NTNU
Norwegian University of
Science and Technology

Digital Image Processing Assignment 2

Marius Blom

Sondre Jensen

2017

TDT4195 - Visual Computing Fundamentals

TABLE OF CONTENTS

Table of Contents i

List of Tables ii

List of Figures iii

1 Theory Questions 1
- 1.1 Convolution . . . 1
  - 1.1.1 Convolution theorem . . . 1
  - 1.1.2 Implementation of frequency filtering . . . 1
- 1.2 High- and low-pass filters are . . . 1
- 1.3 Kind of kernel, high- or low-pass . . . 1
- 1.4 More theory . . . 2
  - 1.4.1 Padding, frequency filtering . . . 2
  - 1.4.2 Pad images . . . 2
- 1.5 Even more theory . . . 2
  - 1.5.1 Transformed to the frequency domain by the Fourier transform . . . 2
  - 1.5.2 How does the spectrum change . . . 2
  - 1.5.3 Sinusoidal grating is rotated in the XY plane . . . 2
- 1.6 Avoid aliasing without increasing the sampling rate . . . 2

2 Frequency Domain Filtering 3
- 2.1 Frequency filtering . . . 3
  - 2.1.1 The filtered image(s) in the spatial domain . . . 4
  - 2.1.2 The spectrum of the image(s) before and after the multiplication . . . 5
  - 2.1.3 The spectrum of the convolution kernel(s) . . . 6

3 Unsharp Masking 7
- 3.1 Implement a unsharp masking function . . . 7
  - 3.1.1 Explain equation . . . 7
  - 3.1.2 Unsharp kernel spectrum . . . 8
- 3.2 Apply Unsharp function . . . 9
  - 3.2.1 Image in spatial domain . . . 9
  - 3.2.2 Spectrum before and after . . . 9

4 Selective Filtering 10
- 4.1 Removing noise from noise-a.tiff . . . 10
  - 4.1.1 Identify periodic noise . . . 10
  - 4.1.2 Create selective filter . . . 10
  - 4.1.3 Remove the noise . . . 11
- 4.2 Removing noise from noise-b.tiff . . . 12
  - 4.2.1 Create selective filter . . . 12
  - 4.2.2 Remove the noise . . . 13

LIST OF TABLES

LIST OF FIGURES

1.1 Prepending to appending padding 2

2.1 High pass filtered, sharper edges 4
2.2 Low pass filtered, smoother 4
2.3 High pass spectrum 5
2.4 Low pass spectrum 5
2.5 High pass kernel 6
2.6 Low pass kernel 6

3.1 Gaussian kernel spectrum 8
3.2 Gaussian filtered 9
3.3 Gaussian spectrum 9

4.1 Identifying noise in spectrum 10
4.2 Identifying noise in spectrum 11
4.3 Filtered image 11
4.4 Identifying noise in spectrum 12
4.5 Identifying noise in spectrum 13
4.6 Filtered image 13

1 THEORY QUESTIONS

1.1 Convolution

1.1.1 Convolution theorem

The convolution theorem entails that a Fourier transformation for a convolution is the same as the dot product. Which will say that if f and g are two functions, then: $\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$ and $\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$

1.1.2 Implementation of frequency filtering

If we have a function g in the spatial domain

$$g = f * h$$

To do a convolution of a function is generally an expensive operation. A better solution is to take the Fourier of f and g, then multiply those together to get the new Fourier transform G. Then we could do the inverse of G Fourier transform to get back to g as seen in equation 1.1

$$g = f * h \tag{1.1a}$$
$$G = \mathcal{F}\{f * h\} \tag{1.1b}$$
$$G = \mathcal{F}\{f\} \cdot \mathcal{F}\{h\} \tag{1.1c}$$
$$\mathcal{F}^{-1}\{G\} = g \tag{1.1d}$$

1.2 High- and low-pass filters are

A low-pass filter is a function that is tasked with filtering out high frequency. This is used in Image Processing as a kernel that remove high intensities, which is useful in order to remove noise from an image. This result in a more smoothed image. A high-pass filter have the opposite task, all intensities lower than the cutoff frequency is filtered out. This result in noise and edges getting amplified.

1.3 Kind of kernel, high- or low-pass

Figure a

Is low-pass kernel, where all high frequency components outside of the center are cut off.

