Signal & Image Processing 2016

Assignment 2 Histograms & Filtering

Nikolai Friis Østergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

Computer Science
University of Copenhagen

February 18, 2016

1. Histogram based processing

1.1.

In mathematical terms, the *Cumulative Density function* (CDF) with respect to (wrt.) to the *Probability density function* (PDF) from the continuous point of view is the definite integral from negative infinity to each value of the variable of the function wrt. the variable:

$$cdf(x) = \int_{-\infty}^{x} pdf(x) \quad \mathrm{d}x \tag{1}$$

In general, we know that no such thing as a negative probability exists, thus $pdf(x) \geq 0 \quad \forall x$. This means that whatever infinitesimally small part of the area under the function will be nonnegative as well.

Integrating nonnegative parts ensures that the slope of the derivative is nonnegative, in other words $cdf(x)$ is non-decreasing.

1.2.

For a constant image, in a discrete setting where $x, c \in \mathbb{N}$, all pixels contain the same value constant denoted c, thus the probability of that specific value is exactly 1, and the rest is 0, as distribution functions are normalised. It will resemble a single spike when plotted (see Figure 1a), and somewhat resembles the normalised continous 1d gaussian function, where standard deviation $\sigma \to 0$, and $x = c$.

$$pdf(x) = \begin{cases} 1 & \text{if } x = c \\ 0 & \text{otherwise} \end{cases} \qquad cdf(x) = \begin{cases} 1 & \text{if } x \geq c \\ 0 & \text{if } x < c \end{cases} \tag{2}$$

The *cdf* resembles the 'sigma' function, 'centered' at $x = c$. It generally looks like a 'ski-hill' or a tilted, stretched 'S', but in the constant image it is is a function which increases from zero to one in one step at $x = c$ (Figure 1b).

The CDF corresponding to a constant PDF (Figure 1c) is a linear function which increases from zero to one between the minimum and maximum values of the function variable (Figure 1d).

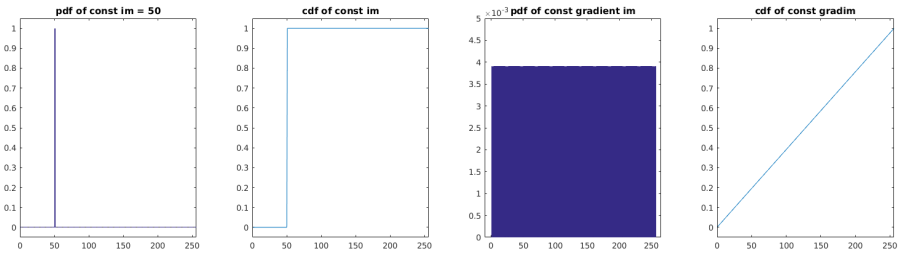

Figure 1: Normalised PDF and CFD in constant / constant gradient greyscale image

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.3.

The regions of fast increase of the function corresponds to high counts of pixel intensities and the flat regions corresponds to low counts (or no counts) of pixel intensities.

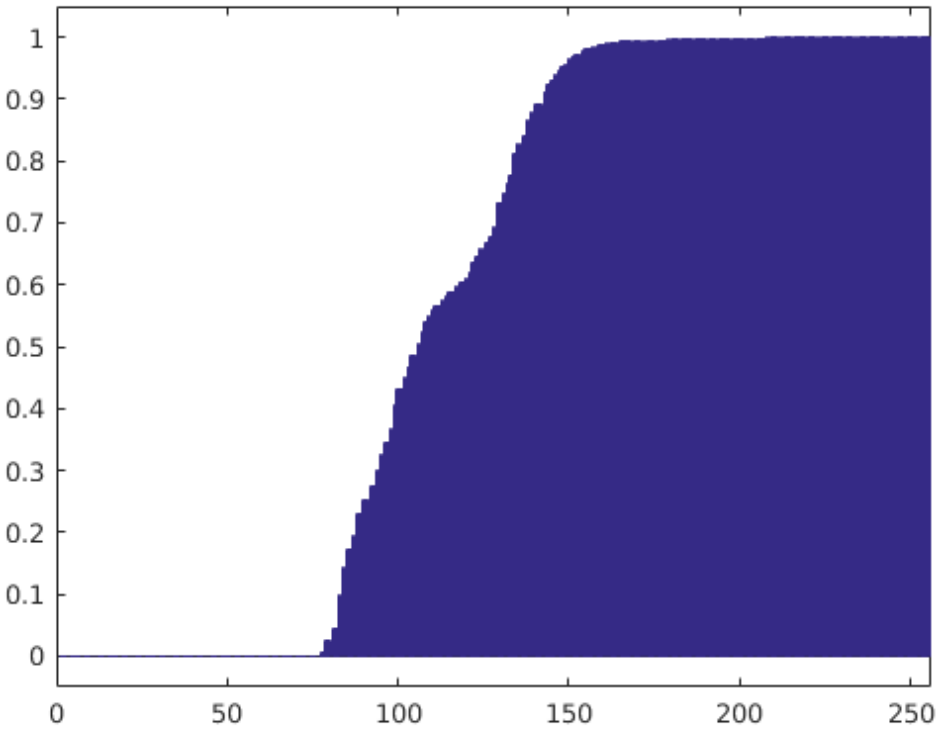

Figure 2: Normalised Cumulative Histogram (cdf(h))

```
function [ cs ] = cdf(pdf)
  % pdf is a histogram
  c = double(cumsum(pdf));
  cs = c / c(end);
end

function cs = cumhist(fname)
  I = imread(fname);  % load image
  h = imhist(I);      % get histogram
  ncs = cdf(h);       % calculate 'normalized cumulative sum'
  % display the resulting array
  bar(ncs); axis([0 256 -0.05 1.05]);
end
```

Listing 1 : 'Code to display Cumulative Histogram'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.4.

