HOME EXAM

FYS-2010-Digital image processing

March 27, 2019

Candidate number: 25
University of Tromsø

Contains 32 pages, including frontpage.

Contents

1 Part A

1.1 1

We have the operators, $S_1 = [-1,0,1]$ and $S_2 = [1,2,1]$.

forming the outer product of the vectors we get,

$$g_x = s_1 \otimes s_2 = \begin{bmatrix} -1\cdot 1 & -1\cdot 2 & -1\cdot 1 \\ 0\cdot 1 & 0\cdot 2 & 0\cdot 1 \\ 1\cdot 1 & 1\cdot 2 & 1\cdot 1 \end{bmatrix}$$

which yields,

$$g_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

and $g_y = s_1 \otimes s_2$ yields,

$$g_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

These are the Sobel operators.

1.2 2

In (b) the operator g_x is used as it increases the edges in the y-direction, the reason for this is that we have zeros in the center and in horizontal direction. When convolving we get the changes in the y-direction(top and bottom difference in the matrix) assigned to the center pixel. In image (c) the g_y kernel has been used as it has enhanced the x-directional edges(changes in x-direction).

1.3 3

The bit plane which represents the most significant bits are the one with most details which is b_7. By setting the two least significant bits to zero shift the intensities to only be $4\cdot k$ intensity, where k is an integer. So here we would only have 4 shades between the intensities in the image. (the human eye cannot perceive the difference of abrupt change of such few intensities [p.626 Rafael C. Gonzales [2018]]). The histogram would look like figure(1)

1.4 4

Each pixel in an 8-bit image consist a byte consisting of numbers between 1 and 0, example $[1,1,1,0,0,0,0,0]$, we have bit planes $b_0 - b_7$, where b_7 is the most significant bit plane. If we have a pixel with the byte, $[1,0,0,0,0,0,0,0]$ its intensity in integer values is 128. If we only allow the last bit to change and rest always zero, $[1,\underbrace{0,0,0,0,0,0}_{always=0}]$ then we either have the intensity 0 or 128. All values above 128-255 will use the last bit in combination with the other bits, therefore we do the following transformation,

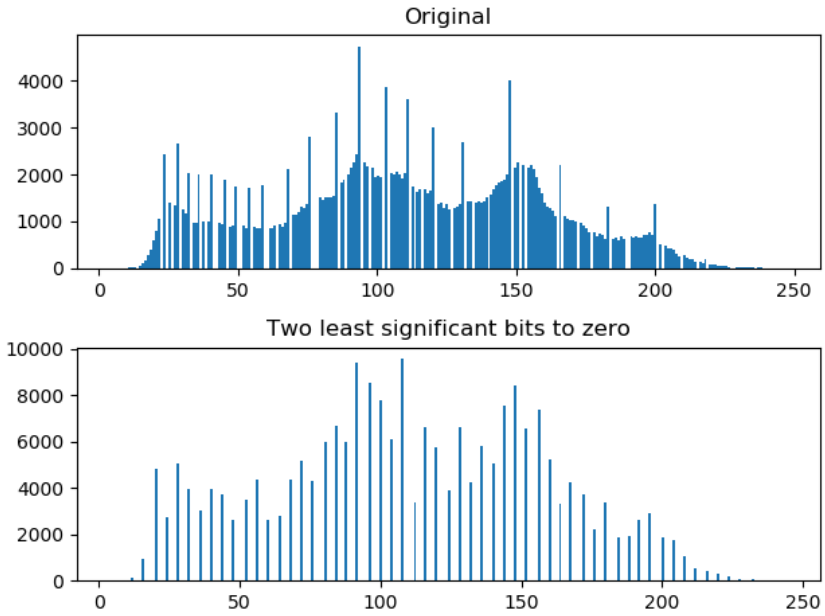

Figure 1: How the histogram changes when setting the two least significant bits(LSB) to zero. Observe how the intensities are shifted to its closest intensity which does not use the last two bits $(1,2,3) \rightarrow [01,10,11]$ every intensity from 3 to 1 is also zero.

$$T(r) = \begin{cases} 0, r \in \{0 \leq r \leq 127\} \\ 128, r \in \{128 \leq r \leq 255\} \end{cases} \tag{3}$$

This will be the most significant bit-plane as it contains the most intensities. Its also possible to store information of the two most significant bits of an watermark and place them into the least significant bits of the original image, and the two last bits are insignificant in how a person perceive the image, you can watermark images without anyone seeing it. This can then be retrieved with an algorithm.

