Computer Vision and Image Processing

Philipp Fröhlich
N1301527G

November 10, 2013

The report has the following form. The source code is written in black and `typewriter` font. The `MATLAB` Output is below and written in the same font, but the color is grey. Annotations and comments are written in this font.

Contents

1 Contrast Stretching

All images are stored in a separate folder called 'images'. This leads to a tidy workspace. But we have to add the name of the folder to the image name.

```
Pc = imread('images/mrttrainbland.jpg');
```

The command `whos` gives a short summary of the matrix properties.

```
whos Pc
```

```
  Name        Size                Bytes  Class    Attributes

  Pc        320x443x3            425280  uint8
```

We see, that the matrix has RGB values. We see this because the third dimension is 3. To convert it to a matrix with grey-scale values, we use the method `rgb2gray`

```
P =rgb2gray(Pc);
whos P
```

```
  Name        Size              Bytes  Class    Attributes

  P         320x443            141760  uint8
```

The command `imshow` shows the picture in a new figure.

```
imshow(P)
```

We find the minimum $\min_P$ and maximum $\max_P$ intensities with the following:

```
min_P = min(P(:)),max_P = max(P(:))
```

```
min_P =

   13


max_P =

  204
```

We want a picture P_1, such that $\min_{P_1} = 0$ and $\max_{P_1} = 255$. To reach this, we subtract at first $\min_P$ from every value and multiply the result with $\frac{\max_{P_1}}{\max_P - \min_P}$

```
P1 =  imsubtract(P,13);
P1 = immultiply(P1, (255/(204-13)));
```

Note that this works only if $\max_P \neq \min_P$. We see, that the minimum and maximum intensity values are correct and that the picture was enhanced.

```
min_P1 = min(P1(:)), max_P1 = max(P1(:))
imshow(P1)
```

```
min_P1 =

    0


max_P1 =

  255
```

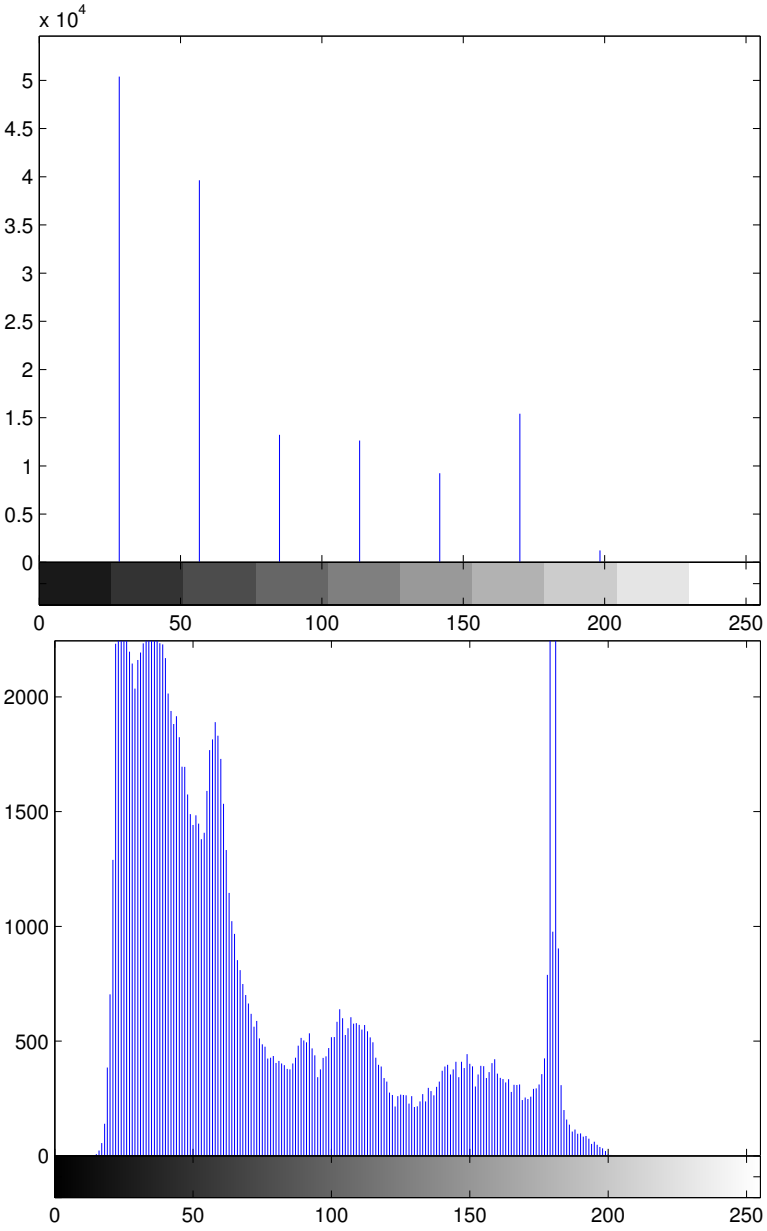
Using the method `imshow(P,[])` does the contrast stretching automatically. So we should see now the same picture as above.

```
imshow(P,[])
```