The tricky part here is really to remember that images are stored laid out row-major in matlab, this means that our perception of I(x,y) has to be queried like I(y,x) in matlab.

Running the code will open a figure, that may be queried multiple times, with the answer will be stated in the title. The result of the calculation has the same effect as asking 'how big a part of the pixels are darker than the one selected?' This means when pressing a relative dark pixel in the image, the result is low (near 0.0), and when pressing a relatively bright pixel, the result will be high (near 1.0).

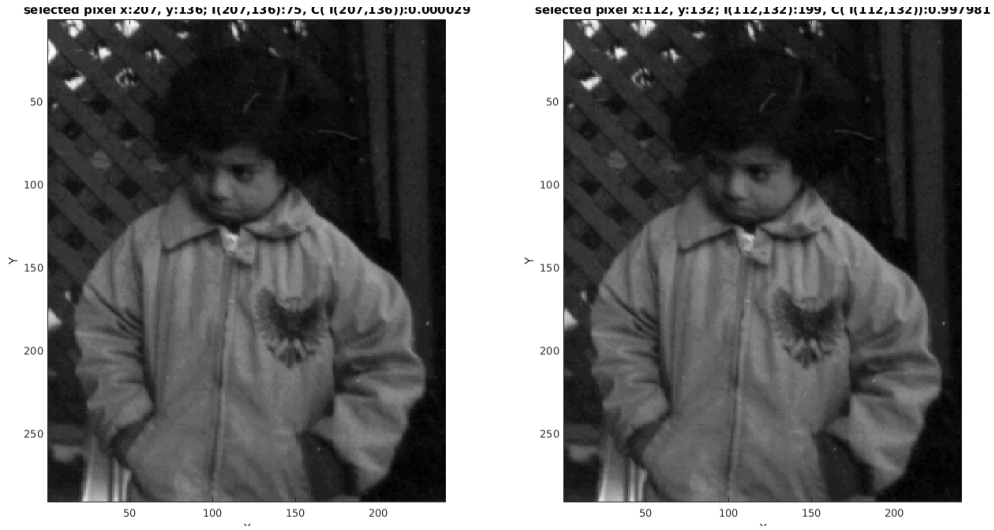

Figure 3: Select a pixel, display intensity and the part this pixel is brighter than

```
function brighterthan(fname)
  I = imread(fname); h = imhist(I); ncs = cdf(h);
  fmt = 'selected pixel x:%d, y:%d; I(%d,%d):%d, C( I(%d,%d)):%f';
  hFig = figure(1); set(hFig, 'Position', [20 20 750 750]);
  imagesc(I); axis image; colormap(gray); xlabel('X'); ylabel('Y');
  while true
    [x,y] = ginput(1);
    x = round(x); y = round(y);
    s = sprintf(fmt,x,y,x,y,round(I(y,x)),x,y,ncs(I(y,x)))
    title(s);
end
```

Listing 2 : 'Code to display Cumulative Histogram'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.5.

The CDF is not invertible in general, because the inverse isn't a function. Take the example from Figure 2, because it has a vertical line, it cannot pass the vertical line test. Also, one can see that as no values below the constant exist, $cdf(v) = 0$, for $0 < v < c$ and $cdf^{-1}(v) = \frac{1}{0}$ and division by zero is an illegal operation.

The pseudo inverse proposed in the assignment text is implemented as follows:

```
function [ pseudo_inverse ] = cdfpinv( cdf, l )
  % find the cdf values larger than l, and return the least of them
   pseudo_inverse = min( find(cdf >= l) );
end
```

Listing 3 : 'cdf pseudo-inverse implementation'

1.6.

The *histmatch* procedure is implemented in matlab, and the results are shown in Figure 4 and Figure 5.