Figure b

Is a high-pass kernel. It blocks more frequencies in y direction, hence the line in the middle. The kernel lets higher frequencies pass, especially in x directions and thereby work as a high-pass filter.

Figure c

Is high-pass kernel, where all low frequency components inside of the center are cut off.

1.4 More theory

1.4.1 Padding, frequency filtering

It is important to do padding when preforming frequency filtering since the dimensions of the filter and the image are most likely of different sizes. We are able to extend the dimension of the image to the same as the filter by adding zeroes to the image.

1.4.2 Pad images

The result of the frequency filtering will not change. The only ting that will change is the location the actual image will end up. If the image has prepending zeroes, will the frequency filtered image end up with appending zeros. This is illustrated in figure 1.1. This is because the kernel we multiply with is centered with zero padding around. This will result in a fair multiplication no matter what padding scheme is used.

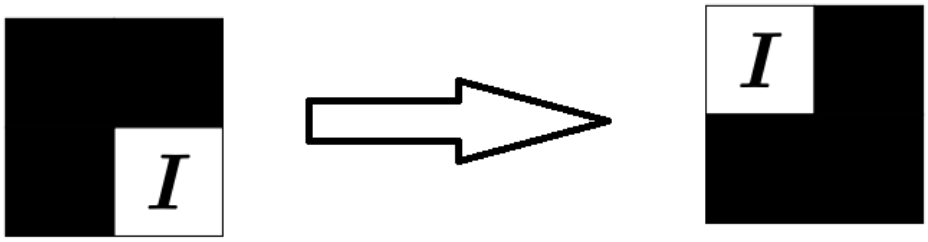

Figure 1.1: Prepending to appending padding

1.5 Even more theory

1.5.1 Transformed to the frequency domain by the Fourier transform

The spectrum will show two dots in the frequency domain. One above and one below the center. These are the spikes for the frequency of this particula sinusoid.

1.5.2 How does the spectrum change

The spectrum will still show two dots in the frequency domain but this time the dots will be further apart since the sinusoidal grating increase.

1.5.3 Sinusoidal grating is rotated in the XY plane

When the sinusoidal grating is rotated, then the dots in the frequency domain will change location. If you look at it as a coordinate system then in this particular case there will be a dot in the first quadrant and a dot in the third quadrant.

1.6 Avoid aliasing without increasing the sampling rate

We can limit the bandwidth with for example a low-pass filter. This will smooth the image, resulting in less aliasing without increasing the sampling rate.

2 Frequency Domain Filtering

2.1 Frequency filtering

```
def spatial2FrequencyDomain(image, kernel):
        #Find dim of image
        ySize = np.size(image,0)
        xSize = np.size(image,1)
        #Padd the image
        image = np.lib.pad(image, ((ySize,0), (xSize,0)), 'constant', con
        kernelSize = np.size(kernel,0)
        kernel = np.lib.pad(kernel, ((ySize-(kernelSize//2),ySize-(kernel
        #Centering
        for y in range (2*ySize):
                for x in range (2*xSize):
                        image[y,x]=image[y,x]*np.power(-1,x+y)
                        kernel[y,x]=kernel[y,x]*np.power(-1,x+y)
        #Converting to frequency domain
        F = np.fft.fft2(image)
        H = np.fft.fft2(kernel)
        return(F, H)
```

```
def frequency2SpatialDomain(G):
        ySize = np.size(G,0)
        xSize = np.size(G,1)
        g = np.zeros((ySize//2, xSize//2))
        g_p = np.fft.ifft2(G)
        g_p = np.real(g_p)
        #Centering
        for y in range (ySize):
                for x in range (xSize):
                        g_p[y,x]=g_p[y,x]*np.power(-1,x+y)
        #Restore padded image
        for y in range (ySize//2):
                for x in range (xSize//2):
                        g[y,x] = g_p[y,x]
        return g
```

```
(F, H) = spatial2FrequencyDomain(image,kernel)
G = F * H
g = frequency2SpatialDomain(G)
```

2.1.1 The filtered image(s) in the spatial domain

We provided the function $spatial2FrequencyDomain$ with the image and kernel that we wanted to use. The function found the size of the image and the kernel and added zero-padding. We then created a loop to center it before we converted the image and kernel into the frequency domain. Then we could do a simple multiplication of the image and kernel to find the filtered image in the frequency domain. After that we ran the $frequency2SpatialDomain$ with the filtered image that converted the image back to the spatial domain. The result can be seen in figure 2.1 and figure 2.2.

