Image Processing
lab 2
Group 1

Alessandro Pianese S4106431
Viljami Linna S4169921

March 25, 2020

Individual contributions

Viljami Linna mainly planned and implemented a code and wrote a report related to the Exercise 1 and 2. The code, and report related to Exercise 3 was mainly planned and implemented by Alessando Pianese. Both members of the group are familiar with solutions of every exercise. Structure and language of the report has been checked by both members of the group.

Exercise 1 – Fourier spectrum

a. The assignment of this part of the exercise was to compute centred Fourier spectrum for the given image. Fourier transform is an operation that allows you to found frequencies of any periodic signal consisting of. Fourier transform gives a sum of frequencies and their phase angles as an output.

In this exercise, the input image (signal) was discrete and two-dimensional so we used discrete 2-d Fourier transform

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi(ux/M+vy/N)} \tag{1}$$

where F is the Fourier transform of f, which is the discrete input signal. M together with N are the dimensions of f.

Every $F(u,v)$ is an imaginary number representing a one term of the frequencies of the corresponding signal so that the absolute value of the $F(u,v)$ is the magnitude, the angle between real axis $F(u,v)$ is the phase angle and u together with v describe the angle speed and direction of that term.

To make Fourier transform centred, we had to shift the result of the Fourier transform so that $F(0,0)$ was represented in the centre of the image. That can be done by replacing every $F(u,v)$ so that

$$F(u,v) = F(u-(M-1)/2, v-(N-1)/2) \tag{2}$$

Finally, the centred Fourier spectrum was obtained by replacing every $F(u, v)$ in the centred Fourier transform by the absolute value of the corresponding term. Listing 1 shows our implementation of this exercise.

```
1 function magnitudes = IPcenteredFourier(filename)
2 file = imread(filename);
3 fourierTransform = fft2(file);
4 magnitudes = abs(fftshift(fourierTransform));
5
6 end
```

Listing 1: Fourier spectrum code

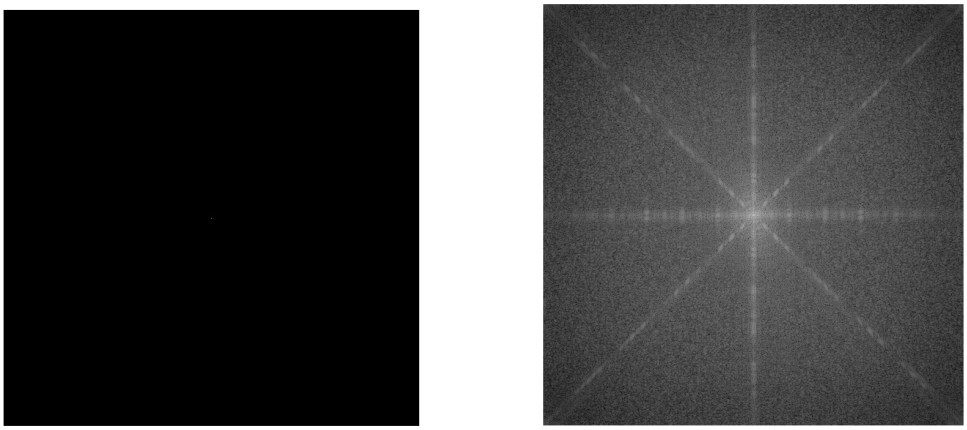
b. In this part of the exercise, we had to display the Fourier spectrum of the given image 'characters.tif.' At first, we used function represented in Listing 1 with addition in row 3, where we add a function which converted the image to a double format before Fourier transform. We let the input parameter to be 'characters.tif.' Next, we scaled magnitude values in a range from 0 to 1 to make spectrum representable.

Different magnitudes were still not nicely separable in the image. That is why we, after testing, improved our solution by scaling magnitudes right after conversion to logarithmic scale so that $M_{log} = ln(1+M)$ where M is the magnitude. Finally, we got improved display by repeating the previously mentioned scaling in a range from 0 to 1 for the logarithmic scale spectrum.