1.5 5

To invert the image we want the follow conditions: black -> white and white -> black. As intensity value representation: 0 -> 255 and 255 -> 0

Using the transformation equation given,

$$T(r) = ar + b$$

Putting in the conditions,

$$T(255) = a \cdot 255 + b = 0 \tag{4}$$

$$T(0) = a \cdot 0 + b = 255 => b = 255$$

Putting b back into equation(4) we get,

$$T(255) = a \cdot 255 + 255 = 0$$

which yields the solution

$$T(r) = -r + 255 \tag{5}$$

We could also replace 255 with $L-1$ in a more general case.

2 Part B

2.1 1

Spatial aliasing is loss of information by not using sufficient sampling rate, depending on the change of frequencies in the intensities one want to have an image of. This follows from the Nyquist sampling theorem which says we need to sample with $f_s = 2f$ where f is the *real* frequency and f_s is the sampling frequency. If we use a checkerboard as an example,

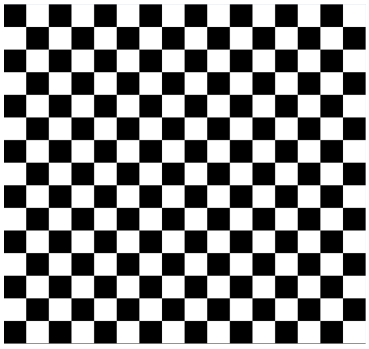

Figure 2: An image of a checkerboard. Image taken from [Rafael C. Gonzales [2018]].

If we sample with less pixels (or increase the frequency rate(the rate of changing intensities) of the object we are imaging) we could end up with only getting every cycle of the black intensity such that we only get a black strip. We could also use a different sampling rate and end up getting a "Fake" image by reconstructing every second black intensity with one white in between such as shown in the sliced checkerboard in figure(3) below. By sampling with less pixels we could end up having the "same" image,

Figure 3: Aliased slice

but with a longer period between the intensities. This also applies to resizing because when we resize down an image we need to remove pixels, which would destroy some of the information in the image. If we then resize it to the original shape we would apply interpolation and would most likely change the real intensity valued pixels to some other neighboring pixel value and therefore damage the quality of the image.

2.2 2

In figure 4.19 in the book [Rafael C. Gonzales [2018]] we can clearly see the aliasing effect in the area of the high frequency pixels. On the scarf, pants and the chair in the background the intensity values changes rapidly(high frequency). Therefore when resizing we would change the sampling rate below the Nyquist and lose information which we cant retrieve when resizing back to original shape.

2.3 3

We resize the image to 50% of its original size, the result can be shown in figure(4) . The resizing is done by only taking out every second pixel in the original image into an new image.