2 Histogram Equalization

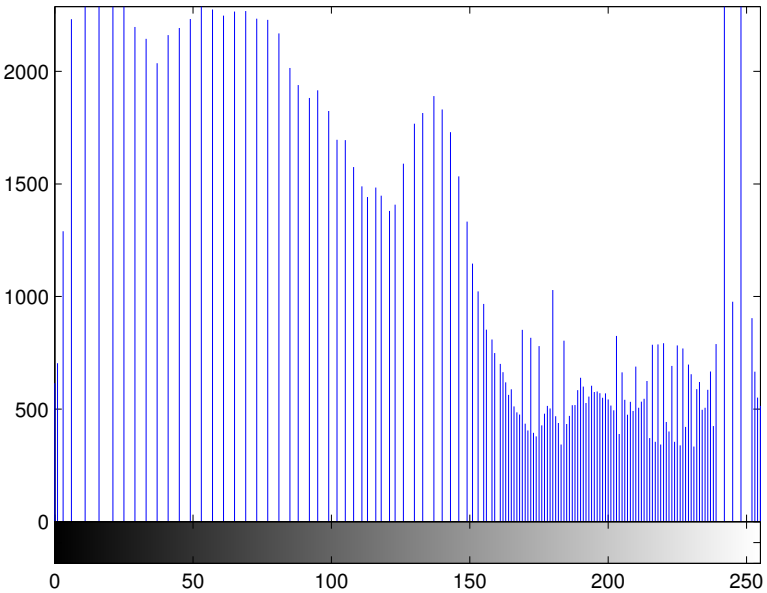
We display the image intensity histogram with 10 and 256 bins using the following commands.

```
%Load P:
P = rgb2gray(imread('images/mrttrainbland.jpg'));
imhist(P,10);
figure
imhist(P,256);
```



We see that the histogram with 256 bins has a higher resolution. But the first gives also a good summary. In both figures, we see that there are more dark than bright pixels. We carry out the histogram equalization, with the following command.

```
P2 = histeq(P,255);
imshow(P2);
figure;
imhist(P2,256);
```

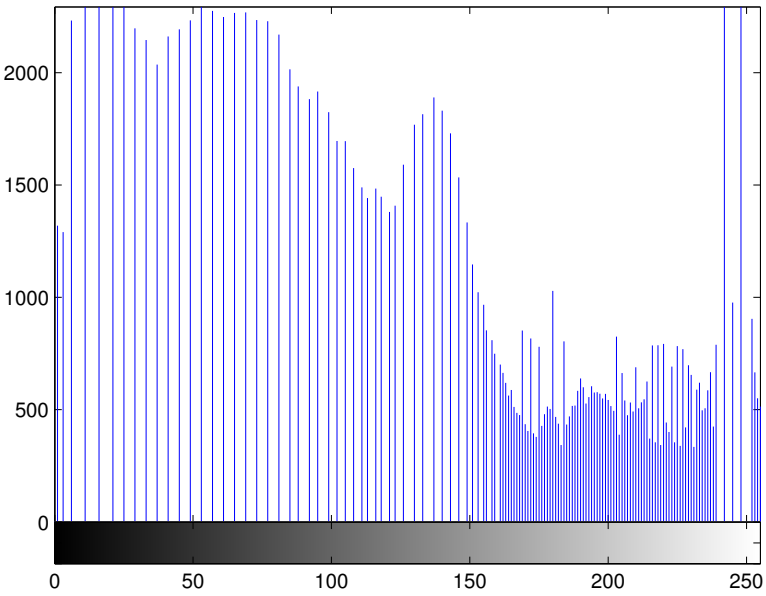


We see in the histogram and in the picture itself, that the number of bright pixels increased. So we obtain a brighter picture. If we run the equalisation again, nothing happens. To see that we subtract the two pictures P_2 and P_3 and test if the result equals 0. If that is the case, the corresponding values of the pictures are equal and the value

in the result `test` is 1. The matrix `test` is a logical matrix, that means that it only contains 1 or 0. So it the minimum of `test` is 1, we know that every value must be 1. Therefore the pictures are equal. of P

```
P3 = histeq(P2,255);
imshow(P3);
figure;
imhist(P3,256);
test = (imsubtract(P2,P3) == 0);
min(test(:))
```

```
ans =

     1
```


