Image Enhancement - Strategies and Application

EQ2330 Image and Video Processing, Project 1

Jan Zimmermann jczim@kth.se

Lars Kuger kuger@kth.se

November 21, 2016

Summary

As a part of project 1 in the course EQ2330 Image and Video Processing at KTH Royal Institute of Technology in Fall Term 2016, several image enhancements techniques are presented, implemented and analyzed in this report. The used approaches are histogram equalization, spatial filtering with mean and median filter and frequency domain filtering. It will be shown that histogram equalization can improve low contrast images. Furthermore, it will be shown that mean filters perform well removing the effects of additive Gaussian noise whereas median filters deliver good results for images disturbed by salt pepper noise. Last but not least, frequency domain filtering is proven to be useful in case of blurred images.

1 Introduction

Nowadays images are a very important and present part of everyday life. This has been particularly true since digital images came up which can be used and transferred easily over the internet. Sometimes these digital images are disturbed or of bad quality though such that image enhancement is necessary. In the course of this paper three different approaches to digital image enhancement will be shown and discussed. Note that only gray scale images are considered and a 8 bit representation for the intensity value is assumed, e.g. a pixel can have a intensity value between 0 and $2^8 - 1 = 255$. Thus, a digital image can mathematically be denoted as $f(x, y)$ where the output is the intensity value at the position x and y.

Firstly, a problem in an image might be that it has very little contrast. This can be illustrated by a histogram of an image which is defined as a discrete function $h(r_k) = n_k$ where r_k represents an intensity value and n_k its respective number of occurrence in the image [2]. For an image with little contrast such a histogram will be rather narrow.

Secondly, an image might be disturbed by noise. Two common types of noise are additive Gaussian noise $n(x, y)$ and salt pepper noise. For additive Gaussian noise, each intensity value is a random number distributed according to a Gaussian distribution $N(\mu, \sigma^2)$ with mean μ and variance σ^2. The disturbed image then is $f(x, y) = s(x, y)+n(x, y)$ where $s(x, y)$ represents the undisturbed image. As opposed to that, salt pepper noise will affect an image by setting intensity values either to 0 or 255 with a probability of $P(0) = p_1$ and $P(255) = p_2$. All other intensity values will remain unchanged.

A third problem that can occur is that images are blurred. This is to say that the available image is disturbed by a degradation transfer function $H(\omega_x, \omega_y)$

and additive noise, at least quantization noise. Such a system is shown in Fig. 1. In the course of this report transfer functions will usually be given in frequency domain and denoted with a capital letter, e.g. $H(\omega_x, \omega_y)$ which corresponds to the discrete Fourier transform of the function in spatial domain $H(\omega_x, \omega_y) = \mathscr{F}\{h(x,y)\} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x,y) e^{-j2\pi(ux/M+vy/N)}$ [2].

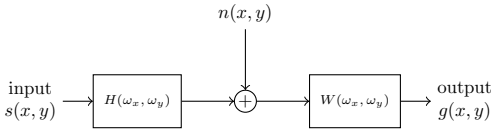

Figure 1: System with degradation transfer function $H(\omega_x, \omega_y)$ and additive noise $n(x,y)$ as well as a restoring filter $W(\omega_x, \omega_y)$.

2 System Description

The first type of images to be enhanced as stated in Section 1 are low contrast images. As aforementioned, a way to detect if an image has low contrast is to analyze the histogram, which in that case will be rather narrow, e.g. the intensity values across the image are very similar and do not use the full range from 0 to 255. Thus, in order to enhance the image the idea is to equalize the histogram such that all intensity values are used equally often. For a continuous pdf, this could be done by applying the probability integral transform which states that the transform $T(x) = \int_{-\infty}^{x} f_X(x)\ dx$ of the random variable X will exactly give a standard uniform distribution [3]. For the discrete case and including the scaling to the full range, this transform is given by

$$T(r_k) = 255 \frac{\sum_{j=0}^{k} h(r_j)}{\sum_{j=0}^{255} h(r_j)} = \frac{255}{MN} \sum_{j=0}^{k} n_k \quad k = 1, ..., 255 \tag{1}$$

where M and N denote the size of image [2]. Note that the transform in the discrete case will not necessarily produce a perfect uniform distribution.