The input image is represented in Figure 2. Our implementation is represented in Listing 2. Figure 1 shows the results of linear and logarithmic scale display of the spectrum.

```
1 filename = 'characters.tif';
2 %calc magnitudes
3 Magnitudes = IPcenteredFourier(filename);
4 %transform to logarithmic scale
5 LogMagnitudes = log(1 + Magnitudes);
6 f1 = figure(1);
7 imshow(rescale(LogMagnitudes,0,1));
8
9 f2 = figure(2);
10 imshow(rescale(Magnitudes,0,1));
```

Listing 2: Spectrum displaying code



(a) Linear scale (b) Logarithmic scale

Figure 1: Fourier spectrum of '`characters.tif`' in linear and logarithmic scale.

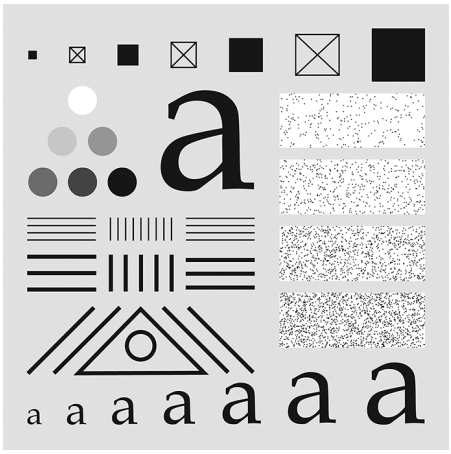


Figure 2: The image '`characters.tif`.'

c. The assignment was to use our result in (a) to compute the average value of the image 'characters.tif' displayed in Figure 2. The average of the image can be calculated from the magnitude of the Fourier spectrum term with angle speed 0.

Because our function in (a) centres the spectrum we first did the inverse shifting. After that, the right term is able to be found at the beginning of the spectrum matrix. To get the average from the constant term we had to divide the magnitude by amount of pixels in image which is same as the number of calculated spectrum frequencies.

The actual code is represented in Listing 3. According to our results, the average of the image was 203.9351.

```
filename = 'characters.tif';
CenteredMagnitudes = IPcenteredFourier(filename);
Magnitudes = ifftshift(CenteredMagnitudes);
Magnitudes(1,1)/(size(file,1)*size(file,2))
```

Listing 3: Image average calculation code.

Exercise 2 – Highpass filtering in the frequency domain

a. The assignment was to implement a function which takes input parameters $D0$(cutoff frequency), n(number of terms), M and N(dimensions of the filter). The function had to construct a Butterworth highpass filter (BHPF) transfer function H according to the given parameters.

The equation of the BHPF filter is

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}} \tag{3}$$

where $D(u,v)$ is the distance from the filter center so that

$$D(u,v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2}. \tag{4}$$

Our implementation is presented in Listing 4

```
function H = IPbhpf(D0,n,M,N)
    H = zeros(M,N);
    for j =1:M
        for i=1:N
            D = sqrt(((j-(M/2))^2)+((i-(N/2))^2));
            H(j,i) = 1/(1 + ((D0/D)^(2*n)));
        end
    end
end
```

Listing 4: IPbhpf

b. In this part of the exercise, we had to write a function IPftfilter which takes an image x and transfer function H as an input parameter and performs convolution filtering of an input image.

Convolution filtering can be performed by the production of the Fourier transform of the image and the transform function of the image as shown in Equation 5

$$\mathcal{F}\{f * h\}(\mu) = F(\mu)H(\mu) \tag{5}$$

We used the above-mentioned feature and perform filtering by computing the centred Fourier transform of the given image x.

```
fourierX = fft2(im2double(x));
centeredFourierX = fftshift(fourierX);
```

Then we performed a production of the centred transform and given transform function H.

```
centeredFourierOutputImage = centeredFourierX.* H;
```

Finally, we made inverse centring and transforming operations to undo the frequency representation of the image.

```
fourierOutputImage = ifftshift(centeredFourierOutputImage);
outputImage = real(ifft2(fourierOutputImage));
```

Entire implementation of the function is displayed in Listing 5 in appendices.

c. The assignment was to perform highpass filtering to 'characters.tif' and duplicate the BHPF results displayed in the right column of the Fig. 4.53 in the coursebook[1]. Both of the figures of the book had got by using Butterworth high pass filter with parameters $n = 2$. In another figure, they used $D0 = 60$ and in another $D0 = 160$. Figure 2 was used as an input parameter in both cases.