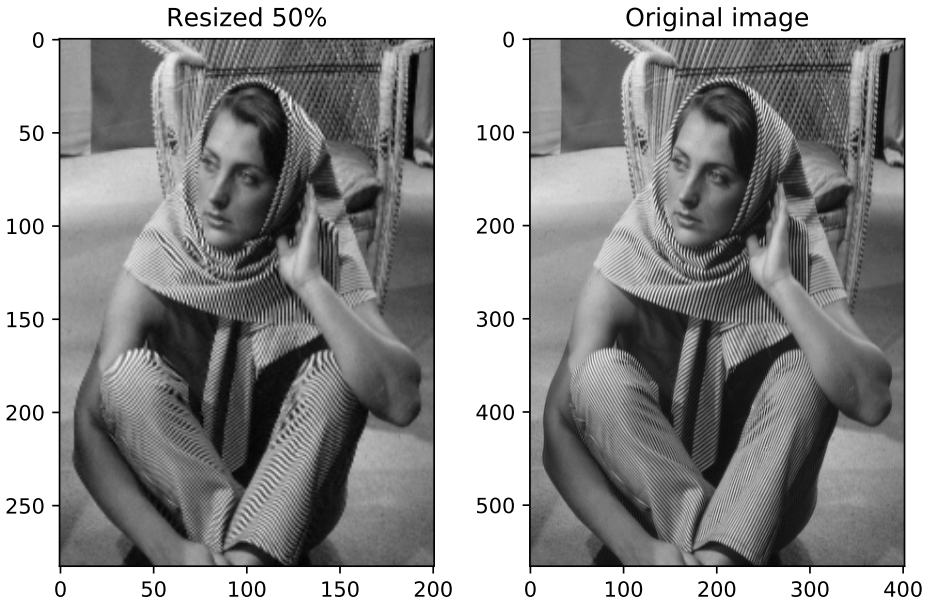

Figure 4: Resized image to 50% of its original size, we can clearly observe the aliasing in the high frequency areas of the image.

In the figure we can see the aliasing effect as the high frequency areas wont be sampled at the right sampling rate which makes it impossible to see the real pattern.

2.4 4

We can blur the images prior to resizing by convolving an averaging filter with the image as follows,

$$g(x,y) = h(x,y) \bigstar i(x,y) \tag{6}$$

where h is the average filter defined as in equation(7) and i(x,y) is the image, $\bigstar$ means convolution.

$$h = \frac{1}{MN}\begin{bmatrix} 1_{1,1} & 1_{1,2} & 1_{1,3} & \dots & 1_{N,1} \\ 1_{1,2} & 1_{2,2} & 1_{3,2} & \dots & 1_{N,2} \\ & \dots\dots\dots\dots & & & \\ 1_{1,M} & 1_{2,M} & 1_{3,M} & \dots & 1_{N,M} \end{bmatrix} \tag{7}$$

when convolving we are applying the filter in the spatial domain, since this is an averaging filter/-moving average(as it traverses the image), we essentially apply a lowpass filter. Lowpass filter will only allow the low frequency components of the image to go through. This means that the image will not have rapid changing intensities(more smudged out/smooth). This is clearly visible in figure(5, this will decrease the effect of aliasing. When applying a filter we want to zero pad our image, in the case of convolving in spatial domain we would have problems fitting the kernel on the edges of the image. In our case the the convolve method from scipy module in python does this.

```
        # Convolve using scipy package
        from scipy.signal import convolve2d
        mask = conconvolve2d(image, Laplacian, mode = 'same')