The second type of image disturbance is noise, as mention in section 1. To reduce the two different kinds of noises, Gaussan and Salt-Pepper, two types of spatial filters were implemented. The first was a 3x3 mean filter, that is applied by convolution to the noisy image,

$$f(x,y) = h(x,y) * g(x,y) \tag{2}$$

where g(x,y) is the noisy image, f(x,y) the denoised image and h(x,y) the function of the mean filter. To achieve the lowpass characteristic of the filter, its impulse response had the structure

$$h(x,y) = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & [1] & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{3}$$

If this matrix is multiplicated to a 3x3 portion of the image, the mean value of the eight surrounding values of the center and the center itself is calculated. After that the value of the center pixel is replaced by the mean value and the shifted by an increment of one. By that, noise reduction should be achieved.
The 3x3 median filter works in a similar way, but here the center point is replaced by the median of the nine values.

In case the image is degraded not only by additive noise but also by a transfer function, Wiener deconvolution is the optimal linear solution and can be applied if the following assumptions can be made [2]. Either the noise or the image have zero mean and they are uncorrelated. Then the Wiener filter is given by

$$\hat{F}(\omega_x, \omega_y) = \frac{|H(\omega_x, \omega_y)|^2}{H(\omega_x, \omega_y) \cdot \left(|H(\omega_x, \omega_y)|^2 + \frac{\Phi_{nn}(\omega_x, \omega_y)}{\Phi_{ff}(\omega_x, \omega_y)} \right)} G(\omega_x, \omega_y) \tag{4}$$

where $\Phi_{zz}(\omega_x, \omega_y)$ is used to denote the power spectrum of $z(x, y)$. Since the $\Phi_{ff}(\omega_x, \omega_y)$ usually is unknown, the following term can, in case $n(x, y)$ is white Gaussian noise, be approximated by a constant $K \approx \Phi_{nn}(\omega_x, \omega_y)/\Phi_{ff}(\omega_x, \omega_y)$. Note that the implementation of the Wiener deconvolution might need to be handle aliasing due to circular convolution.

3 Results

In this section the results of the experiments are described. First, the outcome of the application of histogram equalization on low contrast images is laid out. After that the results of Image denoising with spatial filters is handled and in the end frequency domain filtering is discussed.

In figure 2 the effects of equalization on the histogram are depicted. For comparison the histograms of the original image "Lena.bmp", a low contrast version and an equalized one are shown. The differences are clearly visible. While the gray levels of the original image had a considerable number of occurrences over the whole scale, the low contrast image had a very high count of occurrences in a narrow part in the middle of the gray scale. To restore the contrast of the original image, the algorithm described in section 2 was performed, what results in the relating picture in figure 2. One can perceive, that now the gray levels were again distributed over the whole scale with higher values of occurrences. For the continuous case, the distribution of gray levels should be uniform. However, as a discrete approximation of the probability density function was used, the resulting histogram is not flat (see [1] p. 149).

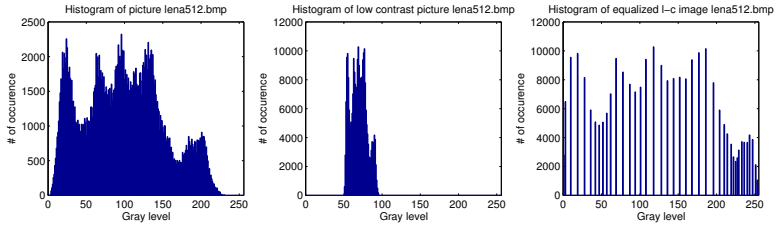

Figure 2: Histograms of image "Lena": original, low contrast and equalized histogram.

