# Image Processing
# lab 2

Klaas Kliffen Jan Kramer

December 4, 2015

## Exercise 1 – Fourier spectrum

**a**. The functions build in functions fft2 and fftshift are used to create the fourier spectrum image centered around the DC component of the fourier transform. Some extra code is added for calculating the average of the image, which will be explained in more detail in part c of this exercise. To be able to view the spectrum, the values need to be scaled. So the log is taken of each value to increase the constrast.

```
% Read the image
x = imread('../images/characters.tif');
% Get image width and heigt
[width, height] = size(x);
% Perform the Fourier transform
spectrum = fftshift(fft2(x));
% Calculate the avarge of the image
% It can be found by taking the dc component (center of the image)
% And dividing it by the number of pixels
avg_fourier = abs(spectrum(width/2+1,height/2+1))/(width*height)
avg_mean = mean(mean(x))

% calculate the magnitude
spectrum = abs(spectrum);

% take the log value for better scaling in octave
logspectrum = log2(spectrum);
% Take note of max and min of the spectrum for image scaling
maxs = max(max(logspectrum));
mins = min(min(logspectrum));
% Scale the image for output to file
spectrumimg = uint8(floor((logspectrum - mins) / (maxs-mins) * 256));
imwrite(spectrumimg, 'spectrum.png');
% Show the log image as a figure
figure, imshow(logspectrum,[]), colormap gray
```

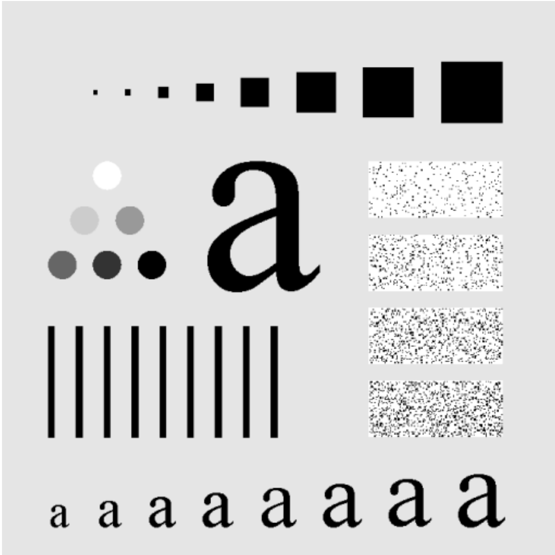


Figure 1: Original image

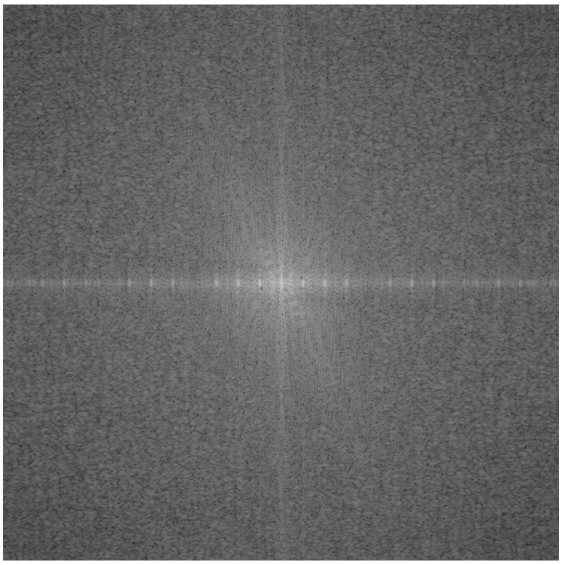


Figure 2: Fourier spectrum of figure 1

**b.**

**c.** The maginitude of the center of the image is the DC component. For images, this is the sum of the grey levels of all pixels. Dividing this by the number of pixels gives the average grey level. In this case: 207.31 for figure 1. This value is equal to the result achieved by using `mean` two times.

## Exercise 2 – Highpass filtering in the frequency domain

**a**. The Gaussian highpass transfer function is given by

$$H(u,v) = 1 - e^{-\frac{D(u,v)^2}{2 \cdot D_0^2}} = 1 - e^{-\frac{(u-P/2)^2+(v-Q/2)^2}{2 \cdot D_0^2}}$$

where $P \geq 2 \cdot M - 1$, $Q \geq 2 \cdot N - 1$, $u = [0, P-1]$ and $v = [0, Q-1]$. This function is used to lessen the strength of low frequencies in a spectrum by multiplying it elementwise with said spectrum. An example can be seen in Figure 3. Note that the white part allow the frequencies to pass through, while the black part weakens low frequencies if the spectrum is centered.