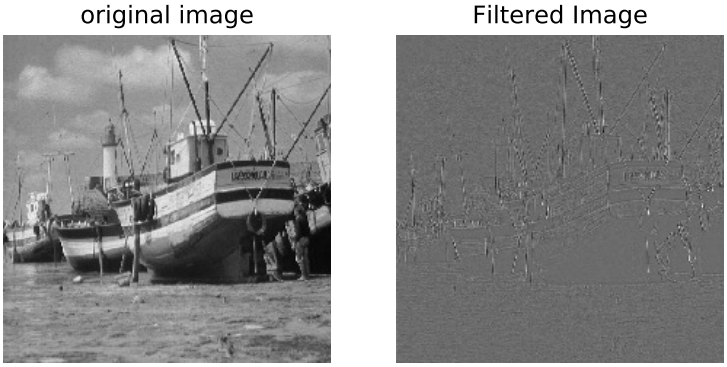

Figure 2.1: High pass filtered, sharper edges

Figure 2.2: Low pass filtered, smoother

2.1.2 The spectrum of the image(s) before and after the multiplication

We can see the spectrum of the original image and the filtered image from figure 2.3 and figure 2.4.

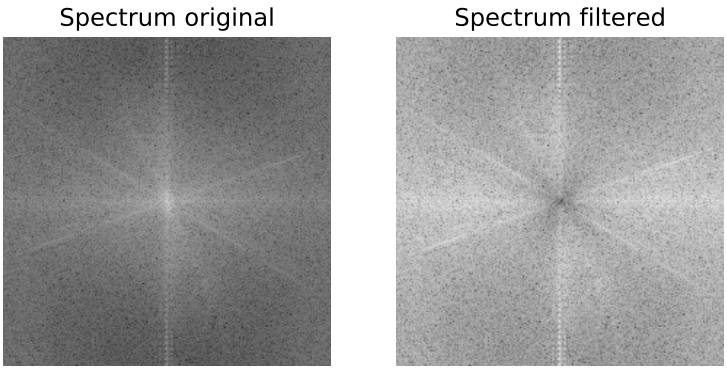

Figure 2.3: High pass spectrum

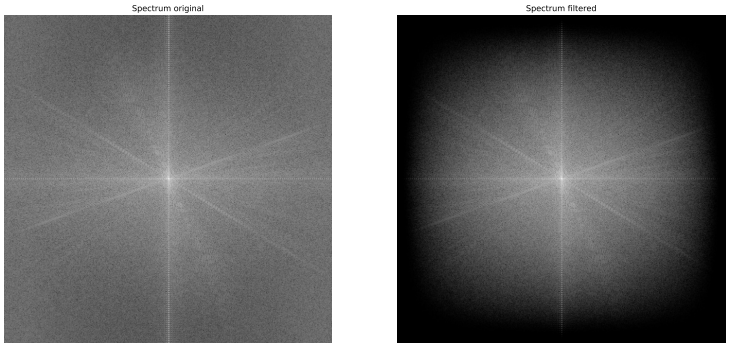

Figure 2.4: Low pass spectrum

2.1.3 The spectrum of the convolution kernel(s)

We used a 3x3 high-pass filter in figure 2.5.

```
kernel_highpass_3x3 = (1/9) * np.array([
        [-1, -1, -1],
        [-1,  8, -1],
        [-1, -1, -1]], dtype=np.float32)
```