A good way of investigating the filtering effects on the noisy images is by analyzing the histograms. To do this, the image "Lena.bmp" was used again. The first picture in figure 2 shows the histogram of the clear image and in figure 3 are the histograms of noisy images and filtered images. It can be seen that the Gaussian noise has a smoothing effect on the histogram. Sharp peaks are thus reduced, what distorts the image. The salt-pepper noise leaves the original histogram almost unchanged, except of two high increases in the number of the gray level values 0 and 255, which represent the salt (white, value 255) and pepper (black, value 0) in the noise. In figure 3, it can also be seen, that when the mean and median filter are applied to gaussian noise, the smoothing effect

of the gaussian noise can be revoked, so that the histograms are more similar to the one without noise. Both filters seem to be equally useful to denoise the image with gaussian noise. For the salt-pepper noise, the difference between the application of mean or median filter is higher. For the mean filter the histogram does not show the original peaks anymore. Especially for medium values of the gray levels the histogram shows almost constant numbers. Furthermore the high numbers for black and white have vanished. The reason for that is that the black and white pixels, which are produced by the saltpepper noise, are equally spread over the whole image. By applying the mean filter an average gray level value of the surrounding pixels is calculated. By that the 0 and 255 values are averaged out, but do increase and diminish the gray level values respectively, what results in a high concentration of pixel numbers in medium gray levels.The median filter on the other hand discards the black and white values for the pixels not by averaging them with their surrounding values, but by replacing them with the median. By that the effect of the noise can be revoked, so that the histogram looks almost the same as before noising the image. By observing the resulting filtered image, it can be seen, that some of the black and white pixels of the noise remain, where the concentration of these were high in a specific area of the noisy image.

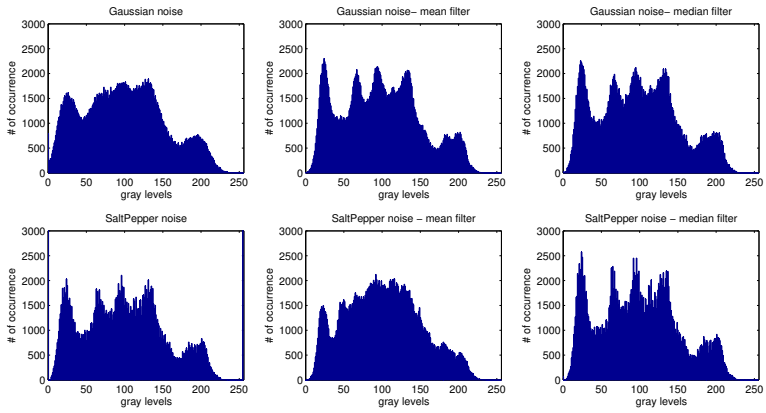

Figure 3: Histograms with applied mean and median filter.

As stated in section 2, the Wiener deconvolution can be used to enhance out of focus images. An out of focus image in frequency domain is shown in Fig. 4. The spectrum of the spatially blurred image is visibly blurred as well in frequency domain.

An exemplary result of the enhancement achieved by Wiener deconvolution is shown in Fig. 5. It is clearly visible that the Wiener deconvolution performs well and enhances the image remarkably. Note that in order to implement the Wiener deconvolution, a value for the constant K has to be chosen. Considered that the optimal value for this constant is $K = \Phi_{nn}(\omega_x, \omega_y)/\Phi_{ff}(\omega_x, \omega_y)$ which we cannot calculate since we are lacking the power spectral density of $f(x, y)$. In contrast, we are able to calculate the variance of $g(x, y)$ which allows us to make the approximation $K \approx \gamma\,[\mathrm{Var}\{n(x, y)\}/\mathrm{Var}\{g(x, y)\}]$. That this is a reasonable approximation can be understood by assuming that the the variance of the noise is rather small compared to the variance of $f(x, y)$. In order to ensure that the deconvolution filter will not grow to large, γ can be chosen to be $\gamma > 1$. Also, when filtering a zero-padded image, black and white lines might appear that are caused by the sharp edge of the image. In order to avoid this, pixels close to the edges of the originally sized images were low-pass filtered.

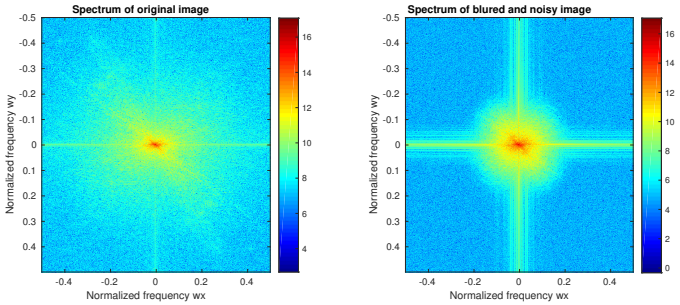

Figure 4: Spectrum of the original "Lena" image (left) and spectrum of the blurred and noisy image (right) which is noticeably blurred also in frequency domain. The DC component is located in the center of the image.