```

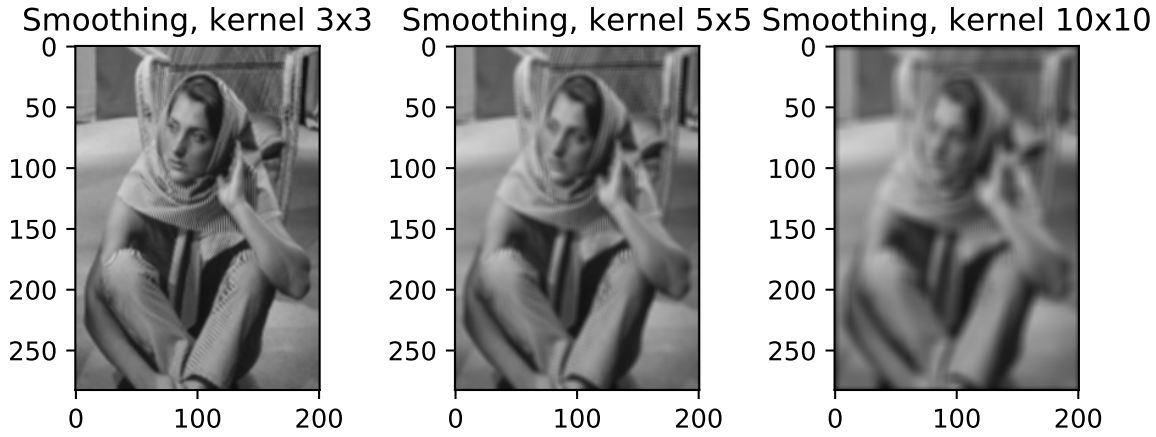

Figure 5: Smoothed the aliased image with different sizes of the averaging kernel. We can see here that the 10x10 kernel has strong blurring. The aliasing seems to go away as we blur the image.

2.5 5

We can try to restore the image by using an sharpening filter we would emphasize the edges better by using filters such as Laplace. Laplace is a second order derivative so here we would emphasize pixels with high frequency and de-emphasize areas with low frequency (high pass filter). We use the Laplace kernel given in equation(8),

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{8}$$

This stems from the fact that we use the Laplacian $\nabla^2 f(x,y)$ which will generate the discrete version $f[x,y+1]+f[x+1,y+1]+f[x+1,y]+f[x-1,y]+f[x-1,y-1]+f[x+1,y-1]-4f[x,y]$(summing to zero yield no response in areas of constant intensity). This is linear and therefore allowed to be used in convolution. By doing this convolution we get a mask g(x,y),

$$g(x,y) = h(x,y) \bigstar i(x,y) \tag{9}$$

here h(x,y) is the Laplacian filter and i(x,y) is the image. To enhance the edges we add this mask to our image. By using the Laplace kernel in equation(8) we will sharpen in all direction. We could for instance use a "plus-shaped" Laplace to neglect diagonal direction. (set diagonals in the matrix to zero.)

$$E(x,y) = i(x,y) + Cg(x,y) \tag{10}$$

Where E(x,y) is the enhanced image and C is an constant which is either -1 or 1 depending on which Laplacian kernel we use. In this case $c = -1$.

The result of the sharping with two different methods are shown in figure(6).

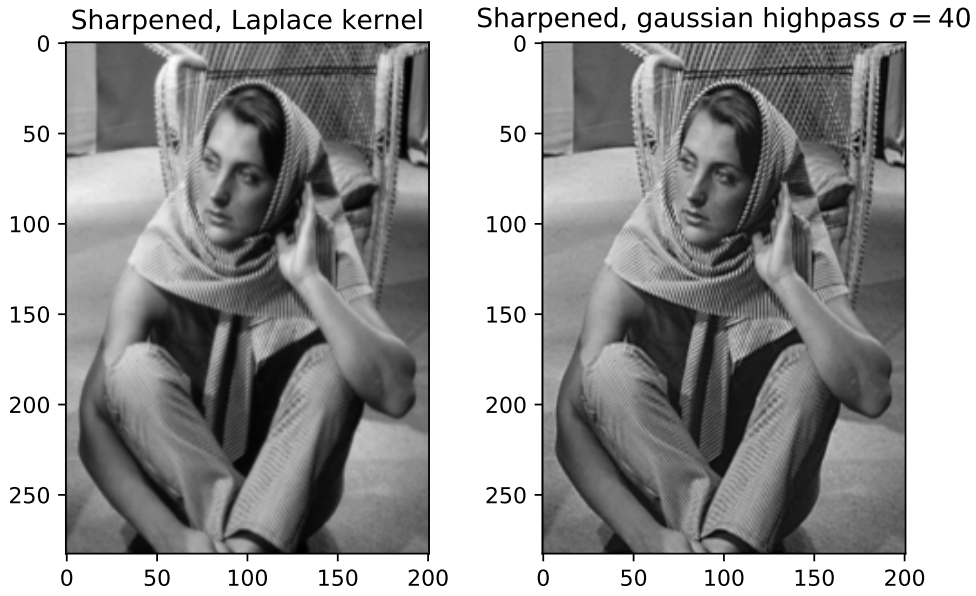

Figure 6: Sharpened by using two different sharping techniques. Remark: Laplace sharping was done in spatial domain and gaussian high pass was done in frequency domain.

