1 Histogram-based processing

1.1

TODO

1.2

If the image is normalized to have values in the range [0,1], then the PDF of a constant image is just the normalized histogram. The CDF is just

$$y(x_k) = \sum_{j=0}^{k} p_x(x_k)$$

The k'th entry is the sum of all previous entries. For a constant PDF the CDF will just be a monotonically increasing, each step incremented by the same constant value.

1.3

1.4

1.5

Generally no, given an image transformed by a CDF, we have no way of knowing which intensities matched to which acummulated intensities.

1.6

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 result 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 is with convolution is

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

For kernels like gaussian, which are symetrical 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 rows pixels higher the the ones below and above. Where Prewitt weigh them equally.

2.3

Figure 1 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 iamge 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 peper 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 bluring the image, about as much as with the mean filter.

Figure 2 shows the computaional cost for increasing window sizes. Th blue line is median filter, red is mean filter. As we can see, 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

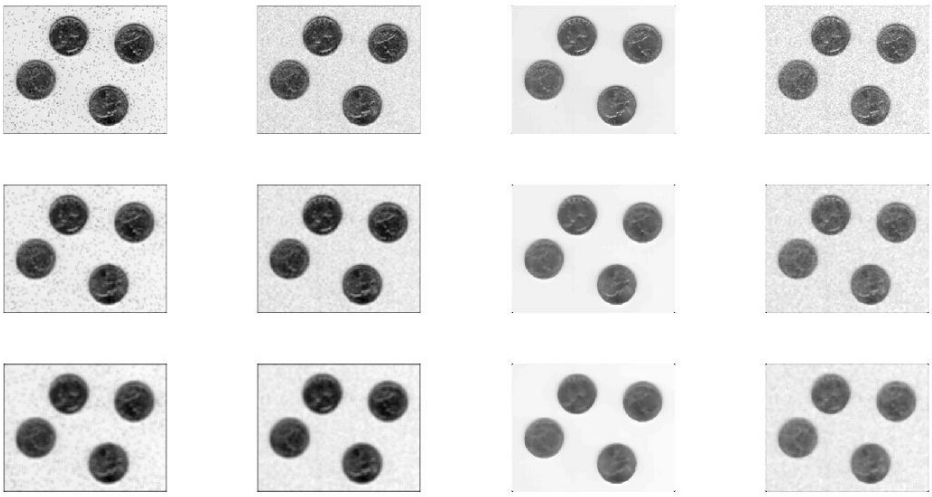

Figure 1: mean and median filter on salt/peper noise and on gaussian noise for different window sizes

increases in computaional 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.

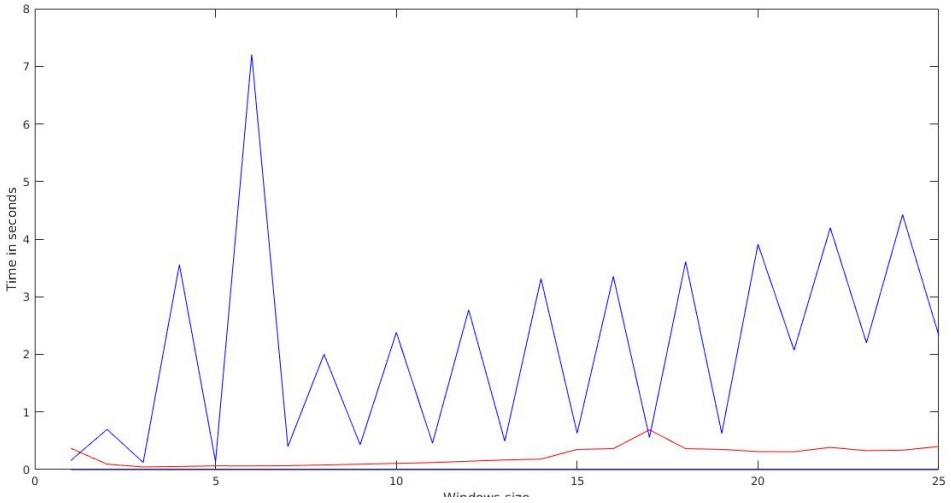

Figure 2: The computational cost of doing mean filter and median filter, based on window size.

2.4

Figure 3 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 og 2, to keep the kernel diameter odd. We can see how the difference between image 5 and image 8, are less notacible 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 weight is neglectable, thus increasing the windows size show no notacible difference. And that is why its easier to see the difference between image 2 and 5 then image 5 and 8.

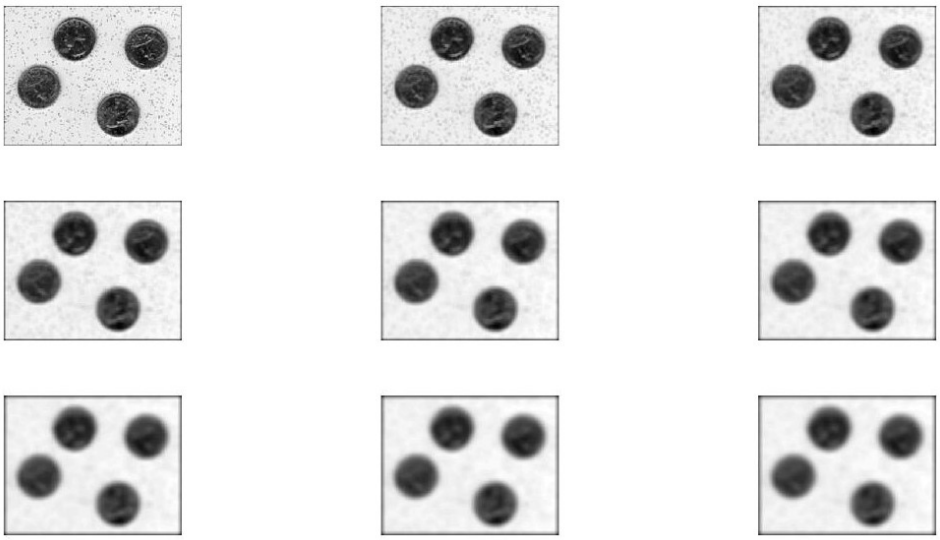

Figure 3: 9 Images with inceasing windows size starting at 3 ending at 19.