Figure 3: The Gaussian highpass filter with $D_0 = 30$

Another thing to note is that filter size is bigger than the image size. The reason for this is that the image is expected to be padded to prevent wraparound errors as shown in Figure 4.32 in the book.

```
function H = IPgaussian (D0, M, N)
% calculate the padded image dimensions based on the suggested value
% in Section 4.7.3 of the book
P = 2 * M;
Q = 2 * N;
% take u = 0, ..., P-1 and v = 0, ..., Q-1 (book Section 4.8.0)
[V, U] = meshgrid(0:P-1, 0:Q-1);
% calculate the squared distance D(u,v) of Eq. (4.8-2)
squaredist = ((U - P/2).^2 + (V - Q/2).^2);
% calculate H(u,v) based on Eq. (4.9-4)
% note that this is done with matrix operations
Hc = ones(P, Q) - e.^(- squaredist / ( 2 * D0^2));
% finally the filter is uncentered
H = ifftshift(Hc);
endfunction
```

**b**. The general way of applying filters in the frequency domain is summarized in Section 4.7.3 of the book. In the case of IPftfilter the steps are as follows:

**Step 1** Pad the image to the size of the filter to prevent wraparound errors.

**Step 2** Perform DFT on this padded image to get the spectrum.

**Step 3** Apply the given filter to the spectrum.

**Step 4** Perform the inverse DFT on the new spectrum and take the real parts.

**Step 5** Extract the final image by removing the padding.

Note that this can also be done with every spectrum shifted to the center. This does not influence the result, so it was omited.

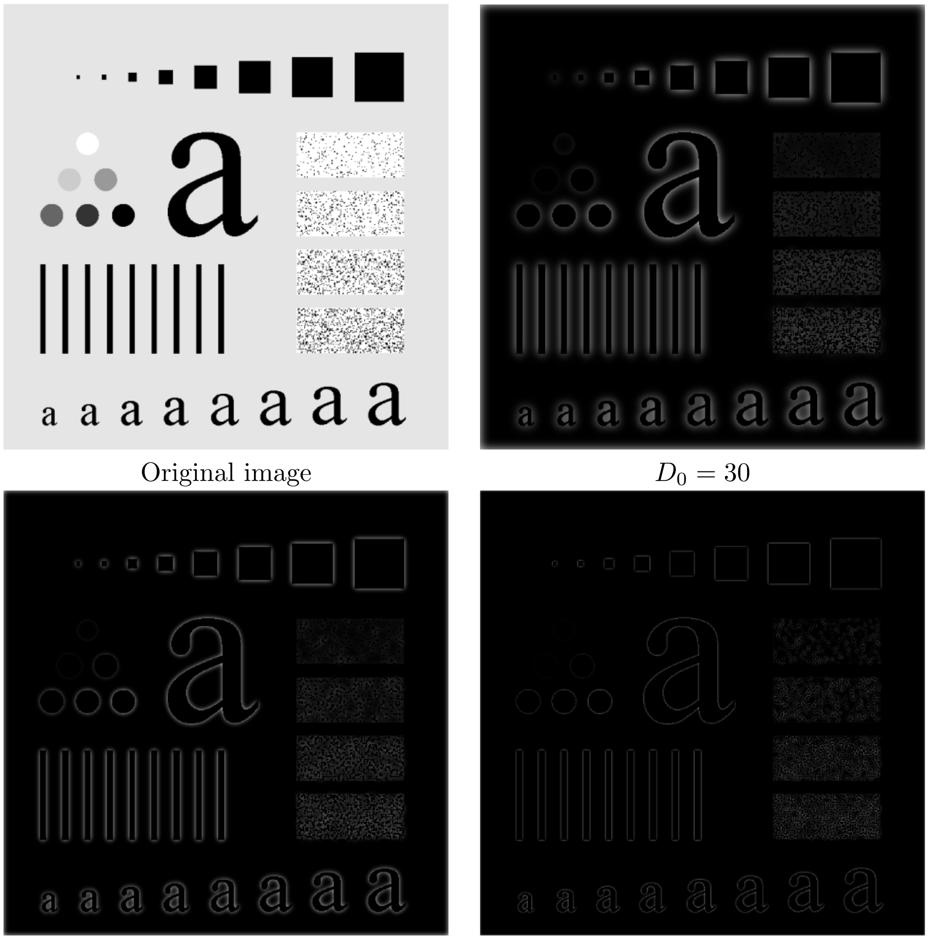
```
function rval = IPftfilter (x, H)

M = size(x, 1);
N = size(x, 2);

% pad the image such that it has the same dimensions
% as the filter transfer function
fp = uint8(zeros(size(H)));
fp(1:M, 1:N) = x;

fp = im2double(fp);
% calculate the spectrum of the padded image
% note that shifting the center is not needed, because the filter H is also
% not centered
Fp = fft2(fp);
% applay the filter to the image
G = H .* Fp;
% convert the new spectrum to the image by taking the inverse DFT, the real
% values and unpadding it
newx = real(ifft2(G))(1:M, 1:N);
rval = im2uint8(newx);

endfunction
```