2000
1500
1000
500
0
0
50
100
150
200
250

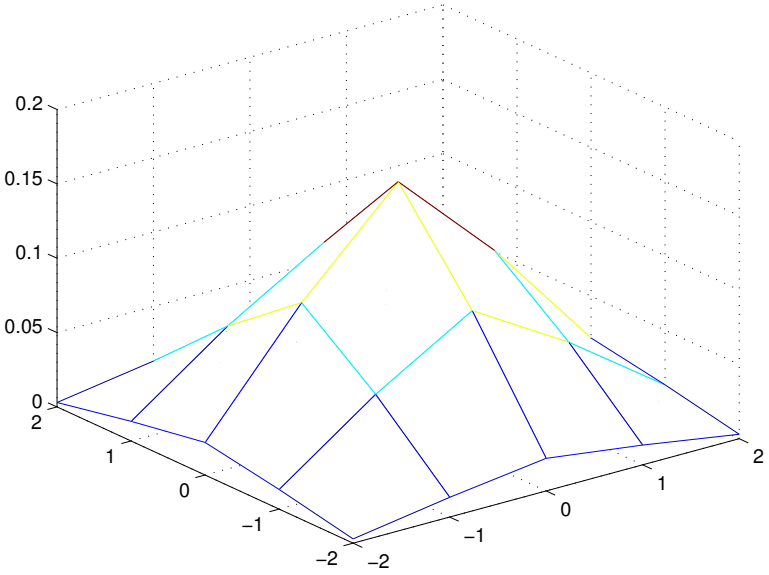
3 Linear Spatial Filtering

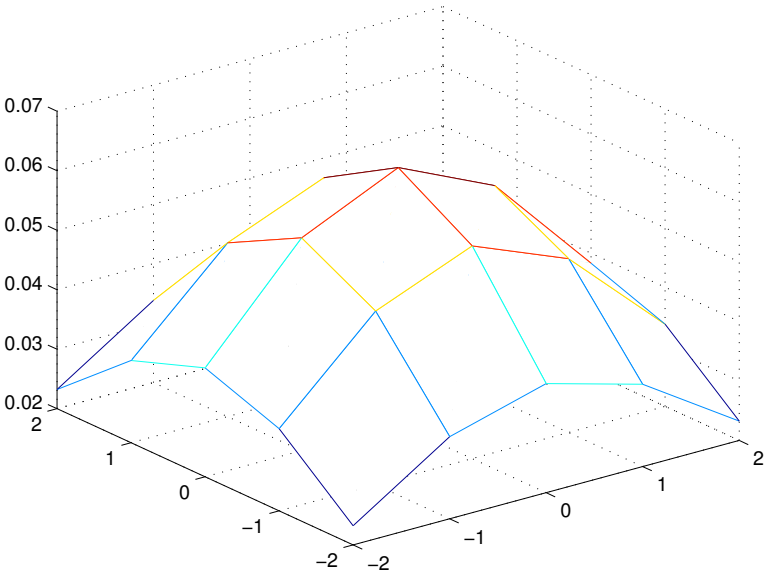
We use the function `getH(sigma)` to compute the matrices. The source code after this part. for $\sigma_1 = 1$ and $\sigma_2 = 2$.

```
H1 = getH(1)
H2 = getH(2)
%Normalisation
%Calculate sums
s1 = sum(H1(:))
s2 = sum(H2(:))
%Multiply with the inverse
H1 = 1/s1 * H1;
H2 = 1/s2 * H2;
%Control
s1 = sum(H1(:))
s2 = sum(H2(:))
% Visualisation with mesh
x = -2:2;
[X,Y] = meshgrid(x,x); %create grid
mesh(X,Y,H1);
figure;
mesh(X,Y,H2);
```

```
H1 =

    0.0029    0.0131    0.0215    0.0131    0.0029
    0.0131    0.0585    0.0965    0.0585    0.0131
    0.0215    0.0965    0.1592    0.0965    0.0215
    0.0131    0.0585    0.0965    0.0585    0.0131
    0.0029    0.0131    0.0215    0.0131    0.0029


H2 =

    0.0146    0.0213    0.0241    0.0213    0.0146
    0.0213    0.0310    0.0351    0.0310    0.0213
    0.0241    0.0351    0.0398    0.0351    0.0241
    0.0213    0.0310    0.0351    0.0310    0.0213
    0.0146    0.0213    0.0241    0.0213    0.0146


s1 =
```

```
    0.9818

s2 =

    0.6297

s1 =

     1

s2 =

    1.0000
```





We see, that both matrices are invariant under 90° rotations, reflection and transposing. In H_1, the center has a higher value compared to the other points.

Now we import the Picture ntugn.jpg from our image folder and view it. To get a better result, we also do a contrast stretching.

```
P = imread('images/ntugn.jpg');
imshow(P,[]);
```

We see that there is a lot of noise. For example in the sky should adjacent neighbours have a similar value. This is not the case here. So we apply our Gaussian filters with zero padding.

```
P1 = conv2(double(P),double(H1),'same');
```

```
P2 = conv2(double(P),double(H2),'same');
% Get back to uint8
P1 = uint8(P1);
P2 = uint8(P2);
% Show results, also with contrast stretching
imshow(P1,[]);
figure;
imshow(P2,[]);
```

We obtained two pictures with less noise. If σ is higher more noise is cancelled, but the image is also more blurred. Now we apply the same filters to the picture ntu-sp.

```
Q = imread('images/ntusp.jpg');
imshow(Q,[]);
```

We see, that the noise is different. In this setting the noise is very bright. Now we apply our filters

```
Q1 = conv2(double(Q),double(H1),'same');
Q2 = conv2(double(Q),double(H2),'same');
% Get back to uint8
Q1 = uint8(Q1);
Q2 = uint8(Q2);
% Show results, also with contrast stretching
imshow(Q1,[]);
figure;
imshow(Q2,[]);
```

In this case, Gaussian filtering is not the best choice. In the first picture the noise value was close to the adjacent pixel values. In this case Gaussian filtering can not work so well, as the noise value is generally far away from the values of the adjacent pixels. So we see, that the brightness of the noise pixels decrease, but the spots get larger, as the noise also affects the adjacent pixels more. In this case, median filtering might be better.

Source Code for `getH`

```
function [ H ] = getH( sigma )
H = zeros(5);
for x = -2:2
    for y = -2:2
        H(x+3,y+3) = exp(-(x^2 + y^2) / (2 * sigma^2) ) / (2 * pi * sigma^2);
    end
end
end
```

4 Median Filtering

As we don't have to compute a additional matrix, the first step is to load the picture again. To optimize the result, we use a contrast stretching again.

```
P = imread('images/ntugn.jpg');
imshow(P,[]);
```

Now we apply the already implemented median filter with the `medfilt2` method.

```
P3 = medfilt2(P); % as 3x3 is default
P5 = medfilt2(P, [5 5]);
imshow(P3,[]);
figure;
imshow(P5,[]);
```

We seem that the result get worse, if neighbourhood size increases. In contrast to that, we see that in the ntusp picture the noise gets cancelled completely.

```
Q = imread('images/ntusp.jpg');
imshow(Q,[]);
figure;
% Apply filters
Q3 = medfilt2(Q); % as 3x3 is default
Q5 = medfilt2(Q, [5 5]);
imshow(Q3,[]);
figure;
imshow(Q5,[]);
```

If the neighbourhood is too big, we lose again some details. So in this case, the 3×3 filter has the best result. This is because the noise points are always single pixels.

