 The other way around

Let's try it the other way around. Given a MTF, what is the corrsponding convolution mask?



- What filter coefficients result in such a MTF spectrum?

  The image shows the following MTF:

$$H(u,v) = (2 + 2\cos(2\pi u)) \cdot (2 - 2\cos(2\pi v))$$

  In a continuous spatial domain, this corresponds to:

$$\begin{aligned} h(x,y) &= \mathcal{F}^{-1}\{2 + 2\cos(2\pi u)\} * \mathcal{F}^{-1}\{2 - 2\cos(2\pi v)\} \\ &= (2\delta(x) + \delta(x-1) + \delta(x+1)) * (2\delta(y) - \delta(y-1) - \delta(y+1)) \end{aligned}$$

The discrete convolution mask is thus:

$$h = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix}$$

- What would this filter do to an image?

  The first term does horizontal smoothing, the last term approximates the second derivative to the vertical variable. Thus, according to the matched filter rule, it responds to horizontal lines in the image. See Figure 9.

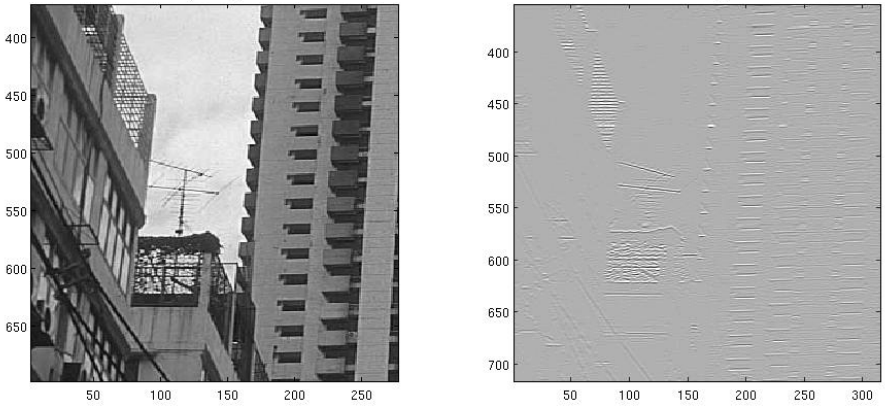


Figure 9: *This filter responds to horizontal lines.*

7 Isotropy

Let's compare the following two operators:

$$\sqrt{\left(\frac{\partial}{\partial x}\right)^2 + \left(\frac{\partial}{\partial y}\right)^2} \tag{6}$$

$$\left|\frac{\partial}{\partial x}\right| + \left|\frac{\partial}{\partial y}\right| \tag{7}$$

- What are their correspondences?

  Both are norms of the gradient operator. These can be used as edge detectors.

- What are their differences?

  The first operator is the L2-norm of the gradient (gradient-magnitude) and therefore an isotropic operator, i.e. it responds equally to edges in all directions. The second operator (L1-norm of the gradient) is anisotropic, i.e. responds stronger to edges in vertical direction than in x- or y-direction. See Figure 11.