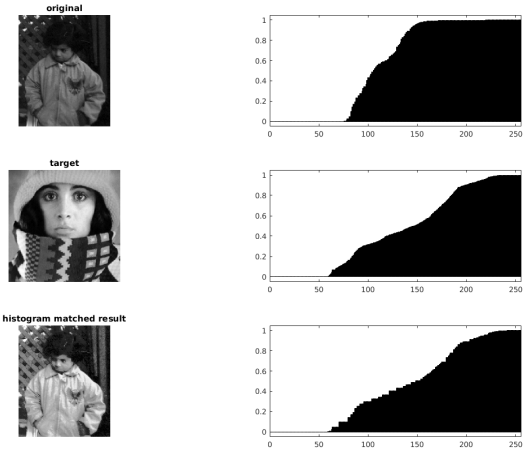

Figure 4: Image 'pout.tif' matched with the histogram characteristics of the 'trui.png' image

```
function [ Ix_matched_to_Iz ] = histmatch(I_x, I_z)
  Cx = cdf(pdf(I_x));
  Cz = cdf(pdf(I_z));
  Czinv = uint8(pinv( Cz, Cx ));
  Ix_matched_to_Iz = applymapping(I_x, Czinv);
end
im         = imrea\tilde{I}_1d('images/pout.tif');
targetim   = imread('images/trui.png');
matched    = histmatch(im, targetim);
```

Listing 4 : 'cdf histmatch implementation'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.7.

The histmatch function applied to the `pout.tif` image and the constant histogram. It seems that the effect is equal to the matlab *histeq* build-in function.

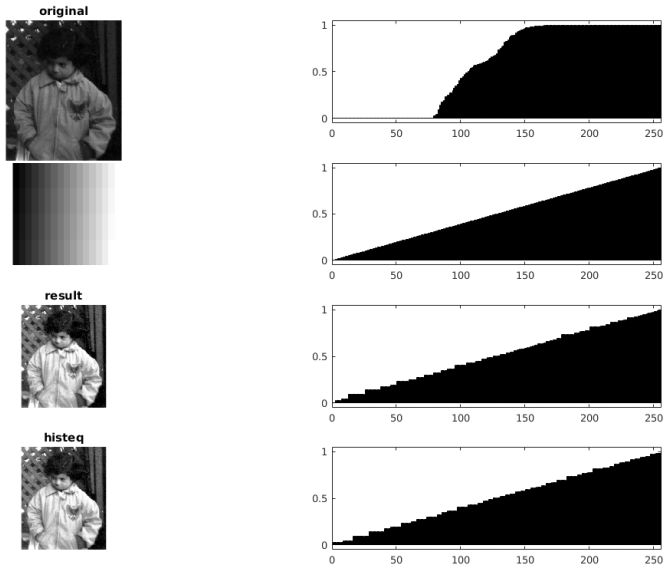

Figure 5: Image 'pout.tif' matched with uniform pdf image, and compared to matlab histeq

```
im          = imread('images/pout.tif');
targetim    = reshape(uint8(0:255),16,16);  % constant gradient (histogram)
matched     = histmatch(im, targetim);
he          = histeq(im);

plots...
```

Listing 5 : 'application of histmatch to constant histogram image'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.8.

1.8.1.

Remeber definition on pseudo inverse: $C^{-1}(l) = min\{\, s|_{C(s)\geq l}\}$ for $s \in \{0, .., 255\}$

$$\tilde{I}_1 = \phi(C_1(I_1)) \qquad \tilde{I}_2 = \phi(C_2(I_2)) \tag{3}$$

$$\tilde{I}_1 = \frac{1}{2}\left(C_1^{(-1)}(C_1(a)) + C_2^{(-1)}(C_1(a))\right) \qquad \tilde{I}_2 = \frac{1}{2}\left(C_1^{(-1)}(C_2(b)) + C_2^{(-1)}(C_2(b))\right) \tag{4}$$

$$\tilde{I}_1 = \frac{1}{2}\left(a + \begin{cases} 0 & \text{if } a < b \\ b & \text{if } b \leq a \end{cases}\right) \qquad \tilde{I}_2 = \frac{1}{2}\left(\begin{cases} 0 & \text{if } b < a \\ a & \text{if } a \leq b \end{cases} + b\right) \tag{5}$$

$$\tilde{I}_1 = \begin{cases} \frac{a}{2} & \text{if } a < b \\ \frac{a+b}{2} & \text{if } b \leq a \end{cases} \qquad \tilde{I}_2 = \begin{cases} \frac{b}{2} & \text{if } b < a \\ \frac{a+b}{2} & \text{if } a \leq b \end{cases} \tag{6}$$

It seems I have somehow failed to prove Imagethat $\tilde{I}_1 = \tilde{I}_2$ even if the images I_1 and I_2 are constant.

1.8.2.

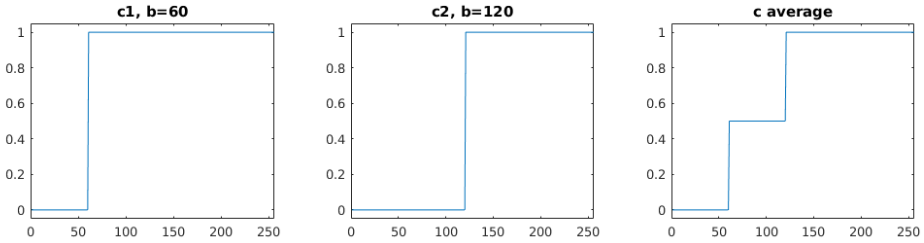

Figure 6: Images with averaged cumulative histograms

Assuming application of histmatch to constant histogram that $\tilde{I}_1 = \tilde{I}_2$, Figure 6 shows that the cumulative histogram of the midway specifications cannot be equal to the average of the cumulative histograms. The reason is that there is clearly 2 'jumps' on the average cdf, and there would be only one jump if $\tilde{I}_1 = \tilde{I}_2$.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

1.9.