5 Suppressing Noise Interference Patterns

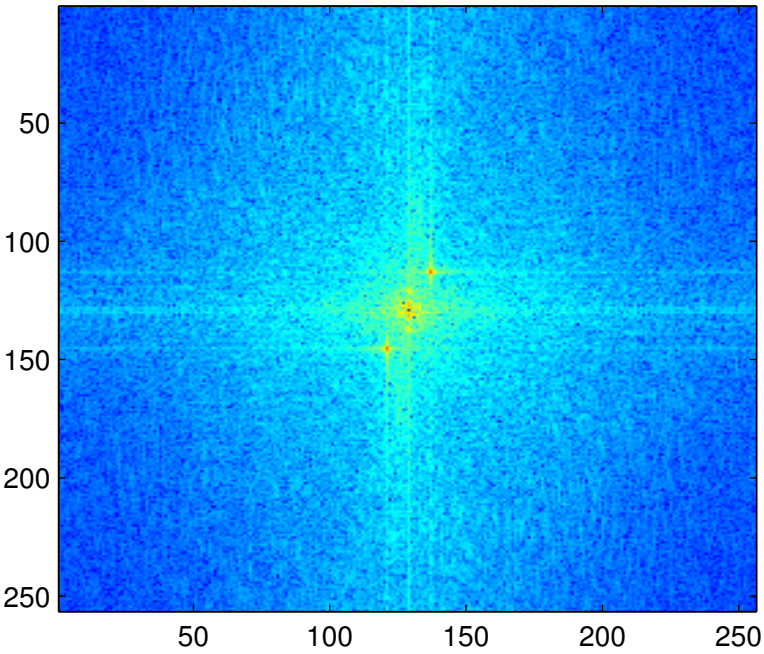
In this exercise we use Fourier transformations to get rid of more complex noise patterns. In this case, diagonal waves.

```
P = imread('images/pckint.jpg');
imshow(P);
```

Now we apply a Fourier Transformation, to see the picture in the frequency domain. This transformation is very popular, so it is already implemented in `MATLAB`.

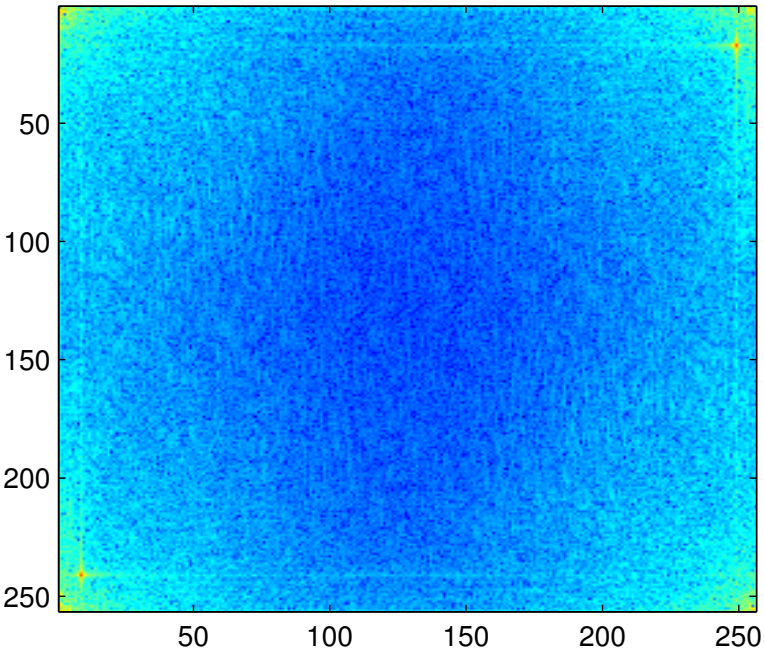
```
F = fft2(P);
% F has complex values
whos F
S = abs(F);
% S has now real positive values
whos S
% Display shifted and transformed image
imagesc(fftshift(S.^0.1));
colormap('default')
```

```
  Name        Size                Bytes  Class     Attributes

  F         256x256           1048576  double    complex

  Name        Size                Bytes  Class     Attributes

  S         256x256            524288  double
```



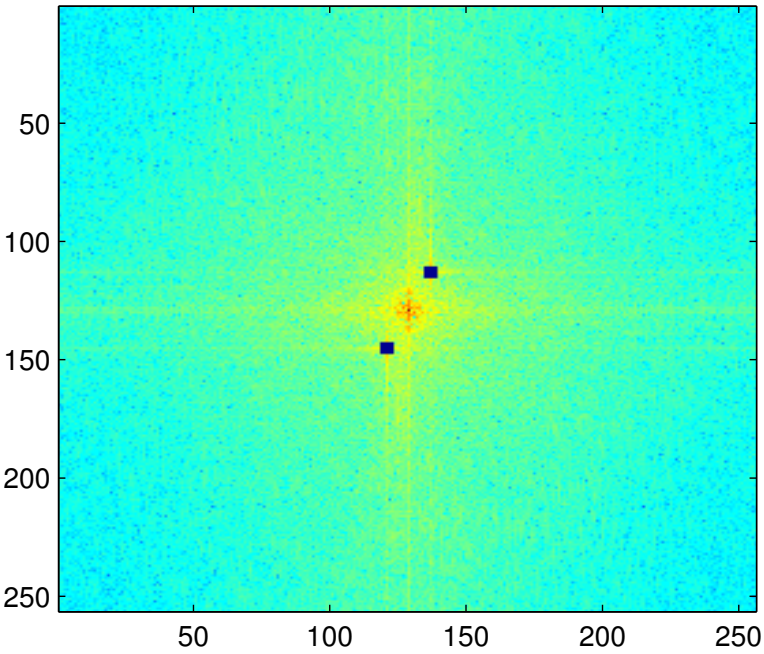
Now we display the power spectrum without `fftshift` method. Then the peaks are at the edges of the picture.help

```
imagesc(S.^0.1);
colormap('default')
% coord  = [up right; down left corner];
coord =    [ 17, 249;
            241, 9];
```



Now we set the area around the specified peaks to zero. This is done "manually" to avoid a lot of error handling code.

```
F1 = F;
a = 2;
F1(17-a:17+a,249-a:249+a) = zeros(2*a + 1);
F1(241-a:241+a, 9-a:9+a) = zeros(2*a + 1);
S1 = abs(F1);
% Display shifted image
imagesc(fftshift(S1.^0.1));
colormap('default')
```



Now we transform F_1 back. To obtain the best result, we do here contrast stretching again.