Original image  $D_0 = 30$

$D_0 = 60$  $D_0 = 160$

Figure 4: Gausian Highpass Filtering

**c.**

## Exercise 3 – Median filtering

**a.** The `IPmedian` function takes the distorted image and a value k as its input parameters. The for each pixel in the image, a window is created with width and height of $2k + 2$. When encountering a boundary, the windowsize is decreased to fit the image. A submatrix is then used to represent all the pixels in the window. From this submatrix, the median is calculated and used for the output image.

```
% Median filter width a 2k+1 x 2k+1 window
function [out] = IPmedian(img,k)
% get the image size
[width, height] = size(img)
%create the output image
dest = uint8(zeros(size(img)));

```

```
% loop over all pixels
for x = 1 : width
  for y = 1 : height
    % determine the edge of the window
    % the size of the window shrinks at the boundaries of the image
    startx = max(x-k,1);
    starty = max(y-k,1);
    endx = min(x+k,width);
    endy = min(y+k,height);
    % get the submatrix
    submat = img(startx:endx,starty:endy);
    [w,h] = size(submat);
    % convert it to a vector to be able to calculate the median
    submat = reshape(submat, 1, w*h);
    % take the median and store it
    dest(x,y) = median(submat);
  end
end
% output the image
out = dest;
```