In our implementation, at first, we read the image and converted it to double format.

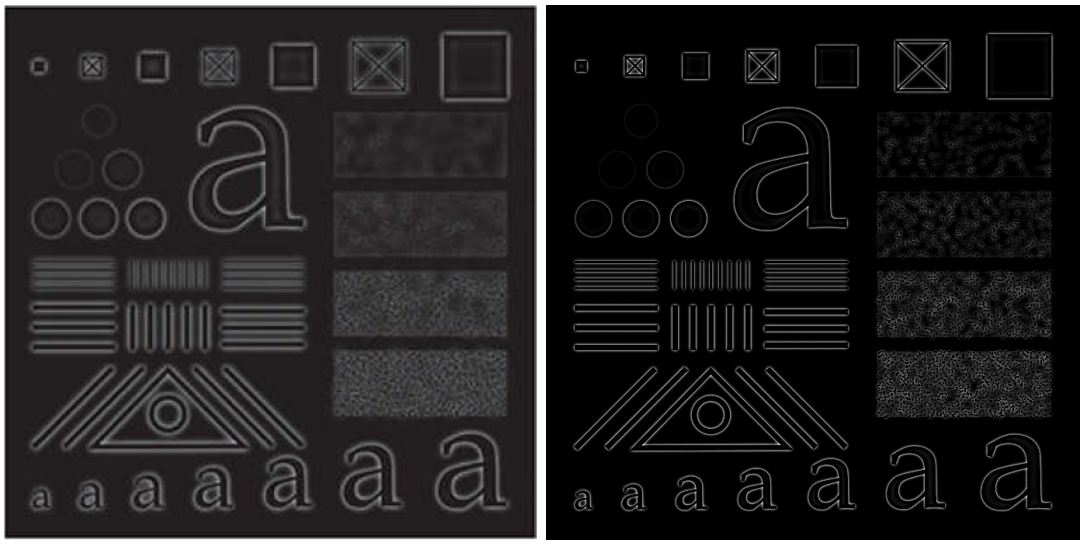
```
file = imread('characters.tif');
file = im2double(file);
```

After that, we used function represented in Listing 4 constructing the transfer function of the high pass filter.

```
H = IPbhpf(60, 2, size(file,1), size(file,2));
filteredImage = IPftfilter(file, H);
```

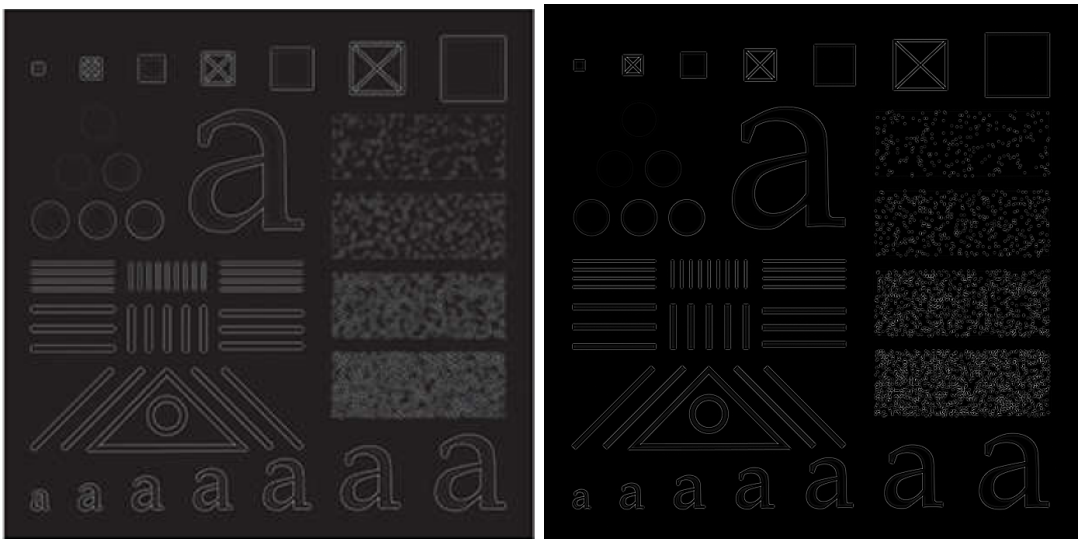
Our implementation is displayed in Listing 6. Results of the book compared with our results are represented in Figure 3 and Figure 4.