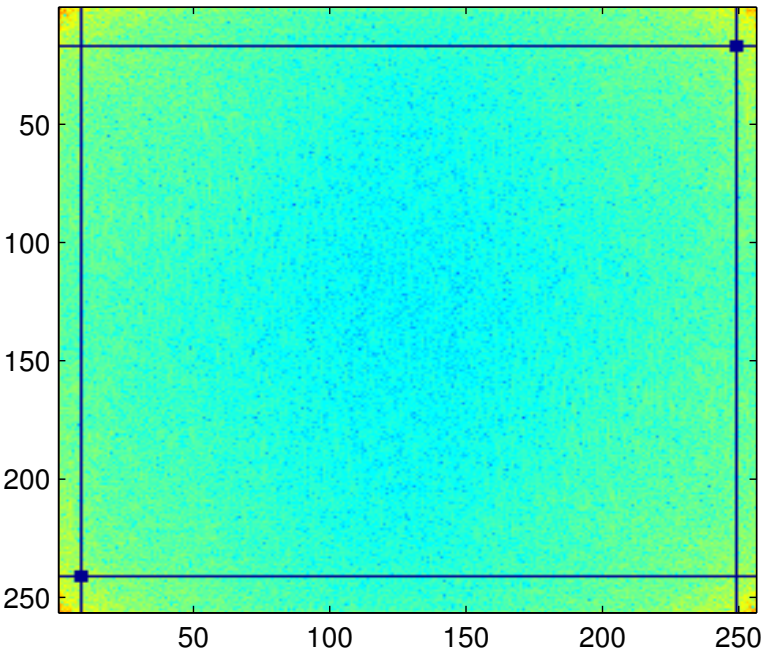
```
P1 = ifft2(F1);
imshow(P,[])
title('Before')
figure;
imshow(P1,[])
title('After')
```

Before

After

This result is better, but if we look at the not shifted power spectrum, we see, that the rows and columns in which the peaks are, are bigger, than their neighbours. So in an additional step we set them also to zero.

```
F2 = F1;
% Get dimensions of the picture
[h w] = size(F1);
% sete rows and columns to zero
F2(:,9) = zeros(h,1);
F2(:,249) = zeros(h,1);
F2(17,:) = zeros(1,w);
F2(241,:) = zeros(1,w);
%Print power spectrum
imagesc(abs(F2).^.1);
colormap('default')
%Print Pictures
figure;
subplot(1,2,1)
imshow(P,[])
title('Before')
subplot(1,2,2)
imshow(P1,[])
title('After')
figure;
imshow(ifft2(F2),[])
title('After additional step')
```



Before

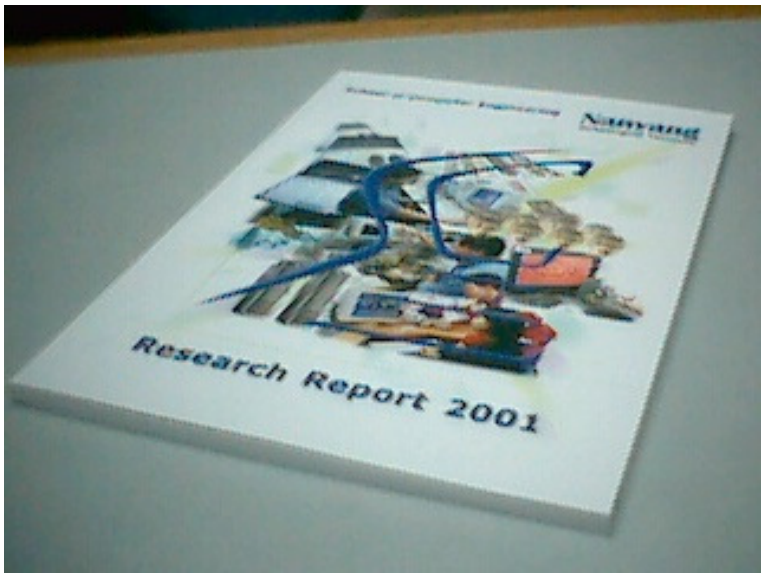
After

After additional step

6 Undoing Perspective Distortion of Planar Surface

First we download the picture and display it.

```
close all;
clear;
P = imread('images/book.jpg');
imshow(P);
```