In this figure we can see that it has more defined edges. The gaussian high pass sharpening method looks alot like the Laplacian, but we could tune the parameter σ(cut off) to fit better. The image is much better then the original resized version which contained aliasing.

The gaussian high pass filter uses the gaussian function (which is a lowpass filter),

$$H_{lp}(u,v) = e^{-D^2/2\sigma^2} \tag{11}$$

This function is in the frequency domain, D is the distance from the centered frequency and σ is the radius parameter. To get the high pass filter we need to subtract it from one.

$$H_{hp}(u,v) = (1 - H_{lp}) \tag{12}$$

To apply this filter we multiply it by the centered frequency domain representation of the image I(u,v) and add the original, all in frequency domain as shown in equation(13). This is also called high-frequency emphasis filter, which also contains a constant to tune it, but here we used k = 1.

$$G(x,y) = (1 + H_{hp})I(u,v) \tag{13}$$

In the previous case we blurred the image PRIOR to resizing, this made the image look good and without aliasing. We have these images in figure(7), now smoothed after resizing.

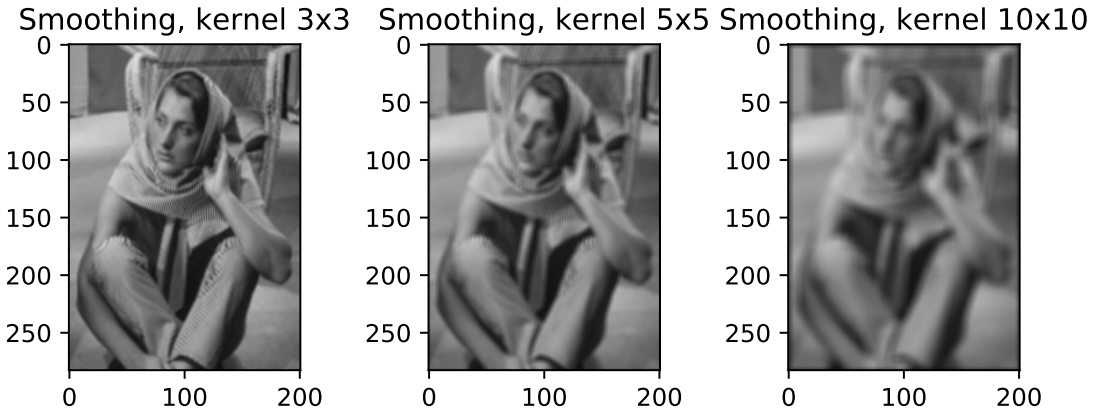

Figure 7: Smoothed images on images blurred after resizing.

If we blur the image after resizing and using Laplace sharping we get the images in figure(8).