Both of the cases, the results differ a bit with contrast but the characteristics are similar. Difference between results may occur because book results were taken as a screenshot from the book which was in PDF format. There may have occurred some compressions when a computer has converted image from a book to different formats. Another possible reason explaining the difference may be image format conversions from uint8 to double in our implementation.



(a) Result in book[1]. (b) Our result

Figure 3: Butterworth highpass filtering results comparison, D0=60.



(a) Result in book[1]. (b) Our result

Figure 4: Butterworth highpass filtering results comparison, D0=160.

Exercise 3 – Median Filtering

a. The function that performs the median filtering is displayed in listing 7. The function `IPmedian` rebuilds an approximation $\hat{f}$ from a noisy image g such that

$$\hat{f}(x,y) = Median[g(s,t)]$$

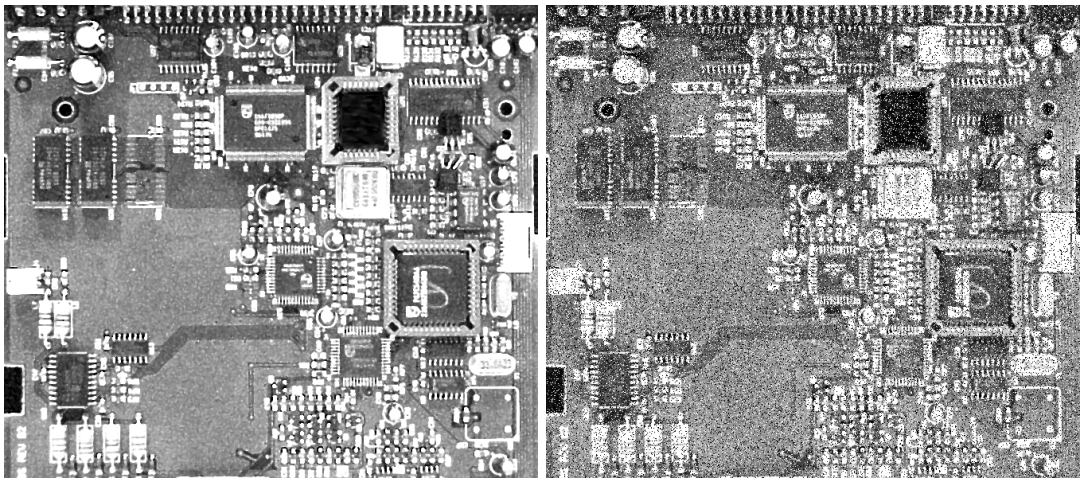
where $g(s,t)$ is a sub-image of the noisy image. The implementation simply select, for each pixel of the output image, the median of the matrix with area $2k+1$ x $2k+1$ as shown on lines 10 and 12

```
5  range=2*k+1;

10          a= floor( range/2 );
11          %get the section of the matrix to extract the median from
12          section=img( max(i-a,1):min(i+a,R), max(j-a,1):min(j+a,C));
13          %get the median
14          med= median(median(section));
15          %get output image
16          outImg(i,j) = med;
```

Functions like `floor()`, `min()` and `max()` are used to avoid out of boundary error and to implement the function for a generic value of k. To obtain the median we called the matlab function `median()` two times on the sub-image.

b. We loaded the `cktboard.tif` image and added salt and pepper noise with the `imnoise` function. Results are displayed in figure 5.