We use now `ginput` to find the coordinates of the 4 corners. We start in the upper left corner and continue clockwise. We also try to find the 4 corners of the book's title page. A DIN A4 paper has the dimensions $210mm \times 297$ mm. We fix the upper left corner to $(0, 0)$.

```
%Applying ginput
[X_input,Y_input] = ginput(4);
% But we use the coordinates below, obtained from ginput, to get always the best
% result
points = [143,29;
          308,47;
          256,215;
          4,159;
          ];
% The desired coordinates are:
des =  [0,0;
        210,0;
        210,297;
        0,297;
```

```
        ];
% Here Y is the vertical and X the horizontal value.
X = points(:,1);
Y = points(:,2);
% Create the first 6 columns of A
A = [X,Y, ones(4,1), zeros(4,3); zeros(4,3), X,Y, ones(4,1)];
A_end = zeros(8,2);
%Compute the two last  columns
for i =1:4
    A_end(i,: ) = des(i,1) * points(i,:);
    A_end(i+4,:) = des(i,2) * points(i,:);
end
%Put everything together
A = [A,-A_end];
v = [des(:,1);des(:,2)];
```

Note that, the rows of A and v are permuted, compared to equation (*). We first consider the rows containing x_{im} and then the rows containing y_{im}. This trick makes it easier to create the matrix in MATLAB. In linear algebra is shown, that this operation does not affect u at all. The differences are shown below:

```
A
v
%Order rows
order = [1,5,2,6,3,7,4,8];
A_star = A(order,:)
v_star = v(order)
```

```
A =

  Columns 1 through 6

       143          29           1           0           0           0
       308          47           1           0           0           0
       256         215           1           0           0           0
         4         159           1           0           0           0
         0           0           0         143          29           1
         0           0           0         308          47           1
         0           0           0         256         215           1
         0           0           0           4         159           1

  Columns 7 through 8
```

```
           0           0
      -64680       -9870
      -53760      -45150
           0           0
           0           0
           0           0
      -76032      -63855
       -1188      -47223


v =

     0
   210
   210
     0
     0
     0
   297
   297


A_star =

  Columns 1 through 6

         143          29           1           0           0           0
           0           0           0         143          29           1
         308          47           1           0           0           0
           0           0           0         308          47           1
         256         215           1           0           0           0
           0           0           0         256         215           1
           4         159           1           0           0           0
           0           0           0           4         159           1

  Columns 7 through 8

           0           0
           0           0
      -64680       -9870
           0           0
      -53760      -45150
      -76032      -63855
           0           0
```

```
      -1188      -47223


v_star =

     0
     0
   210
     0
   210
   297
     0
   297
```

Now we we want to compute the variable $u = A^{-1}v$. To do that, we use the \ operator

```
u = A\v
% Bringing it back into the original 3 times 3 matrix
U = reshape([u;1], 3, 3)';
%Testing
w = U*[X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
%Is w and des close enough?
if sum(abs(w(1:2,:) -des')) < 1e-12
    T = maketform('projective', U');
    P1 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
    figure;
    imshow(P1);
else
    error('Projection did not work, use other points')
end

u =

    1.5198
    1.6250
 -264.4529
   -0.4211
    3.8598
  -51.7218
    0.0002
    0.0056
```

The new image is a little bit blurred, especially in the upper half of the picture. In the original picture, we see, that this is the part, which is farer away from the camera. A pixel in the back of the original picture represents a bigger area than a point in the foreground. Therefore we have less information about the background and so the upper half of the new picture must have worse quality.

7 Edge Detection

In a first step, we load the image,convert it into a a grey level image and display it.

```
close all
clear all
P = imread('images/maccropped.jpg');
P = rgb2gray(P);
```

```
imshow(P);
```

We want to convolute the Sobel mask with the image, therefore the vertical matrix must have the following form:

$$V = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

The horizontal Sobel filter is:

$$H = V^T$$

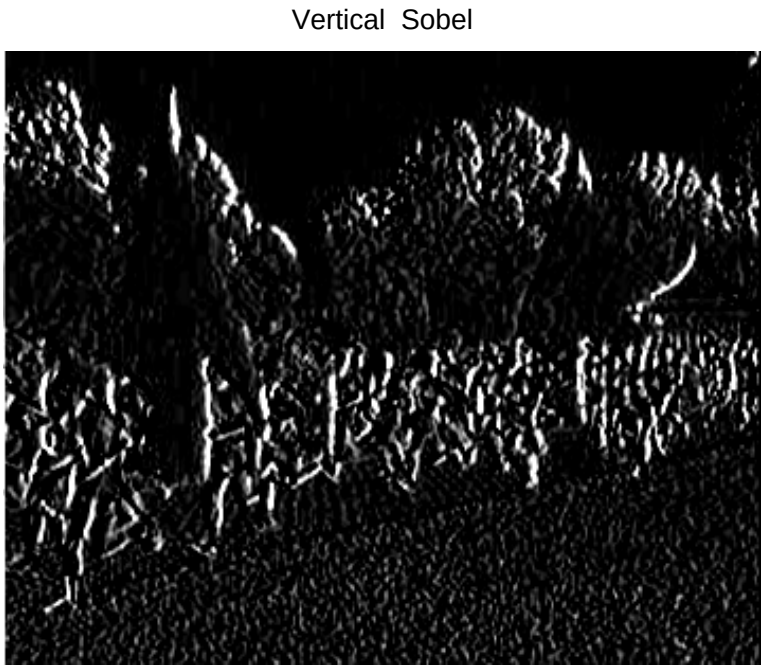
If a edge is not vertical nor horizontal, we can not detect it. We see this, if we at the edge which separates street and grass. To detect edges with an angle of 45°, the following matrices are suitable.