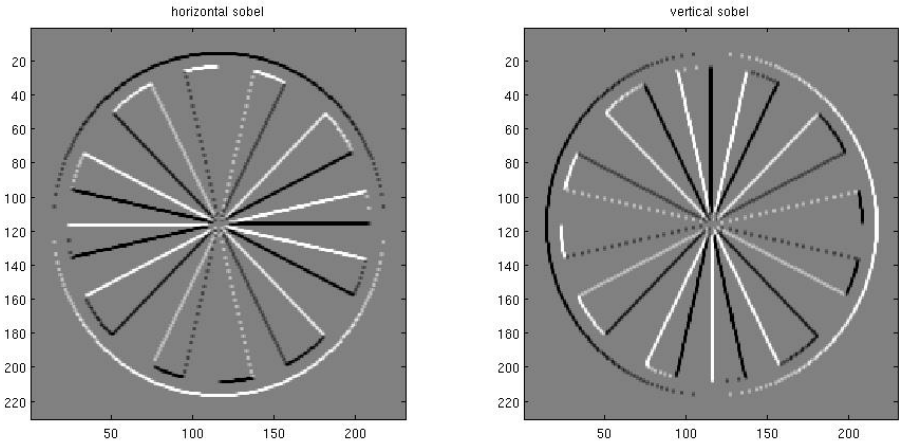


Figure 10: *Sobel edge filtering in horizontal and vertical direction.*

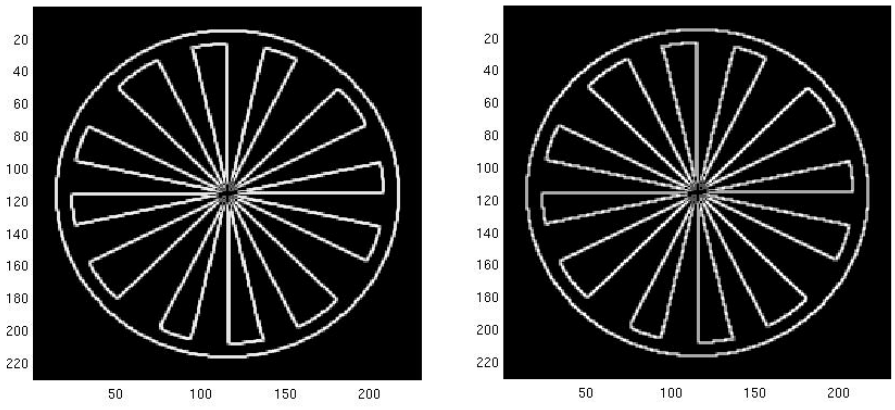


Figure 11: *Isotropic and anisotropic edge filtering.*

- How would one implement such filters?

  One possible implementation was proposed by Sobel:

$$\frac{\partial}{\partial x} \quad \leftarrow \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial}{\partial y} \quad \leftarrow \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

  This corresponds to a smoothing operation in the orthogonal direction followed by differential filtering in the desired edge direction. To combine them to a norm of the gradient we have to use the filter-responses and perform further computations on them. It is not possible to implement the $L1$ or $L2$ norm directly as a linear filter, since neither of these norms is linear.

9 Unsharp masking

Start off with the following Gaussian mask:

$$G = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{9}$$

- How can we use it to make a Laplacian?

  A convolution with a Gaussian kernel corresponds to a diffusion process. The diffusion equation is given as:

  $$\frac{\partial f}{\partial t} = c \cdot \Delta f,$$

  where $\Delta = \nabla^2$ is the Laplacian operator and c a positive constant. Approximating $\frac{\partial f}{\partial t} = G_\sigma * f - f$ (i.e. the difference between the blurred image and the original one) and ignoring the constant c, we can construct a Laplacian operator L:

  $$\begin{aligned} G_\sigma * f - f &= L * f \\ \Rightarrow L &= G_\sigma - \delta(0,0) \end{aligned}$$

  The MTF of this operator is given as:

  $$\mathcal{F}\{L\} = \mathcal{F}\{G_\sigma\} - 1$$

  Note that this is simply the high-pass filter in equation (1) in part 2 of this exercise session with a negative sign. Since our given Gaussian G is not normalised, we have to scale the Dirac accordingly. In discrete terms, that means multiplying it with the sum of all coefficients of the Gaussian (remember that for normalised kernels this would be 1):

  $$\begin{aligned} L(i,j) &= G(i,j) - \delta\left(x - \frac{N}{2}, y - \frac{N}{2}\right) \cdot \sum_{i,j}^{N,N} G(i,j) \\ &= G(i,j) - \delta\left(x - \frac{N}{2}, y - \frac{N}{2}\right) \cdot 16 \\ &= \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \end{aligned}$$

- How to use this Laplacian in unsharp-masking?

  Unsharp masking is reversing the diffusion or adding the high-frequent parts back that have been removed by a blurring filter (e.g. by a convolution with Gaussian kernel). Since the Laplacian represents the negative high-pass filter, we have to subtract the Laplacian from the blurred image in order to add the high frequent parts back to the image. Let us assume, that a blurred image u is the result of a convolution of the original image

f with a Gaussian kernel G, i.e. $u = G * f$. Using the Laplacian from above and the blurry image u, we can try to solve for f:

$$\begin{aligned} L * f &= [G - \delta(0,0)] * f \\ &= G * f - \delta(0,0) * f \\ &= u - f \\ \Leftrightarrow f &= u - L * f \\ &\approx u - \alpha \cdot L * u \\ &= (\delta(0,0) - \alpha \cdot L) * u \end{aligned}$$

Since we cannot compute $L * f$, we approximate it with $\alpha \cdot L * u$. For the sake of readability, we once again do not scale the Laplacian but the Dirac (with $C = \frac{1}{\alpha}$). We can then write this in terms of discrete convolution masks:

$$H(i,j) = C \cdot \delta\left(x - \frac{N}{2}, y - \frac{N}{2}\right) - L(i,j) = \begin{bmatrix} -1 & -2 & -1 \\ -2 & C+12 & -2 \\ -1 & -2 & -1 \end{bmatrix}$$

The constant C must be chosen to regulate the amount of reversed diffusion. The result of this method is demonstrated in Figure 14.

Figure 14: *Unsharp masking improves the perceived sharpness of the image. (left) smoothed input image (right) result after unsharp masking.*