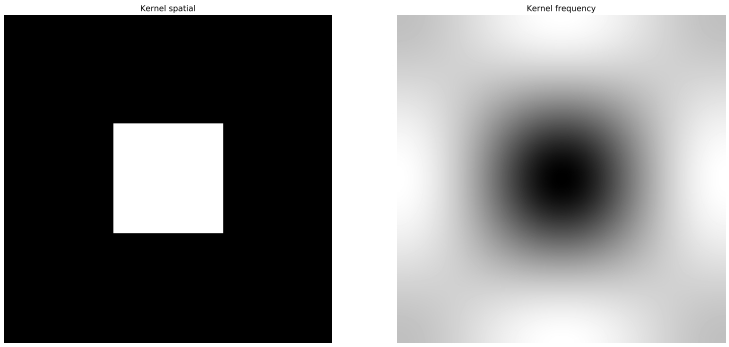

Figure 2.5: High pass kernel

We used a 5x5 lowpass filter on figure 2.6

```
kernel_lowpass_5x5 = (1/256) * np.array([
  [1, 4, 6, 4, 1],
  [4, 16, 24, 16, 4],
  [6, 24, 36, 24, 6],
  [4, 16, 24, 16, 4],
  [1, 4, 6, 4, 1]])
```

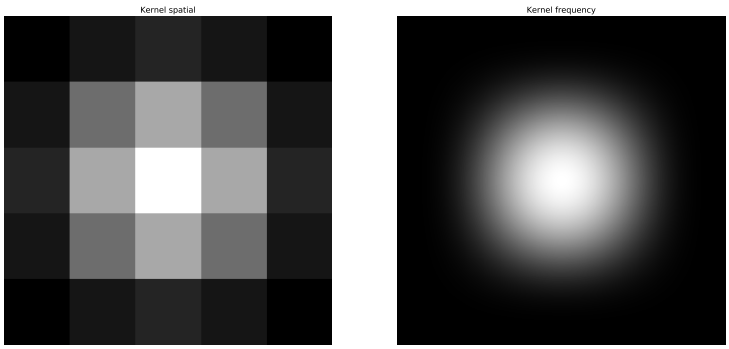

Figure 2.6: Low pass kernel

3 UNSHARP MASKING

3.1 Implement a unsharp masking function

The unsharp kernel where found using the code bellow.

```
def unsharpning_kernel(kernel):
        # Find size of array
        size = np.size(kernel,0)
        # set number 1 to be center of the impulse
        impuls = np.array([[1]])
        # pad with zeroes
        impuls = np.lib.pad(impuls, ((size//2,size//2), (size//2, size//2
        # Equation
        kernel = impuls + (impuls - kernel)
        return kernel
```

3.1.1 Explain equation

By taking basis in the equation given in the task were we able to calculate the unsharp kernel.

$$Kernel_{Sharp} = \delta + 1 * (\delta - Kernel_{Gaussian}) \tag{3.1}$$

We replaced the image from the original equation with the Dirac delta function and the smoothed image with the Gaussian kernel, which would have been used to achieve a smoothed image. By calculating equation 3.1 were we able to create a new kernel, which would result in a sharper image when applied.

3.1.2 Unsharp kernel spectrum

```
kernel_Gausian_5x5 = (1/256) * np.array([
  [1, 4, 6, 4, 1],
  [4, 16, 24, 16, 4],
  [6, 24, 36, 24, 6],
  [4, 16, 24, 16, 4],
  [1, 4, 6, 4, 1]])
```