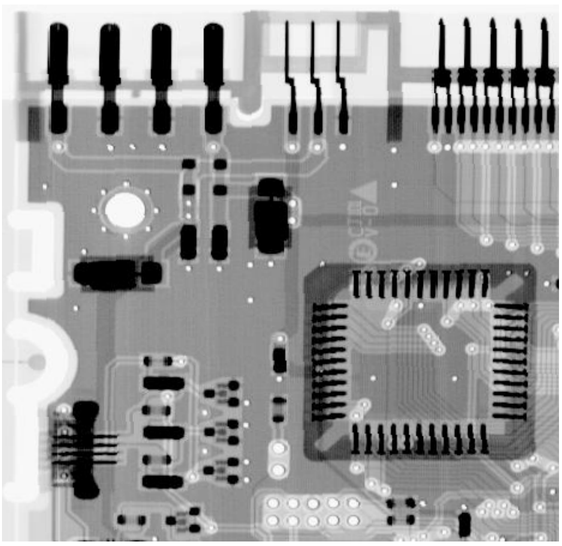


Figure 5: Original image of the circuitboard

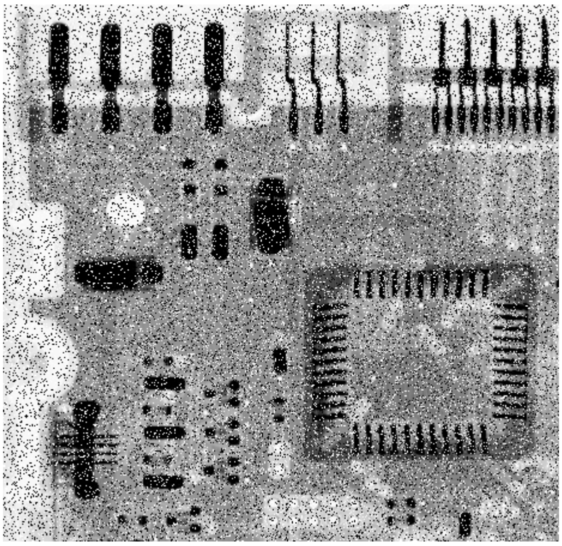
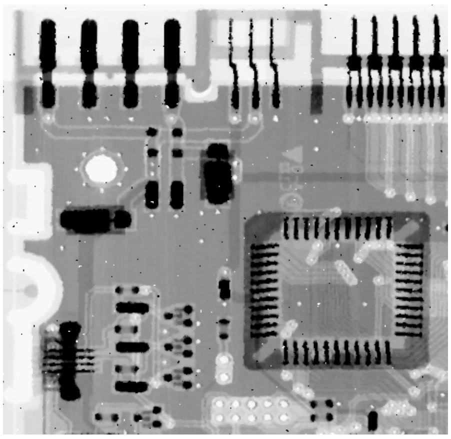
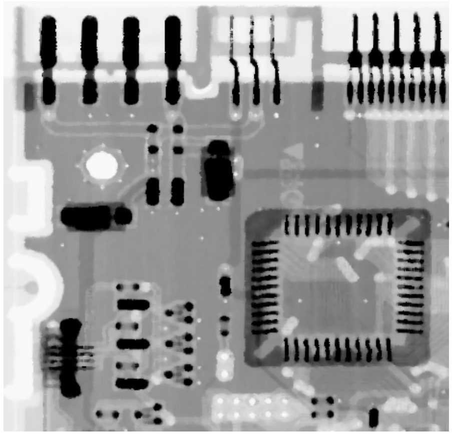


Figure 6: Figure 5 with salt & pepper noise with $Pa = Pb = 0.2$

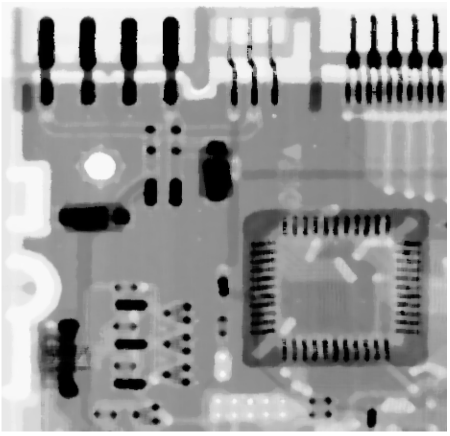
**b**.



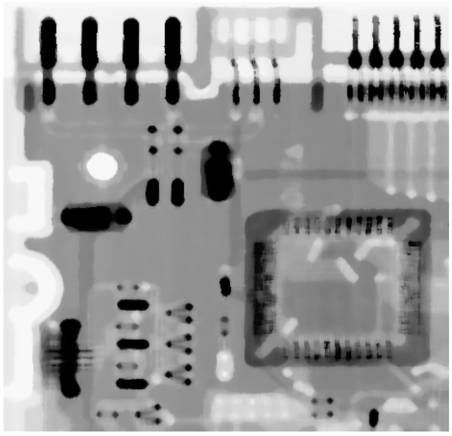
$k = 1$



$k = 2$



$k = 3$



$k = 4$

Figure 7: Using median filtering with different window values $k$ on figure 6

**c.** In the image created by filtering with $k = 1$, some noise is still present. This is equal to the image 5.10b in the book. Using $k = 2$ removes all noise particles. Increasing k further reduces the sharpness of the image as can be seen in the two lower images in 7

## Task distribution

| ex1 | design | implementation | answers questions | writing report |
|---|---|---|---|---|
| Klaas | 50% | 100% | 50% | 50% |
| Jan | 50% | 0% | 50% | 50% |

| ex2 | design | implementation | answers questions | writing report |
|---|---|---|---|---|
| Klaas | 50% | 0% | 50% | 50% |
| Jan | 50% | 100% | 50% | 50% |

| ex3 | design | implementation | answers questions | writing report |
|---|---|---|---|---|
| Klaas | 50% | 100% | 50% | 50% |
| Jan | 50% | 0% | 50% | 50% |