Figure 5: Image "Lena" that was blurred at first (left) and then restored using the Wiener deconvolution (right). It can be seen that the restored image is of much better quality and only the edges suffer from minor blurry appearance. $\gamma = 4$ was used.

4 Conclusions

The discrete nature of histogram equalization is the reason for its only approximate nature. That is why the original picture cannot be restored completely in most cases. Nevertheless it is a valuable tool for raising the contrast in low-contrast images. Also, the results showed that spatial image filtering is powerful tool to remove noise from images. Depending on the kind of noise, the results may vary with application of different filters. Eventually, it was shown that for a reasonable choice of the constant K and proper handling of the image edges, the Wiener deconvolution can improve blurred images remarkably.

Appendix

Who Did What

Report

1. Summary: Lars Kuger
2. Introduction: Lars Kuger
3. System Description
   (a) Part 1: Lars Kuger
   (b) Part 2: Jan Zimmermann
   (c) Part 3: Lars Kuger
4. Results
   (a) Part 1: Jan Zimmermann
   (b) Part 2: Jan Zimmermann
   (c) Part 3: Lars Kuger
5. Conclusion: Jan Zimmermann and Lars Kuger

MatLab Code

Note that for assignments 1 and 2, each of us programmed his own separate solution.

Jan Zimmermann:

- FreqDomainFilt.m
- HistogramEqualization21.m
- ImageDenoising22.m
- ImageRestoring3.m

Lars Kuger:

- assignment21.m
- assignment22l.m
- jzlk_hist.m
- jzlk_wienerFilter.m
- jzlk_wienerFilterWithoutTapering.m

MatLab code

FreqDomainFilt.m

```
% Assignment 3 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

%% Clear variables and Command Window
clc;
clear all;

%% Load Image

f = imread('lena512.bmp');

[M,N] = size(f);

%% Image Bluring and adding noise

h = myblurgen('gaussian', 8);
noise_mean = 0;
noise_var = 0.0833; % Why 32? Maybe 0.0833 instead?
noise_gaussian = mynoisegen('gaussian', M, N, noise_mean, noise_var);

% linear convolution
f_blured = conv2(double(f), h, 'same');
g = f_blured +noise_gaussian;



%% Fast Fourier Transform
F = fft2(f);
G = fft2(g);
H = fft2(h);

Fmagn = abs(F);
Gmagn = abs(G);

%% Image restoration with own function

% Restore image with Wiener Filter. The blurred image is tapered before
% being filtered
f_restored = jzlk_wienerFilter(g, h, noise_var);

% Restore image with Wiener Filter. The blurred image is zero padded before
% being filtered in order to avoid artifacts due to circular convolution
f_restored_wET = jzlk_wienerFilterWithoutTapering(g,h,noise_var);


%% Image restoration
%
% nsr = noise_var/var(double(f(:)));
```

```
% f_restored = uint8(deconvwnr(g, h, nsr));


%% Plots

fig1 = figure(1);
subplot(1,2,1);
imshow(f);
title('Original Image');

subplot(1,2,2);
imshow(uint8(g));
title('Blurred and Noisy Image');

fig2 = figure(2);
subplot(1,2,1);
imshow(uint8(g));
title('Blurred and Noisy Image');

subplot(1,2,2);
imshow(f_restored);
title('Restored Image')


fig3 = figure(3);

x = -M/2:M/2-1;
x = x/M;
y = -N/2:N/2-1;
y = y/M;

subplot(1,2,1);
imagesc(x,y,fftshift(log(Fmagn)));
colormap(jet);
colorbar
title('Spectrum of original image');
xlabel('Normalized frequency wx');
ylabel('Normalized frequency wy');


subplot(1,2,2);
imagesc(x,y,fftshift(log(Gmagn)));
colormap(jet);
colorbar
title('Spectrum of blured and noisy image');
xlabel('Normalized frequency wx');
ylabel('Normalized frequency wy');
```