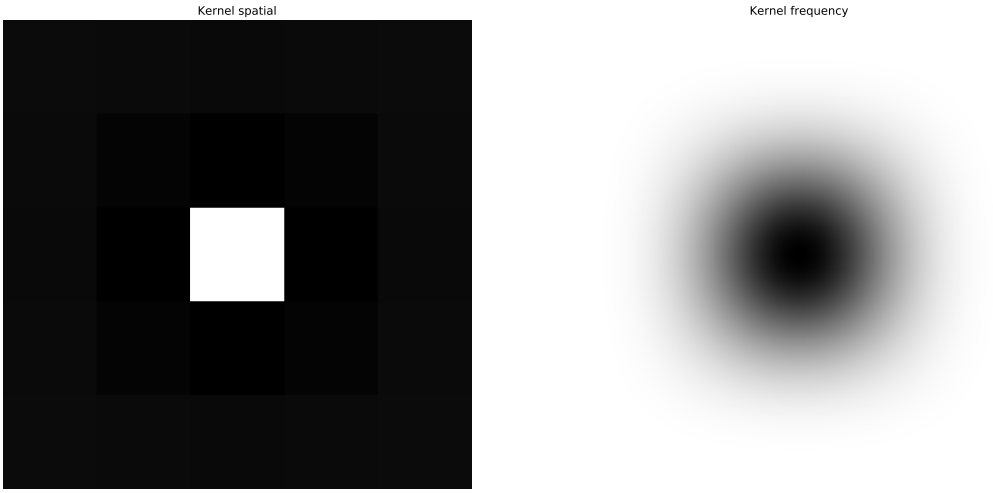

Figure 3.1: Gaussian kernel spectrum

3.2 Apply Unsharp function

3.2.1 Image in spatial domain

Figure 3.2: Gaussian filtered

3.2.2 Spectrum before and after

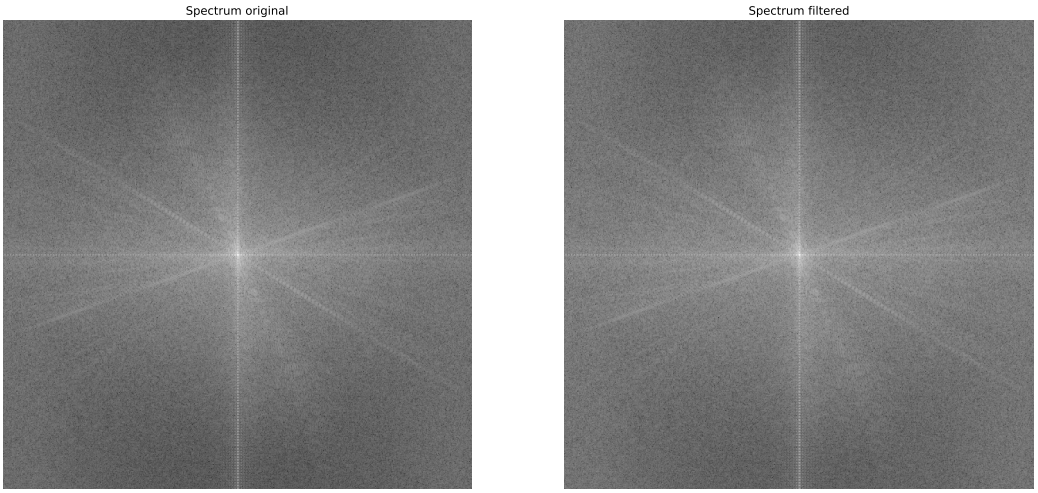

Figure 3.3: Gaussian spectrum

4 Selective Filtering

4.1 Removing noise from noise-a.tiff

4.1.1 Identify periodic noise

We created a small Matlab script to be able to display the spectrum of the unfiltered image. Then we identified the periodic noise, as seen in figure 4.1

```
originalImage = imread('noise-a.tiff');
spec_origin = fft2(double(originalImage));
spec_img = fftshift(spec_origin);
figure;
spec_img = log(1+abs(spec_img));
imshow(spec_img,[]);
```

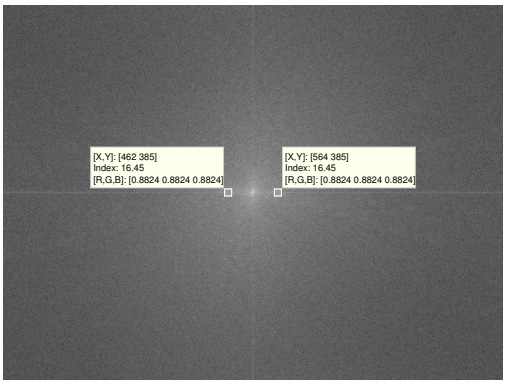

Figure 4.1: Identifying noise in spectrum

4.1.2 Create selective filter

By using the points found in Matlab, we created a function in Python to filter the image.

```
def filtering(image):
        for x in range(552,576):
                for y in range(384,386):
                        image[y,x] = 0
        for x in range(450,474):
                for y in range(384,386):
                        image[y,x] = 0
        return image
```

4.1.3 Remove the noise

We converted the image to the frequency domain and plotted the spectrum of the image as can be seen on the left side in figure 4.2. We then ran the image through our filtering function that corrected the noise. We tuned the range until we got the result we were happy with. The changes to the spectrum can be seen in the right side of figure 4.2