2.5

Figure 4 show the eight.tif, with salt and peper 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 that increasing σ and windows size, rapidly improves the noise reduction, but also blures 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 then remove it.

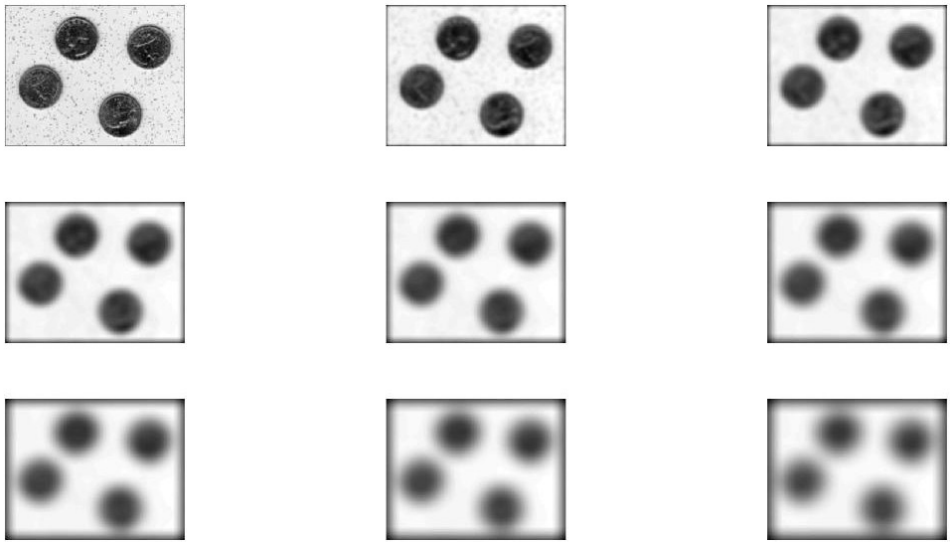

Figure 4: 9 Images with inceasing sigma and windows size starting at sigma = 1 and ending at sigma = 17, with increments of 2.

3 Bonus question

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
```

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 5. 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 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.

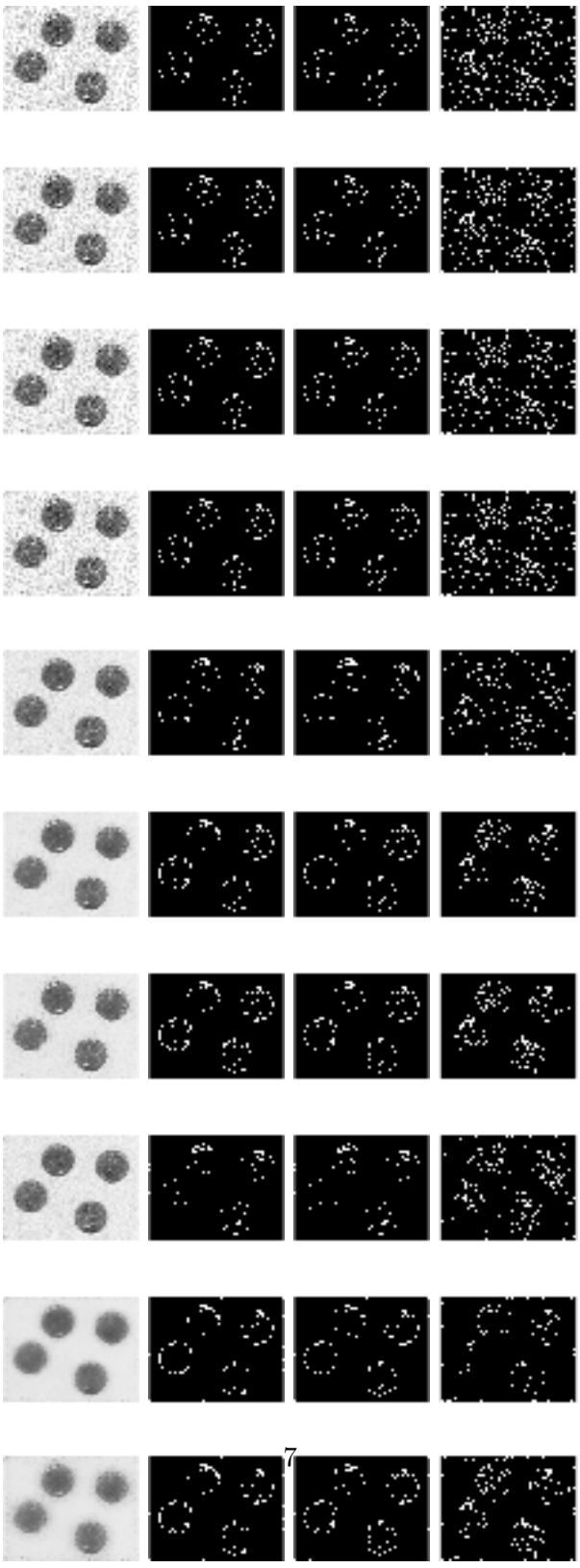

Figure 5: Bilateral filtering and edge detection applied to *eight.tif* corrupted by gaussian noise, for various σ and τ.