Figure 7 shows the two original images in the top row, and it is seen that the leftmost seems darker, which is confirmed by the cdf displayed underneath the image. In the second row, the two images are 'maidwayed' with respect to the other, and seeming at a similar intensity level on visual inspection, which is confirmed by the cdf's below, which are both very similar, and both looks like a midway compromise of the two top ones.

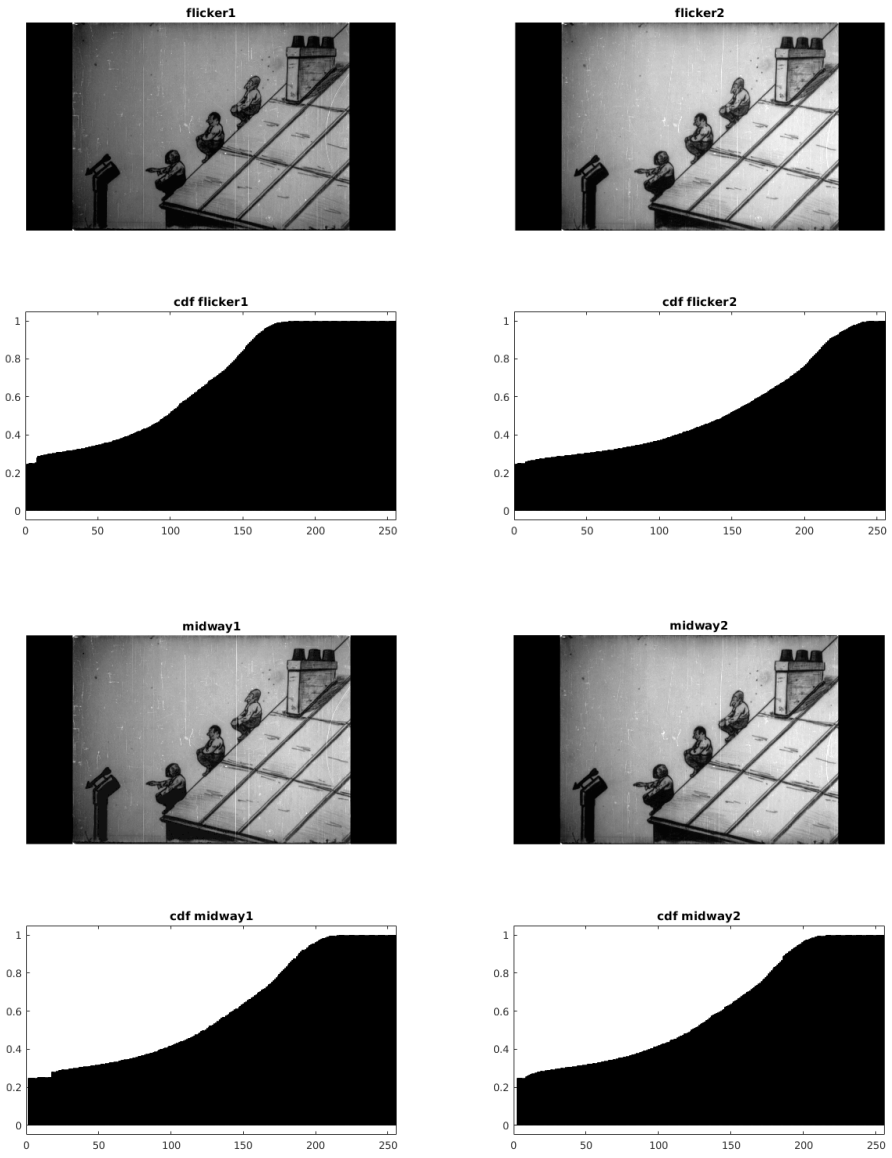

Figure 7: Original image frames, and midwayed image frames

Figure 8 shows the absolute difference in the images, which is of course big along the edges of movement in the two different frames, but further it is evident that the intensity of the original frames differ very much on average, and this difference is almost totally gone in the 'midway'ed' imageset.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

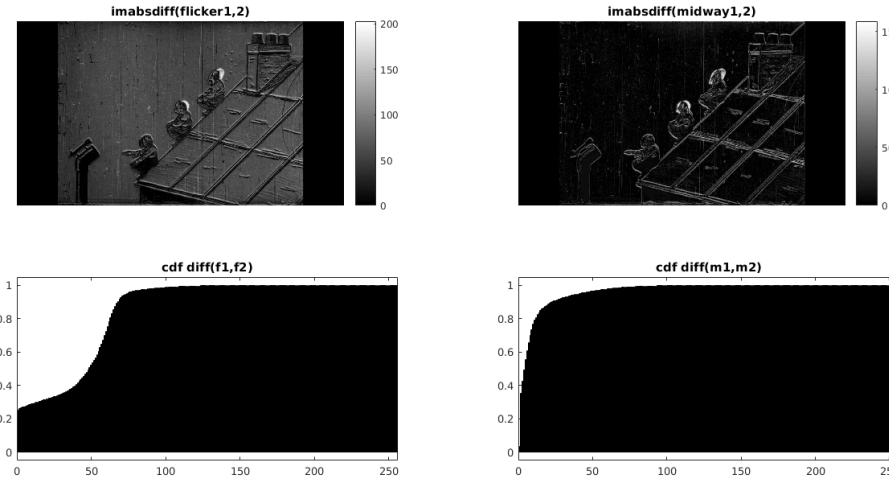

Figure 8: Absolute difference of originals vs midwayed image frames

To generalize the midway equalization, to an arbitrary number n of images, it is written:

$$\phi = \frac{1}{n}\sum_{i=1}^{n} C_i^{(-1)}$$

The matlab function IS generalized, it currently returns an midway'ed version of the first image argument, 'averaged' with them all.

```
function out = midway(Is)
  % general midway function, taking up to n images, usage midway({im_1, .., im_n})
   phi = zeros([1 256]);   n  = length(Is);
   I   = Is{1};            CI = cdf(pdf(I));
   for i=1:n
       C    = cdf(pdf(Is{i}));
       Cinv = pinv(C, CI);
       phi  = phi + Cinv;
   end
   phi = uint8( phi / n );
   out = applymapping(I, phi);
end

im3  = midway({im1, im2});
im4  = midway({im2, im1});

d12 = imabsdiff(im1,im2);
d34 = imabsdiff(im3,im4);
```

Listing 6 : 'Midway function and its application.'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

2. Image filtering and enhancement

2.1.

The approximation for the x value uses the difference between the pixel before and the pixel after. The y value is calculated the same way. This results in the kernel:

$$K = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

The imfilter function can either use correlation or convolution, the default is correlation. Correlation and convolution are almost the same, but convolution flips the kernel, on both axis. So the resulting kernel for the simle approximation approach with convolution is

$$K = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

For kernels like gaussians, which are symmetrical around the center, it does not matter which we choose to use, since kernels that will run over the image for the correlation and convolution will be the same.

2.2

The kernel in 4.5.2.1 shows the kernel for the x derivative, and thus these will be used for discussion. Instead of just looking at the pixel to the right and the left, we also use the information about the row above and below. By using the row above and below, we don't interpret a single noise pixel as being part of an edge, since the pixels below and above does not indicate that there is a continued edge. This makes noise get lower values than real edge, which spanned multiple rows. The same logic is applied for the y derivative, where we look at the derivatives and both sides. Sobel weighs the current row's pixels higher the the ones below and above. Where Prewitt weighs them equally.

2.3.

Figure 9 shows the eight.tif image with both salt and pepper noise, and gaussian noise, filtered with mean and median filter. The first column shows the image with salt and pepper noise with a mean filter, the second column is the image with gaussian noise filtered with mean filter. The third column shows the image with salt and pepper noise, filtered with a median filter and the fourth column shows the image with gaussian noise, with a median filter. The first row has a windows size of 3, the second window size of 5 and the third a window size of 7. Looking at the first column we can see how increasing windows size improves the denoising, but also the edge blur a bit. In the second column we see again how increasing the windows size improves the denoising. For the median filter the salt and pepper in column 3 is as good as gone in the first image, with windows size 3, increasing the window size only blurs the image a bit more, and makes the background brighter. In column 4 with gaussian noise we see that the increase in window size improves the denoising, while blurring the image, about as much as with the mean filter.

Figure 10 shows the computational cost for increasing window sizes. The blue line is median filter and the red is mean filter. As we can see, the median filter is generally more expensive to use, compared to the mean filter. Another thing to notice is that the median filter has a lower computational cost when the windows size is odd. It also looks like the median filter spikes more, and increases in computational cost, when N becomes bigger, see when $N \geq 21$ on the graph, compared to the mean filter, which seems to be increasing at a steady pace.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