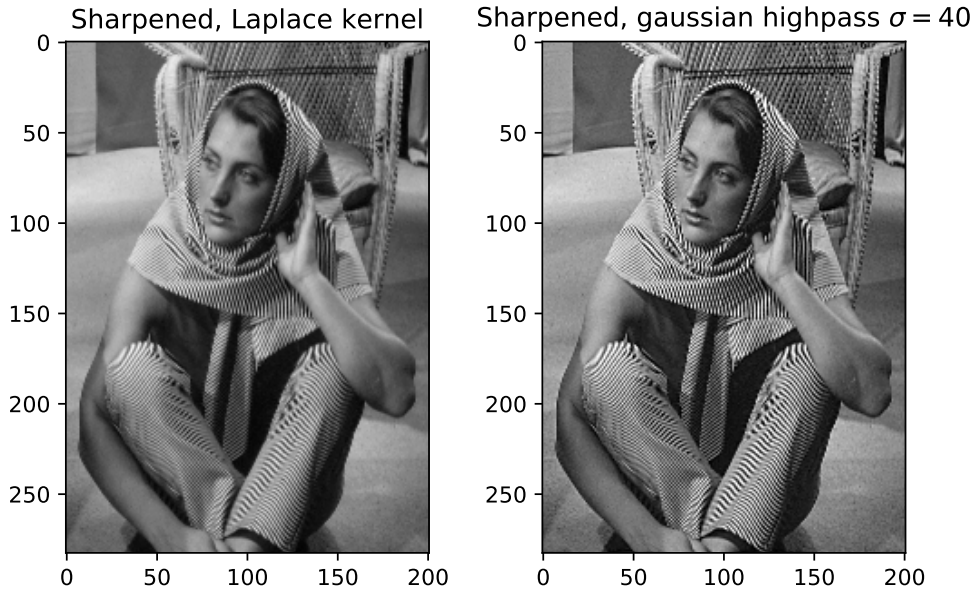

Figure 8: Sharpened by using two different sharping techniques on image blurred after resizing. Here we can clearly see that aliasing is visible again.

In this case we can see that the aliasing is back, so we see how sharpening works counter to blurring.

3 Part C

3.1 1

By visually inspecting the images,in *F1.png* we can clearly see the distinctive *salt & pepper* noise with its black/white broken pixels. In *F2.png* it looks like gaussian noise, in *F3.png* we have vertical periodic noise, in *F4.png* we have horizontal periodic noise. In the last image *F5.png* we have a superposition of horizontal and vertical periodic noises. The images of shown in figure(9).

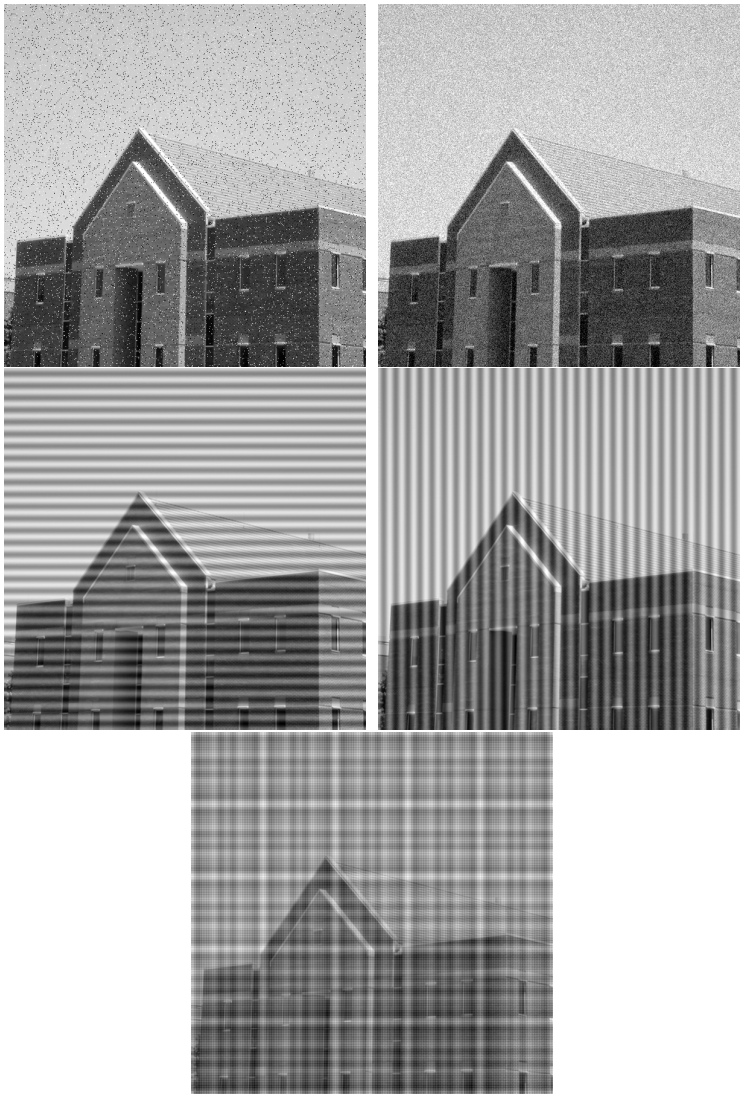

Figure 9: Image F1.png, F2.png, F3.png, F4.png, F5.png

We could find the histogram representation of the image to see the noise pattern, in figure(10) we can see that the gaussian noise form gaussian structures in the histogram. The salt and pepper noise