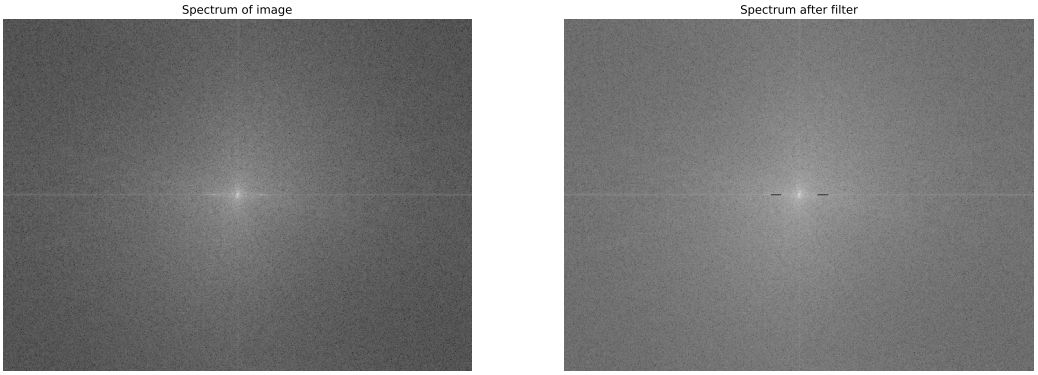

Figure 4.2: Identifying noise in spectrum

After that we converted the image back to spatial domain as seen in figure 4.3. If looked on closely it is possible to still see some noise on the edges of the image.

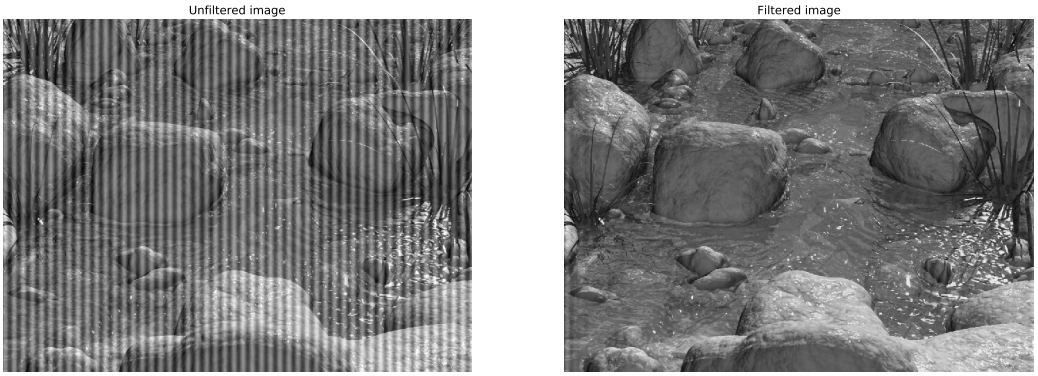

Figure 4.3: Filtered image

4.2 Removing noise from noise-b.tiff

We used the same apporach as with the first image. Found the spectrum in Matlab and got some coordinates. It is clear that some of the noise is formed as a cirlce, so we wanted to find the center of the circle and radius as seen in figure 4.4. By using these points we made a function in Python, which basically does the same as the earlier task.

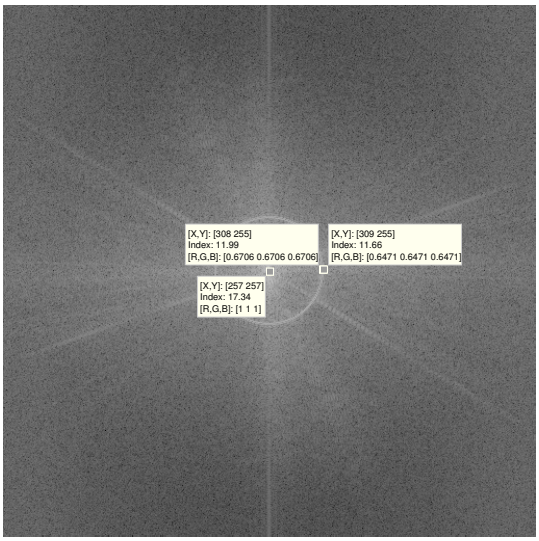

Figure 4.4: Identifying noise in spectrum

4.2.1 Create selective filter

```
def filtering(image):
        width, height = 11, 11
        centerX, centerY = 257, 257
        radius = 52
        epsilon = 19
        # draw the circle
        for y in range(100, 400):
            for x in range(100, 400):
                # see if we're close to (x-a)**2 + (y-b)**2 == r**2
                if abs((x-centerX)**2 + (y-centerY)**2 - radius**2) < eps
                    image[y][x] = 0
        return image
```