HistogramEqualization21.m

```
% Assignment 2.1 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

%% Clear variables and Command Window
clc;
clear all;

%% Load Image

I = imread('images/lena512.bmp');
[M,N] = size(I);

%% Plot Histogram
fig1 = figure(1);
imhist(I);
xlabel('grey levels');
ylabel('# of occurrence');
title('original histogram');

%% low-contrast image

a = 0.2;  % 0 < a < 1
b = 50;   % 0 < b < 255(1-a)

Ilow = a*I+b;

% fig2 = figure(2);
% imshow(uint8(Ilow));

fig3 = figure(3);
imhist(Ilow);
xlabel('grey levels');
ylabel('# of occurrence');
title('low contrast histogram');

%% Histogram Equalization
num_greyval = imhist(Ilow);
p_vec = num_greyval./(M*N);

for i =1:length(p_vec)
    sum_vec =  cumsum(p_vec(1:i));
    g(i) = sum_vec(end);
end
g = g*255;

for i = 1:N
    for j = 1:M
        Ieq(i,j) = g(Ilow(i,j));
    end
end
```

```
fig4 = figure(4);
imhist(uint8(Ieq));
xlabel('grey levels');
ylabel('# of occurrence');
title('equalized histogram');
```

ImageDenoising22.m

```
% Assignment 2.2 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

%% Clear variables and Command Window
clc;
clear all;

%% Load Image

%I = imread('images/lena512.bmp');
I = imread('images/lena512.bmp');
[M,N] = size(I);

%% Noise Generation with mynoisegen

noise_gaussian = mynoisegen('gaussian', 512, 512, 0, 64);
noise_saltpepper = mynoisegen('saltpepper', 512, 512, .05, .05);

%% Apply noise to image and plot histogram

I_gaussian = double(I) + noise_gaussian;

I_saltpepper = I;
I_saltpepper(noise_saltpepper == 0) = 0;
I_saltpepper(noise_saltpepper == 1) = 255;
I_saltpepper = uint8(I_saltpepper);


%% Create mean filter and apply to noisy image

mean_filt = ones(3,3)*1/9;

I_gaussian_meanfilt = conv2(double(I_gaussian), double(mean_filt), 'same');
I_saltpepper_meanfilt = conv2(double(I_saltpepper), double(mean_filt), 'same');


%% Create median filter and apply to noisy image
I_gaussian_zeropad = I_gaussian;
I_saltpepper_zeropad = I_saltpepper;

I_gaussian_zeropad = [zeros(1,M); I_gaussian_zeropad; zeros(1,M)];
I_gaussian_zeropad = [zeros(M+2,1), I_gaussian_zeropad, zeros(M+2,1)];

I_saltpepper_zeropad = [zeros(1,M); I_saltpepper_zeropad; zeros(1,M)];
I_saltpepper_zeropad = [zeros(M+2,1), I_saltpepper_zeropad, zeros(M+2,1)];

% while round(length(I_gaussian_zeropad)/3) ~= length(I_gaussian_zeropad)/3
%     zero_pad = zeros(1,length(I_gaussian));
%     I_gaussian_zeropad = [I_gaussian, zero_pad'];
%     I_gaussian_zeropad = [I_gaussian_zeropad; [zero_pad 0]];
```

```
%
%     I_saltpepper_zeropad = [I_saltpepper, zero_pad'];
%     I_saltpepper_zeropad = [I_saltpepper_zeropad; [zero_pad 0]];
% end

for i = 2:1:length(I_gaussian_zeropad)-1
    for j = 2:1:length(I_gaussian_zeropad)-1
        grey_val_gaussian = [];
        grey_val_saltpepper = [];


        for k = -1:+1
            for l = -1:+1
                grey_val_gaussian = [grey_val_gaussian, I_gaussian_zeropad(i+k, j+l)];
                grey_val_saltpepper = [grey_val_saltpepper, I_saltpepper_zeropad(i+k, j+l)
            end
        end

        med_gaussian = median(grey_val_gaussian);
        med_saltpepper = median(grey_val_saltpepper);

        I_gaussian_medfilt(i,j) = med_gaussian;
        I_saltpepper_medfilt(i,j) = med_saltpepper;
    end
end

I_gaussian_medfilt = I_gaussian_medfilt(2:end, 2:end);
I_saltpepper_medfilt = I_saltpepper_medfilt(2:end, 2:end);

fig1 = figure(1);

subplot(2,3,1)
imhist(uint8(I_gaussian));
title('Gaussian noise');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])


subplot(2,3,2)
imhist(uint8(I_gaussian_meanfilt));
title('Gaussian noise- mean filter');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])


subplot(2,3,3)
imhist(uint8(I_gaussian_medfilt));
title('Gaussian noise- median filter');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])
```

```
subplot(2,3,4)
imhist(I_saltpepper);
title('SaltPepper noise');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])


subplot(2,3,5)
imhist(uint8(I_saltpepper_meanfilt));
title('SaltPepper noise - mean filter');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])


subplot(2,3,6)

imhist(I_saltpepper_medfilt);

title('SaltPepper noise - median filter');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])




fig2 = figure(2);

subplot(2,3,1)
imshow(uint8(I_gaussian));
title('Gaussian noise');

subplot(2,3,2)
imshow(uint8(I_gaussian_meanfilt));
title('Gaussian noise- mean filter');

subplot(2,3,3)
imshow(uint8(I_gaussian_medfilt));
title('Gaussian noise- median filter');

subplot(2,3,4)
imshow(I_saltpepper);
title('SaltPepper noise');

subplot(2,3,5)
imshow(uint8(I_saltpepper_meanfilt));
title('SaltPepper noise - mean filter');

subplot(2,3,6)
imshow(I_saltpepper_medfilt);
```

```
title('SaltPepper noise - mean filter');

fig3 = figure(3);
subplot(3,1,1)
imhist(I);
title('histogram of clear image');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])

subplot(3,1,2);
imhist(uint8(I_gaussian));
title('histogram of image with gaussian noise');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])

subplot(3,1,3);
imhist(I_saltpepper);
title('histogram of image with salt-peper-noise');
xlabel('grey-scale values');
ylabel('#');
axis([0 256 0 3000])

fig4 = figure(4);
subplot(1,2,1)
imshow(uint8(I_gaussian));
title('Gaussian noise');
subplot(1,2,2)
imshow(uint8(I_saltpepper));
title('SaltPepper noise');
```