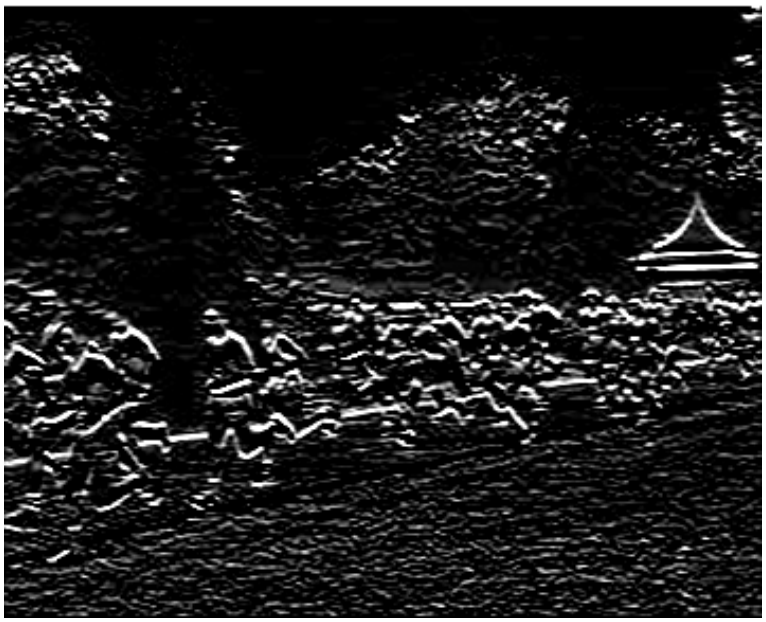
$$D_- = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix} \text{ and } D_+ = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

The results of both filters are displayed below.

```
vert = [1 0 -1 ;
        2 0 -2 ;
        1 0 -1];
hori = vert';
%Apply convolution
Pv = conv2(double(P),double(vert),'same');
Ph = conv2(double(P),double(hori),'same');
imshow(uint8(Pv))
```

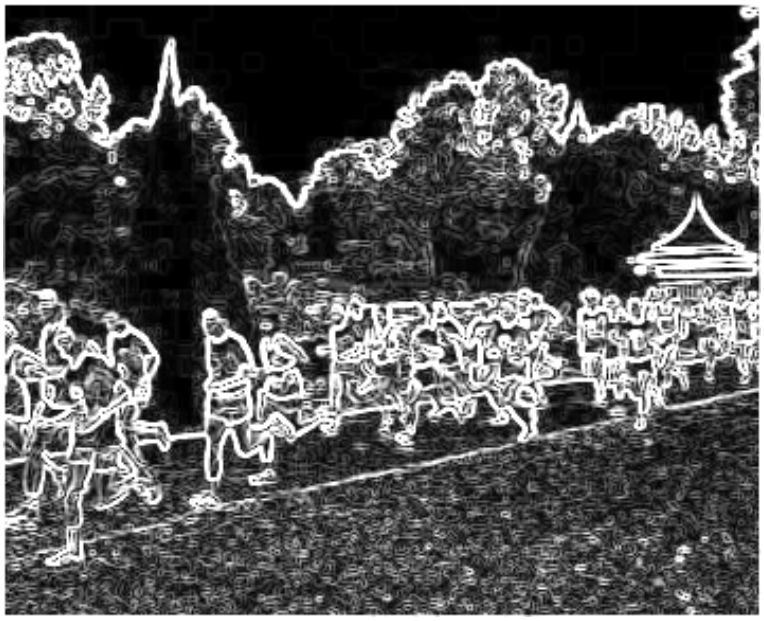
```
title('Vertical  Sobel')
figure;
imshow(uint8(Ph))
title('Horizontal Sobel')
```





Now we add and square the results `Pv` and `Ph`. In this case there are two main reasons to square the sum. Firstly squaring eliminates negative values. Secondly the difference between the values are amplified.

```
Ps = imadd(immultiply(Pv, Pv),immultiply(Ph, Ph));
Ps = sqrt(Ps);
%show result
close 1
imshow(uint8(Ps));
```



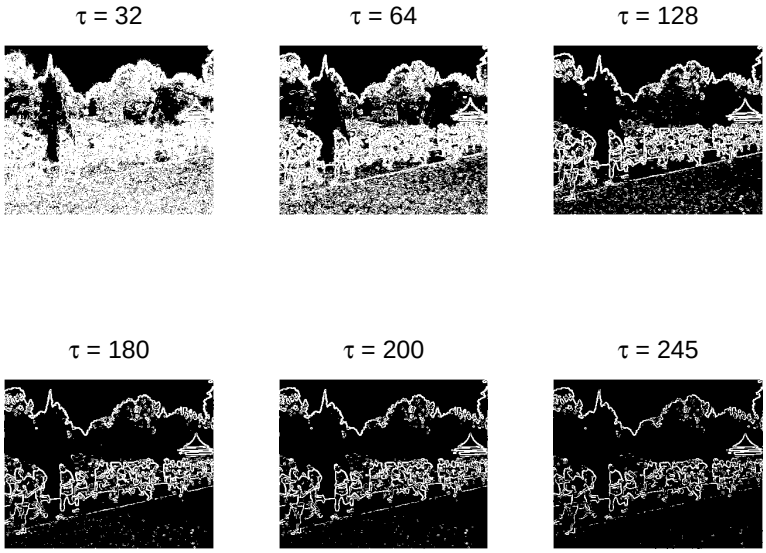
Now we apply different thresholds. I chose the following thresholds:

$$\tau \in \{32, 64, 128, 180, 200, 245\}$$

If the threshold is chosen too low, we have to much information. If the threshold is chosen too high, important information and details are lost.

```
E32 = Ps > 32;
E64 = Ps > 64;
E128 = Ps > 128;
E180 = Ps > 180;
E200 = Ps > 200;
E245 = Ps > 245;
subplot(2,3,1)
imshow(E32);
title('\tau = 32')
subplot(2,3,2)
imshow(E64);
title('\tau = 64')
subplot(2,3,3)
imshow(E128);
```