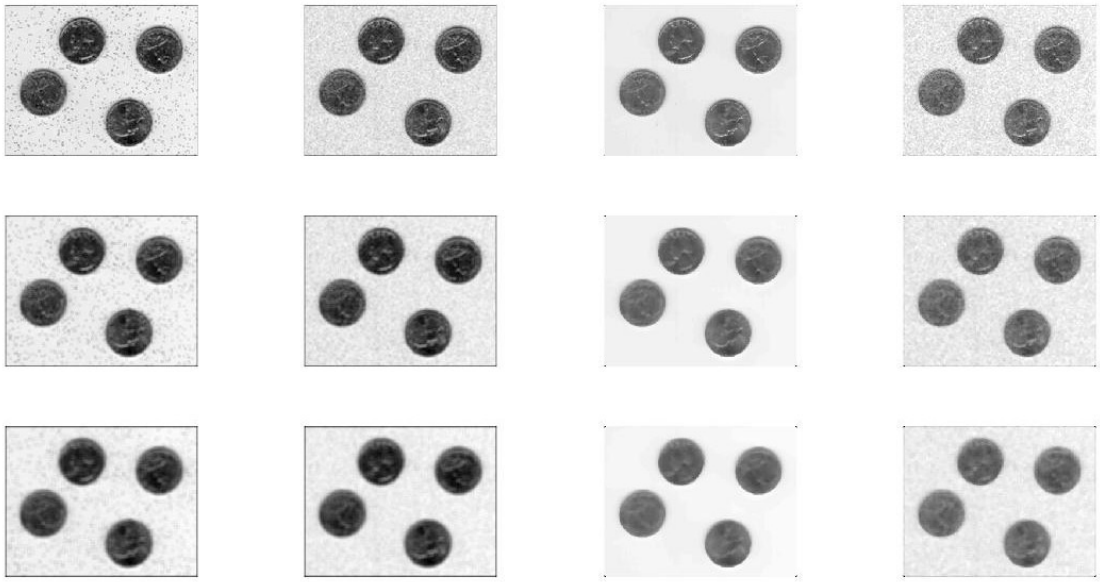

Figure 9: mean and median filter on salt/pepper noise and on gaussian noise for different window sizes

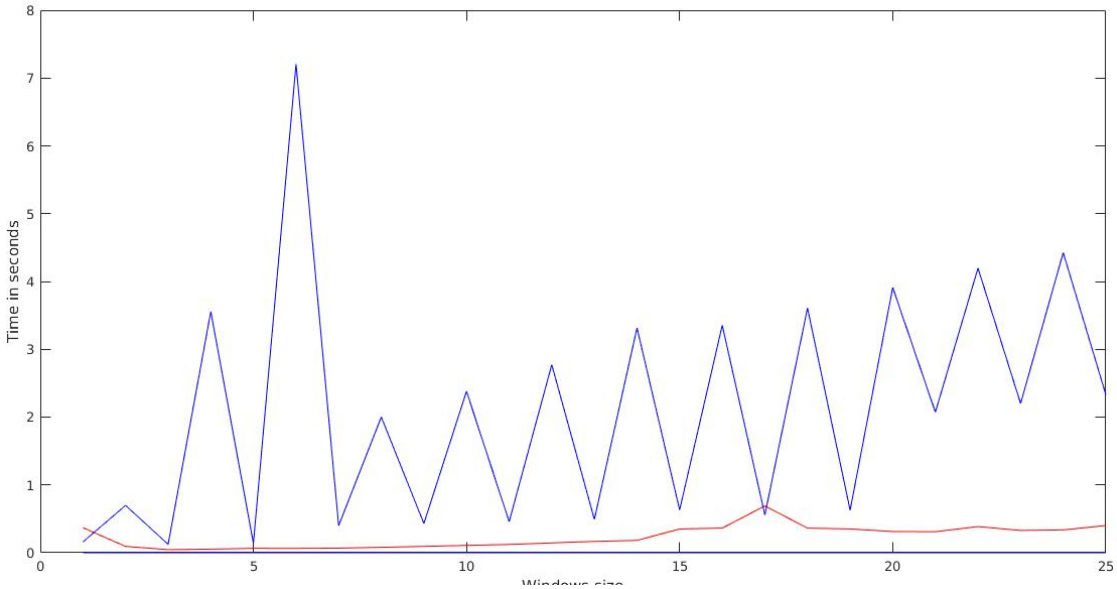

Figure 10: The computational cost of doing mean filter and median filter, based on window size.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

2.4.

Figure 11 shows the eight.tif, with salt and pepper noise, filtered with a gaussian kernel where $\sigma = 5$, with windows size in the range 3:19, with increments of 2, to keep the kernel diameter odd. We can see how the difference between image 5 and image 8, are less noticeable then the difference between image 2 and image 5. This is because with a smaller windows, the image is blurred less. At some point the windows is so big, that the values furthest away from the center are so small, that their weights are neglect able, thus increasing the windows size show no noticeable difference. And that is why its easier to see the difference between image 2 and 5 then image 5 and 8.