ImageRestoring3.m

```
% Assignment 3 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

%% Clear variables and Command Window
clc;
clear all;

%% Load Image

%g = imread('man512_outoffocus.bmp');
g = imread('boats512_outoffocus.bmp');

[M,N] = size(g);

%% Bluring and Noise

h = myblurgen('gaussian', 8);

noise_var = 0.0833;


%% Image restoration
% Restore image with Wiener Filter. The blurred image is edgetapered before
% being filtered

f_restored = jzlk_wienerFilter(g, h, noise_var);

%% Plots

fig3 = figure(3);
subplot(1,2,1);
imshow(uint8(g));
title('blured and noisy image');

subplot(1,2,2);
imshow(uint8(f_restored));
title('restored image');
```

assignment21.m

```
% Assignment 2.1 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

% pick one image from directory images
picName = 'lena512.bmp';

% read image and make it a vector
orgimg  = imread(picName, 'bmp');
[N, M] = size(orgimg);
imVec      = orgimg(:);


%% create histogram

% specify bin edges
figure;
subplot(2,2,1);
imhist(imVec);
title(sprintf('Histogram of picture %s', picName));
xlabel('Gray level');
ylabel('# of occurence');

%% simulate low contrast image

a = 0.2;
b = 50;

% check if a and b are valid
if a<=0 || a >= 1
   fprintf('Error: a must be in the range 0 to 1 but a=%f', a);
   return;
end
if b<=0 || b >= 255*(1-a)
   fprintf('Error: b must be in the range 0 to 255*(1-a) but b= %f', b);
   return;
end

% g(x, y) = min(max(a  f (x, y) + b, 0), 255)
% max and min operations as well as roundings are automatically done by
% matlab since img is in uint8 format
% to test this, just uncomment the following line
% diff = round(a*double(orgimg)+b) - double((a*orgimg+b));
lcimg = a*orgimg + b;
lcimVec = lcimg(:);

% plot the corresponding histogram
subplot(2,2,2)
imhist(lcimVec);
lchvalues = imhist(lcimVec);
title(sprintf('Histogram of low contrast picture %s', picName));
```

```
xlabel('Gray level');
ylabel('# of occurence');


%% Implement histogram equalization

% probability density function and cumulative distribution function
imgpdf = 1/(N*M) * lchvalues;
imgcdf = cumsum(imgpdf);
cdf8bit = imgcdf*(2^8 -1);

% equalization
eqimg = zeros(N, M, 'uint8');
for xx=1:N
   for yy=1:M
       eqimg(xx, yy) = cdf8bit(lcimg(xx,yy));
   end
end

% plot the corresponding histogram for the equalized image
subplot(2,2,3)
imhist(eqimg(:));
eqvalues = imhist(eqimg(:));
title(sprintf('Histogram of equalized l-c image %s', picName'));
xlabel('Gray level');
ylabel('# of occurence');


%% Plot the images

% plot original image, low contrast image and equalized low contrast image
figure;
subplot(2,2,1)
imshow(orgimg);
title(sprintf('Original image %s', picName));