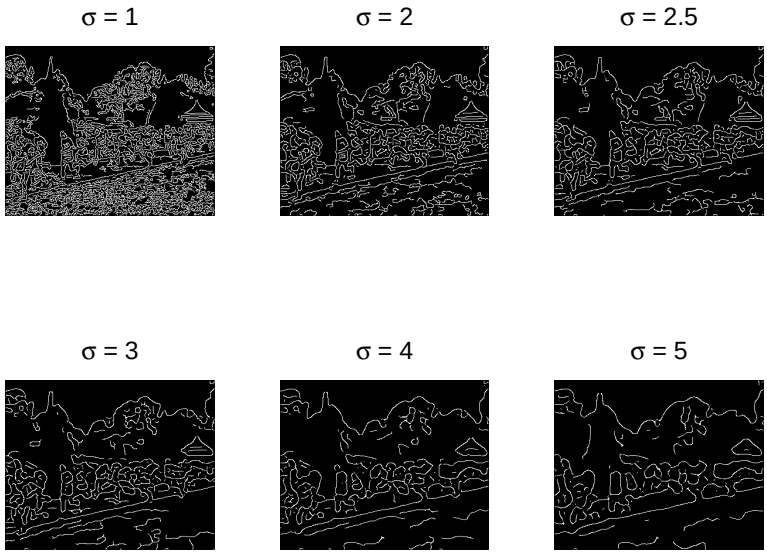
```
title('\tau = 128')
subplot(2,3,4)
imshow(E180);
title('\tau = 180')
subplot(2,3,5)
imshow(E200);
title('\tau = 200')
subplot(2,3,6)
imshow(E245);
title('\tau = 245')
```



The parameter σ is the standard deviation of the Gaussian filter. As we have learned in class, this parameter decides how much influence the adjacent pixels have. In one of the previous experiments was shown, that a higher σ value leads to more noise cancellation, but small details can disappear and the images becomes more blurred. In this case the results are similar. If σ is small we see a lot of edges and also noise, e.g. in the grass. If σ is higher, we might have lost some edges, but more noise is reduced.

```
subplot(2,3,1)
E1 = edge(P,'canny', [.04 .1], 1);
imshow(E1);
title('\sigma = 1');
subplot(2,3,2)
E2 = edge(P,'canny', [.04 .1], 2);
imshow(E2);
title('\sigma = 2');
subplot(2,3,3)
E2_5 = edge(P, 'canny', [.04 .1] , 2.5);
imshow(E2_5);
```

```
title('\sigma = 2.5');
subplot(2,3,4)
E3 = edge(P,'canny', [.04 .1], 3);
imshow(E3);
title('\sigma = 3');
subplot(2,3,5)
E4 = edge(P,'canny', [.04 .1], 4);
imshow(E4);
title('\sigma = 4');
subplot(2,3,6)
E5 = edge(P,'canny', [.04 .1], 5);
imshow(E5);
title('\sigma = 5');
```



The variable `tl` is the *lower threshold*. The lower this threshold is, the more details can be seen in the result. So the minimum intensity change leading to a edge detection is controlled by `tl`. Therefore every white point in a picture with lower threshold `tl` is also white in every picture with a smaller lower threshold. This is shown in by calculating the differences and finding their minimum and maximum. There we also see see, that negative values for `lt` produce the same output as $\texttt{lt} = 0$.

```
subplot(2,3,1)
E_tl1 = edge(P,'canny', [ -.5 .1], 2);
imshow(E_tl1);
title('tl = -0.5');
subplot(2,3,2)
E_tl2 = edge(P,'canny', [.0 .1], 2);
imshow(E_tl2);
title('tl = 0');
```

```
subplot(2,3,3)
E_tl3 = edge(P,'canny', [.03 .1], 2);
imshow(E_tl3);
title('tl = 0.03');
subplot(2,3,4)
E_tl4 = edge(P,'canny', [.05 .1], 2);
imshow(E_tl4);
title('tl = 0.05');
subplot(2,3,5)
E_tl5 = edge(P,'canny', [.07 .1], 2);
imshow(E_tl5);
title('tl = 0.07');
subplot(2,3,6)
E_tl6 = edge(P, 'canny', [.0999 .1] , 2);
imshow(E_tl6);
title('tl = 0.0999');

%Calculate differences
diff1 = E_tl1 - E_tl2;
diff2 = E_tl2 - E_tl3;
diff3 = E_tl3 - E_tl4;
diff4 = E_tl4 - E_tl5;
diff5 = E_tl5 - E_tl6;

%max(diffx(:)) = 1 means white point in x but not in x+1
%min(diffx(:)) = 0 means no black point in x, which is white in x+1,
%otherwise the minimum would be -1

max1 = max(diff1(:));min1 = min(diff1(:));
max2 = max(diff2(:));min2 = min(diff2(:));
max3 = max(diff3(:));min3 = min(diff3(:));
max4 = max(diff4(:));min4 = min(diff4(:));
max5 = max(diff5(:));min5 = min(diff5(:));

if E_tl1 == E_tl2
    disp('Negative lower thresholds produce the same output as lt = 0');
end
if [max2 max3 max4 max5 ] == ones(1,4)
    disp('If lt increases, white points (= noise or edge pixels) disapear');
end
if [min2 min3 min4 min5] == zeros(1,4)
    disp('If lt increases, no black point turns white -> number of detected points de
end
```

```
Negative lower thresholds produce the same output as lt = 0
If lt increases, white points (= noise or edge pixels) disapear
If lt increases, no black point turns white -> number of detected points decreases
```

tl = −0.5

tl = 0

tl = 0.03

tl = 0.05

tl = 0.07

tl = 0.0999