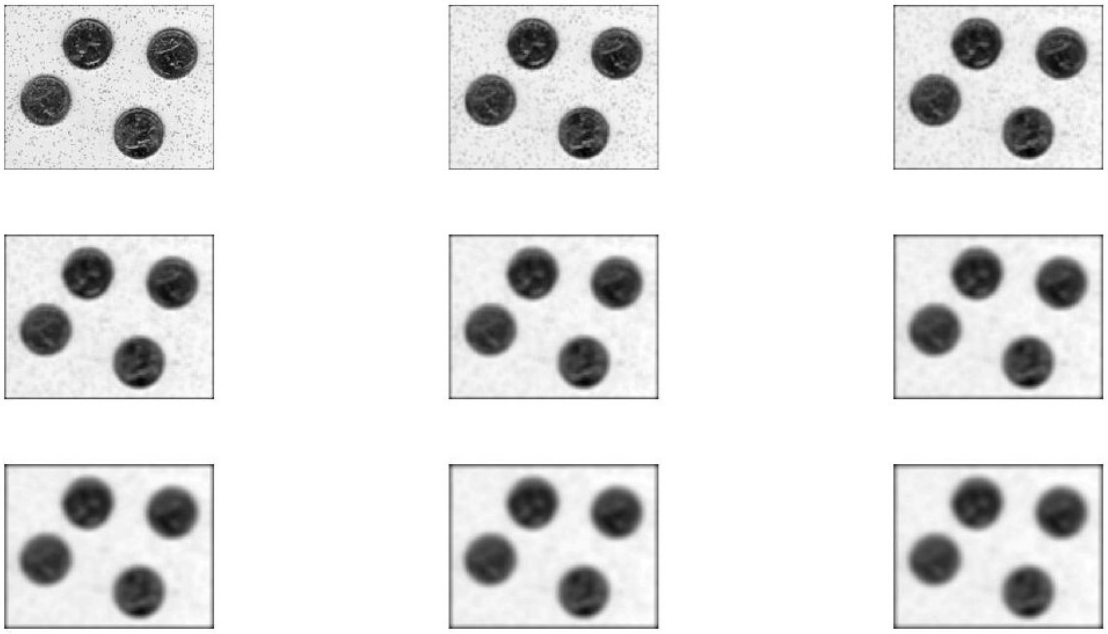

Figure 11: 9 Images with inceasing windows size starting at 3 ending at 19.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

2.5.

Figure 12 show the eight.tif, with salt and pepper noise, filtered with a gaussian filter, with increasing σ starting at 1 and ending at 17, in incremental steps of 2. The windows size N is 3*sigma + 1, and then rounding up to nearest odd integer. What we can see is that increasing σ and windows size, rapidly improves the noise reduction, but also blurs the edges to an equal extend. In image 4, the noise is as good as gone, so from there on, the only difference is just that the edges in the following images are more blurred out. So increasing the σ and windows size does makes the filter more effective at removing noise, but also makes finding edges harder. The noise removal is not something that improves the larger σ, since when sigma = 7 and window size = 23, the noise is gone, and we can't do better than remove it.

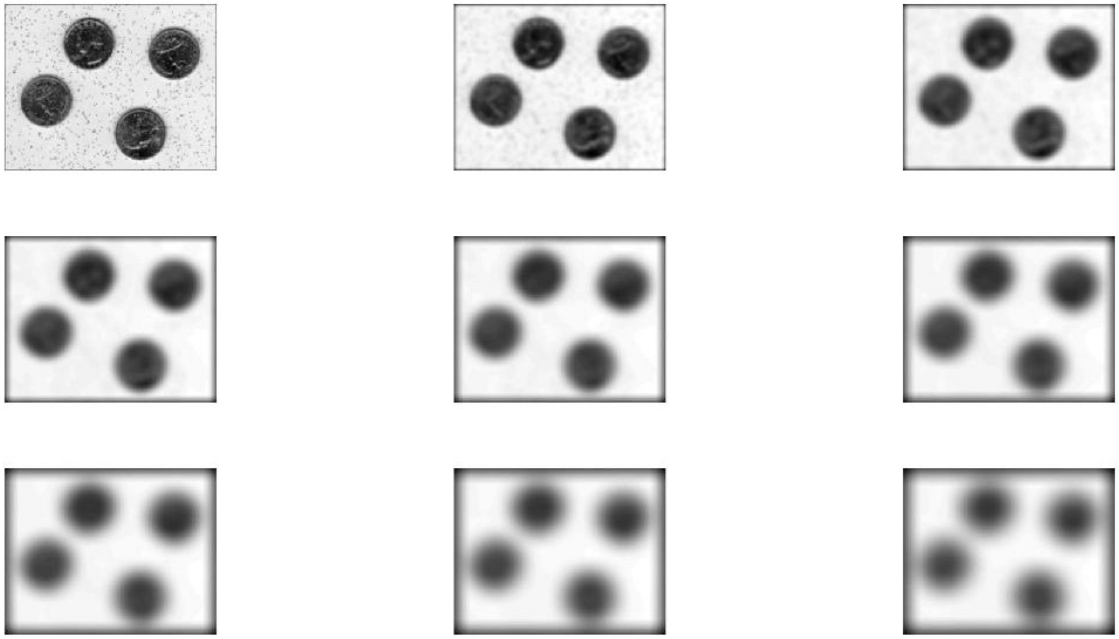

Figure 12: 9 Images with increasing sigma and windows size starting at sigma = 1 and ending at sigma = 17, with increments of 2.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

3 Bonus questions

3.1.

The bilateral filter is not linear, because of the non-linear dependency on the weights w. The term that differs from the usual Gaussian filter is the range intensity difference smoothing term g_τ. The interpretation of the τ parameter is that it is the intensity smoothing parameter, which for low values weighs intensity differences higher and for high values weighs intensity differences the same. Bilateral filtering becomes usual Gaussian filtering in the limit of τ as it approaches ∞, because g_τ approaches 1.