has large spike at the white and black intensity(0 and 255), but the overall structure of the image stays intact, its much harder to know the structure of the image with the gaussian noise.

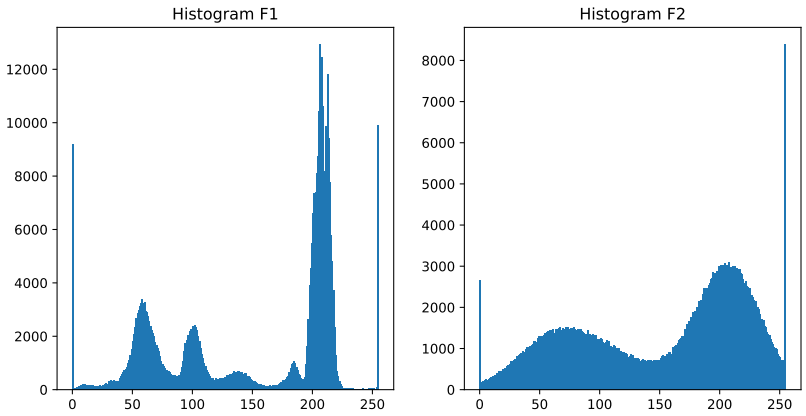

Figure 10: Histogram of the images. It looks like we have two different gaussian noise distributions in image F2.

The periodic noises would be a lot easier to see in the frequency domain as sine waves in frequency domain look like conjugated impulses. In figure(11) we can see the impulses.