4.2.2 Remove the noise

We focused on the circle when we converted the image to the frequency domain. Our function marked the circle like we wanted, as seen to the right in figure 4.5

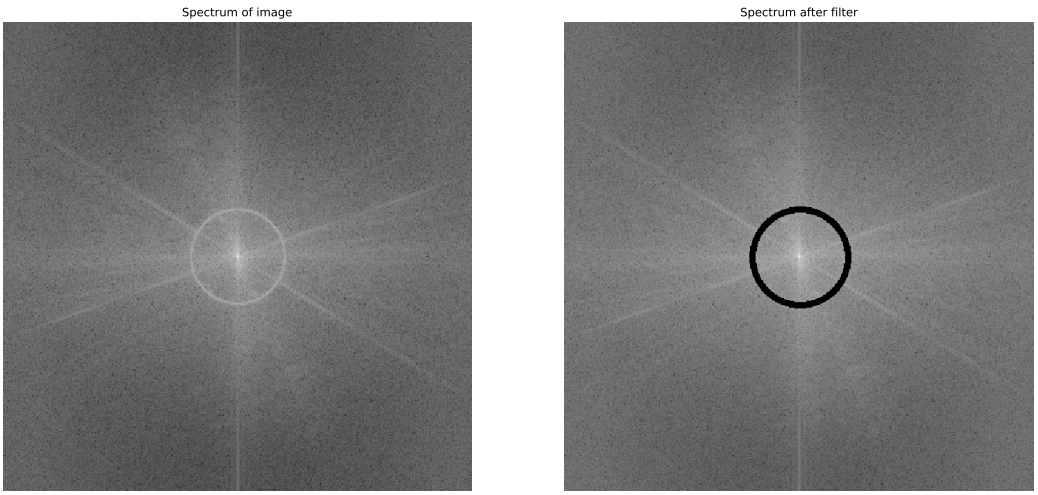

Figure 4.5: Identifying noise in spectrum

It is clear that much of the periodic noise have been removed after converting the image back to the spatial domain. We did not spend as much time to adjust the parameters for this filter, so the result is not as good as it could be. But there is still a major difference between the unfiltered and filtered image, as can be seen in figure 4.6.

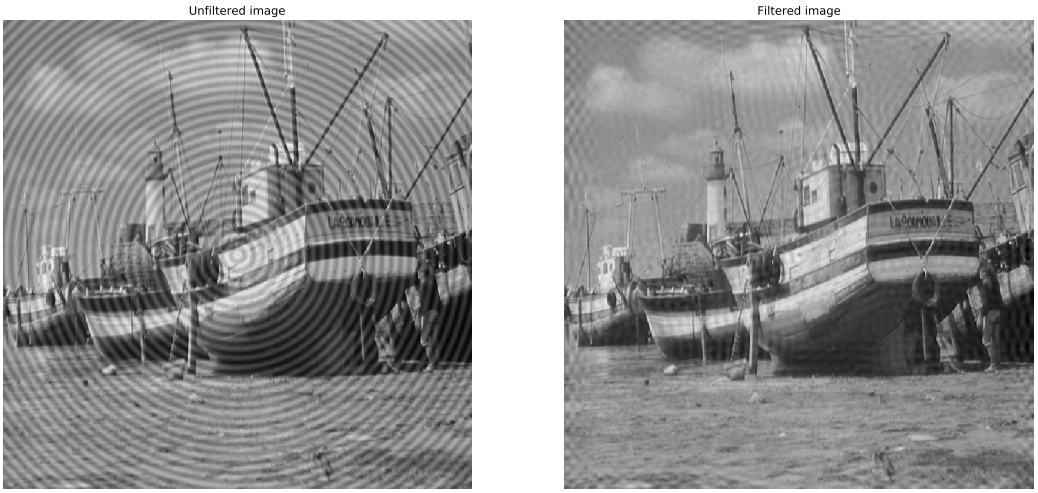

Figure 4.6: Filtered image