subplot(2,2,2);
imshow(lcimg);
title(sprintf('Low contrast image %s', picName));
subplot(2,2,3);
imshow(eqimg);
title(sprintf('Equalized l-c image %s', picName));
```

assignment22l.m

```
% Assignment 2.2 for Project 1 in EQ2330
% Fall Term 2016
% Course EQ2330 Image and Video Processing
% Project 1
% Authors: Jan Zimmermann, Lars Kuger

% pick one image from directory images
picName = 'lena512.bmp';

% read image
orgimg  = imread(picName, 'bmp');
[M, N] = size(orgimg);

%% Generate Noise and Disturb Images

% Gaussian distributed noise
sigma2 = 64;
mu = 0;
gaussnoise = mynoisegen('gaussian', M, N, mu, sigma2);
im_gaussn = double(orgimg) + gaussnoise;
im_gaussn = uint8(im_gaussn);

% Salt pepper noise
im_saltp = orgimg;
n = mynoisegen('saltpepper', 512, 512, .05, .05);
im_saltp(n==0) = 0; % set the gray value to 0 at the positions where n==0
im_saltp(n==1) = 255;

%% Plot images and histograms

histfig = figure;

% Histogram of original image
subplot(3,3,1);
jzlk_hist(orgimg, 'Histogram without noise');

% Histogram with Gaussian noise
subplot(3,3,2);
jzlk_hist(im_gaussn, 'Histogram with Gaussian Noise');

% Histogram with salt pepper noise
subplot(3,3,3);
jzlk_hist(im_saltp, 'Histogram with salt pepper noise');


imfig = figure;

% image with Gaussian noise
subplot(2,3,1);
imshow(im_gaussn);
title('Image with Gaussian Noise');

% image with salt peppers noise
```

```
subplot(2,3,4);
imshow(im_saltp);
title('Image with Salt Pepper Noise');


%% Mean filter

% mean filter
mfilter = 1/9*ones(3);

% conv2 will zero pad automatically and return only relevant part when
% 'same' is given as a parameter
im_gaussn_mfilt = uint8(conv2(double(im_gaussn),mfilter, 'same'));
im_saltp_mfilt = uint8(conv2(double(im_saltp), mfilter, 'same'));

% histograms
figure(histfig);

subplot(3,3,5);
jzlk_hist(im_gaussn_mfilt, 'Histogram - Gaussian Noise - Filtered');

subplot(3,3,6);
jzlk_hist(im_saltp_mfilt, 'Histogram - Salt Pepper Noise - Filtered');

% images
figure(imfig);

subplot(2,3,2);
imshow(im_gaussn_mfilt);
title('Gaussian Noise - Mean');

subplot(2,3,5);
imshow(im_saltp_mfilt);
title('Salt Pepper - Mean');


%% Median filter

% Apply 3x3 median filter, zero padding is automatically done
im_gaussn_median = medfilt2(im_gaussn);
im_saltp_median = medfilt2(im_saltp);

% histograms
figure(histfig);

subplot(3,3,8);
jzlk_hist(im_gaussn_median, 'Histogram - Gaussian Noise - Median');

subplot(3,3,9);
jzlk_hist(im_saltp_median, 'Histogram - Salt Pepper - Median');

% images
figure(imfig);
```

```
subplot(2,3,3);
imshow(im_gaussn_median);
title('Gaussian Noise - Median');

subplot(2,3,6);
imshow(im_saltp_median);
title('Salt Pepper - Median');
```