3.2.

```
function [ I_filtered ] = bilateral_filtering(I, N, sigma, tau)

%Padding image boundaries with N zeros.
I_padded = zeros(size(I, 1) + 2 * N, size(I, 2) + 2 * N);
I_padded(N + 1 : N + size(I, 1), N + 1 : N + size(I, 2)) = double(I);

k = fspecial('gaussian', [N N], sigma);
I_filtered = zeros(size(I));
for i = 1 : size(I, 1)
    for j = 1 : size(I, 2)
        x_i = (N + i - ceil(size(k, 1) / 2)) : (N + i + floor(size(k, 1) / 2) - 1);
        y_i = (N + j - ceil(size(k, 2) / 2)) : (N + j + floor(size(k, 2) / 2) - 1);
        w = k.*exp(-(I_padded(x_i, y_i) - I(i, j)).^2 / (2 * tau^2));
        I_filtered(i, j) = sum(sum(I_padded(x_i, y_i).*w)) / sum(sum(w));
    end
end

end
```

bilateral_filtering.m

The function first pads the image with zeroes to avoid out of bounds exceptions. It then computes the spatial gaussian filter. Finally it computes the filtered image.

3.3.

Bilateral filtering and edge detection applied to *eight.tif* corrupted by gaussian noise, for various σ and τ can be seen in figure 13. The first column is the filtered image, the second is *Sobel* edge detection, the third is *Prewitt* edge detection, and the fourth is *Zero-cross* edge detection. The image in the first row and the first first column is the original unfiltered image. $\sigma = 1$ in rows 2, 5, and 8, $\sigma = 9$ in rows 3, 6, and 9, and $\sigma = 17$ in rows 4, 7, 10. $\tau = 10$ in rows 2, 3, and 4, $\tau = 50$ in rows 5, 6, and 7, and $\tau = 90$ in rows 8, 9, and 10. The window size is three times σ.

The reduction in noise is increased as τ and σ is increased, but so is the amount of blurring. The best settings amongst those applied seems to be $\sigma = 9$ and $\tau = 50$, which correspond to row 6.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

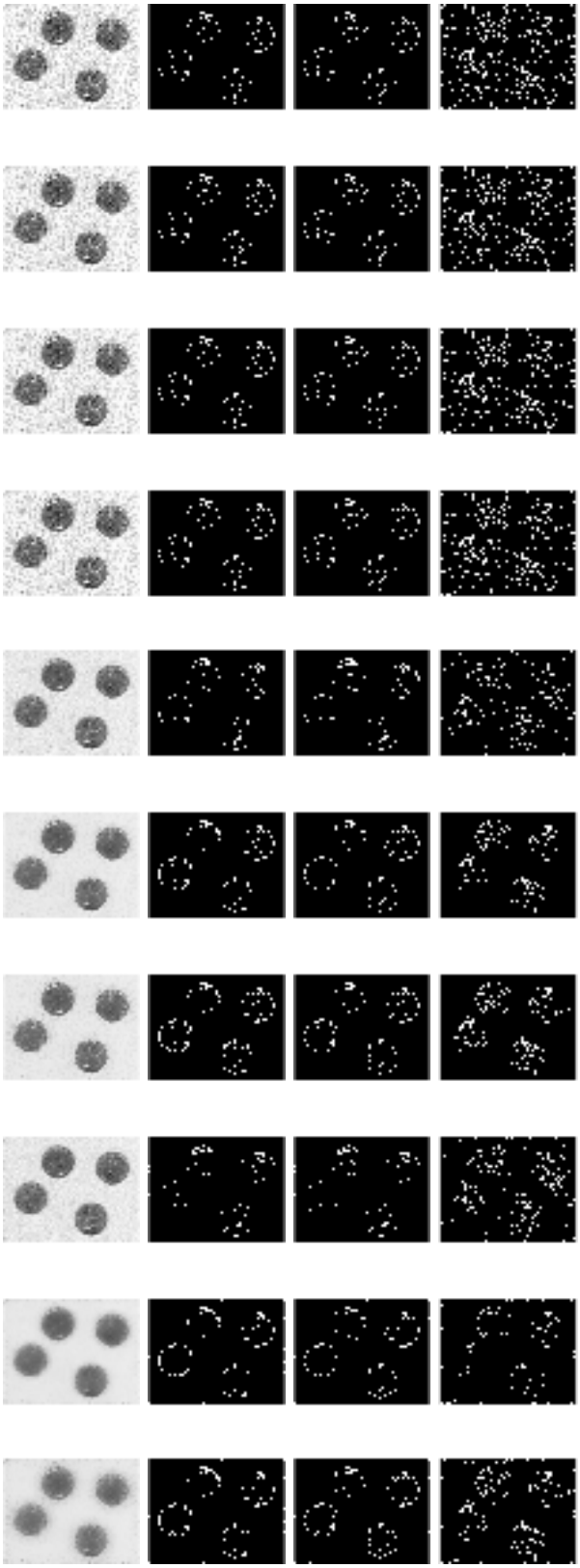

Figure 13: Bilateral filtering and edge detection applied to *eight.tif* corrupted by gaussian noise, for various σ and τ.

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947

helper methods

```
function out = applymapping(in, map)
    % apply the mapping
    [cs rs] = size(in);
    out = uint8(zeros([cs rs]));
    for r = 1:rs
        for c = 1:cs
            out(c,r) = map( in(c,r)+1 );
        end
    end
end
```

Listing 7 : 'applies a mapping'

Nikolaj Friis tergaard - ltm741
Mads Christian Thoudahl - qmh332
Michael Feveile Mariboe - njp947