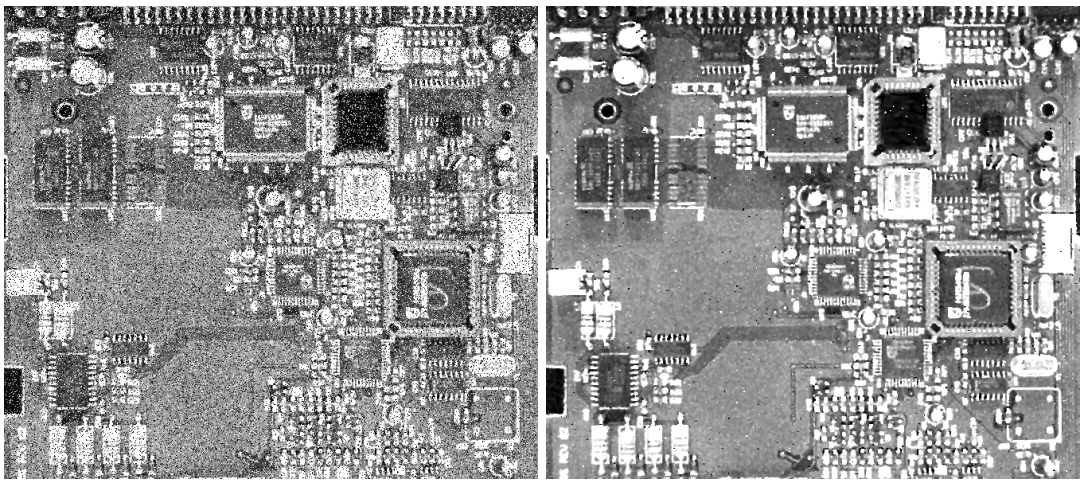


(a) The image: '`cktboard.tif`' (b) noisy version of '`cktboard.tif`'

Figure 5: Before and after addition of noise.

c. We tried to apply the median filtering to the image 5b by calling the function implemented earlier `IPmedian(noisy, 1)`. The parameter k, which is 1 in this example, is used to calculate the dimension of the area to consider. Since the area is defined as a $2k+1$ x $2k+1$, with $k=1$, we are going to have a 3x3 area. Results are shown in figure 6b. As we

can observe, the noise has been significantly reduced by the algorithm. It's also observable how the dots still remaining are bigger than one pixel and also of the same color (black or white). This means that, for certain agglomerates of pixels, since the noisy pixel were predominant over the normal pixels, the median for certain sub-images is the value of the noisy pixels itself. It may be possible to try and remove those pixels by applying the same algorithm again, hoping to shrink those noisy chunks with every iteration, until they're completely gone.



(a) noisy version of '`cktboard.tif`' (b) restored version of noisy '`cktboard.tif`'

Figure 6: Before and after addition of noise.

Appendix A The MATLAB code

```
function outputImage = IPftfilter(x,H)
 %centered transform
 fourierX = fft2(im2double(x));
 centeredFourierX = fftshift(fourierX);

 % filter
 centeredFourierOutputImage = centeredFourierX.* H;

 %inverse centered transform
 fourierOutputImage = ifftshift(centeredFourierOutputImage);
 outputImage = real(ifft2(fourierOutputImage));
end
```

Listing 5: IPftfilter

```
file = imread('characters.tif');
file = im2double(file);
H = IPbhpf(60, 2, size(file,1), size(file,2));
filteredImage = IPftfilter(file, H);
imshow(filteredImage);
```

Listing 6: Filtering 'characters.tif' with Butterworth high pass filter.

```
function out = IPmedian(img, k)
R = size(img,1);
C = size(img,2);

range=2*k+1;

for i=1:R
    for j=1:C

        a= floor( range/2 );
        %get the section of the matrix to extract the median from
        section=img( max(i-a,1):min(i+a,R), max(j-a,1):min(j+a,C));
        %get the median
        med= median(median(section));
        %get output image
        outImg(i,j) = med;
    end
end

imwrite(outImg, './restoredImage.png');

out=1;
end
```

Listing 7: IPmedian

References

[1] Rafael C. Gonzalez. *"Digital Image Processing"*. ISBN, 2018.