jzlk_hist.m

```
function [h] = jzlk_hist( image, ptitle )
%Creates histogram of image

h = histogram(image(:), 0:256);
title(ptitle);
xlabel('Gray level');
ylabel('# of occurence');

end
```

jzlk_wienerFilter.m

```
function [ fhat ] = jzlk_wienerFilter( g, h, sigma2 )
%Implements Wiener Filtering by simplified formular eq.5.8-6
%  FHat(u,v) = 1/H(u,v) * |H(u,v)|^2 / (|H(u,v)|^2+K) G(u,v)

[M,N] = size(g);
[V,W] = size(h);

% Aliasing could be prevented by padding with zeros. This will lead to some
% sort of dark margin though (0=black), so instead replicate such that the
% blurry margin will be aproximately the same colour as the actual edge
% of the image
g = padarray(g, size(h), 'replicate');

% Taper edges to remove disturbing lines. Refer to p. 275 or chapter
% 5.11.5, course book 3rd edition for description of the problem. Instead
% of windowing which would also effect pixels in the center here edge
% tapering is used. Refer to
% https://se.mathworks.com/help/images/avoiding-ringing-in-deblurred-
% images.html?searchHighlight=edgetaper
window = hamming(32);
PSF = window * window';
PSF = PSF ./ sum(PSF(:));
gTapered =  edgetaper(double(g),PSF);

% Repeat the procedure to smooth edges
for ii=1:15
    gTapered =  edgetaper(double(gTapered),PSF);
end


% Make Fourier transforms of same size
tSize = size(gTapered);
G = fft2(gTapered, tSize(1), tSize(2));
H = fft2(h, tSize(1), tSize(2));

% The exact value for K is given by K = sigma2/var(f).
K =  4*sigma2 / var(double(g(:)));


% Calculate Wiener Transfer Function
H2 = abs(H).^2;
TransferFcn = H2./(H.*(H2+K)); % eq 5.8-6

% Get estimated original image from disturbed image
Fhat = TransferFcn .* G;

% Take inverse Fourier transform to go back to spatial domain
fhat = uint8(ifft2(Fhat));

% Extract only relevant pixels
off1 = ceil((V+1)/2);
off2 = ceil((W+1)/2);
fhat = fhat(off1:M+off1-1, off2:N+off2-1);
```

```
end
```

jzlk_wienerFilterWithoutTapering.m

```
function [ fhat ] = jzlk_wienerFilterWithoutTapering( g, h, sigma2 )
%Implements Wiener Filtering by simplified formular eq.5.8-6 without
%tapering the edges
%  FHat(u,v) = 1/H(u,v) * |H(u,v)|^2 / (|H(u,v)|^2+K) G(u,v)

[M,N] = size(g);
[V,W] = size(h);

% Prevent aliasing due to circular convolution
g = padarray(g, size(h));

% Make Fourier transforms of same size
G = fft2(g, size(g,1), size(g,2));
H = fft2(h, size(g,1), size(g,2));

% The exact value for K is given by K = sigma2/var(f).
K = 4* sigma2 / var(g(:));

% Calculate Wiener Transfer Function
H2 = abs(H).^2;
TransferFcn = H2./(H.*(H2+K)); % eq 5.8-6

% Get estimated original image from disturbed image
Fhat = TransferFcn .* G;

% Take inverse Fourier transform to go back to spatial domain
fhat = ifft2(Fhat);

% Extract only relevant pixels
off1 = ceil((V+1)/2);
off2 = ceil((W+1)/2);
fhat = uint8(fhat(off1:M+off1, off2:N+off2));

end
```

References

[1] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2nd ed., 2002

[2] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 3rd ed., 2009

[3] Diebold, Francis X., Todd A. Gunther, and Anthony S. Tay, *Evaluating density forecasts*, National Bureau of Economic Research Cambridge, Mass., USA, 1997