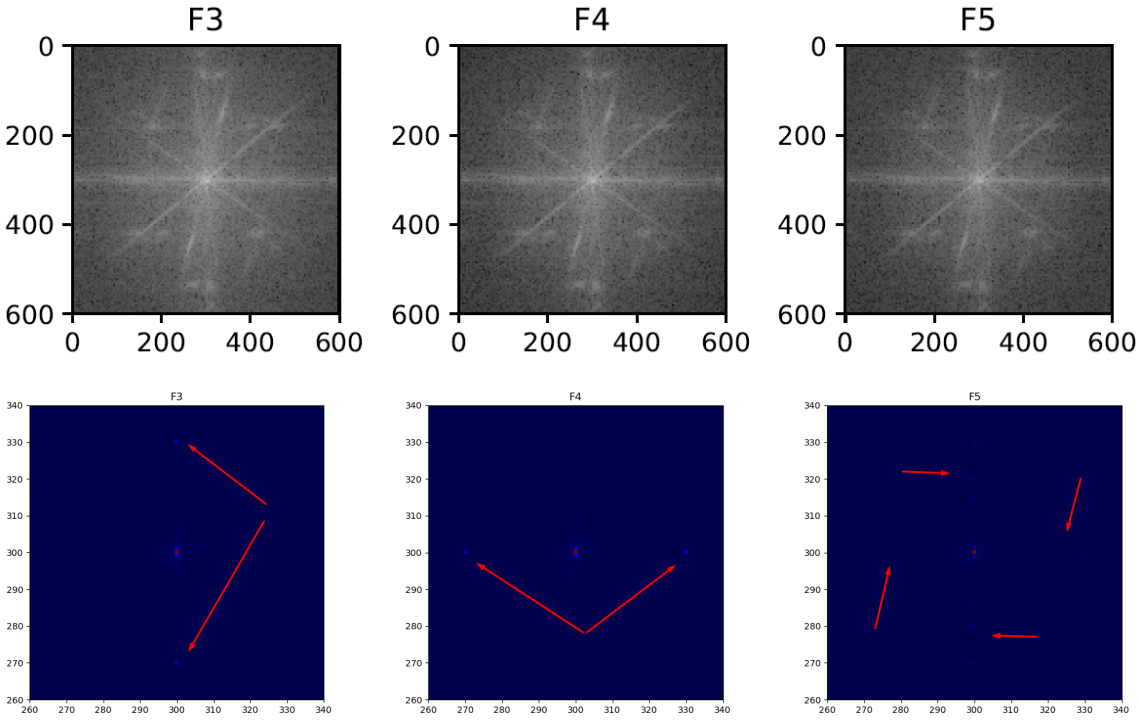

Figure 11: First row show the frequency representation of the images in dB scale. If we do not use dB scale we get higher difference between the energy of the spectrum, but by zooming in we can clearly see the impulses from the periodic noise.

3.2 2

For removing salt and pepper noise a median filter would be a good choice. With the median filter we rank the values from lowest to highest value and choose the 50% percentile intensity value in the kernel and set this value as the new pixel intensity of the new image. We do this for all the pixels in the image.

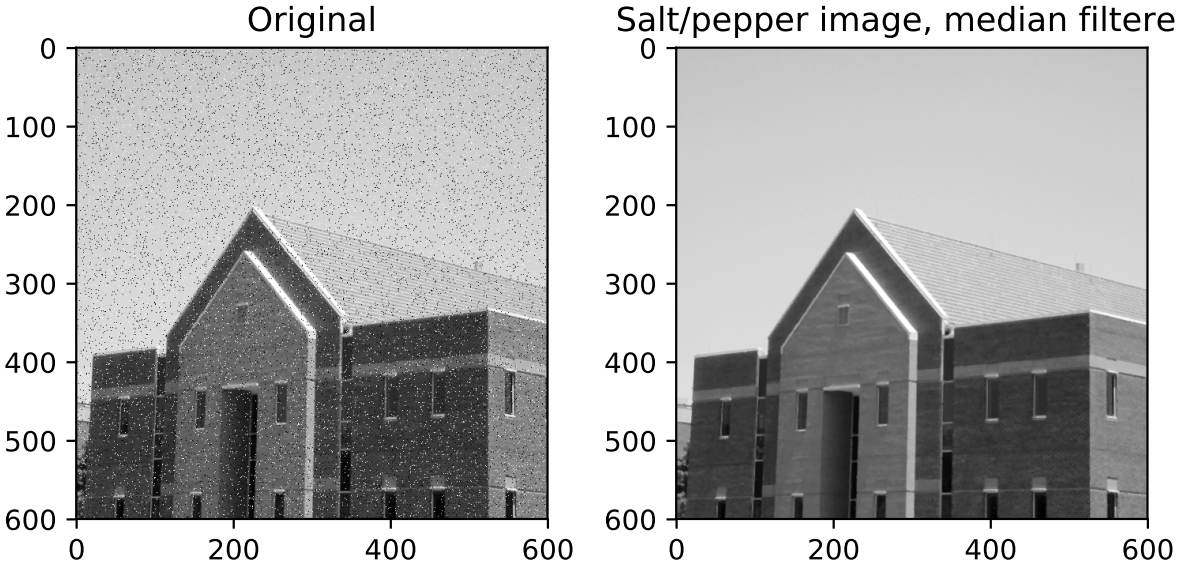

Figure 12: F1(salt & pepper) image filtered using median filter with a 3x3 kernel.

Here g(r,c) is the values from the image which are inside our convolving kernel.

For the gaussian noise image F2 we use an adaptive filter defined as,

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_{S_{xy}}}\left[g(x,y) - \bar{z}_{S_{xy}}\right] \tag{14}$$

Here $\hat{f}(x,y)$ is the filtered image, $g(x,y)$ is the noisy image, σ_η^2 is the variance of the noise, $\sigma_{S_{xy}}$ is the variance inside the kernel which is traversing the image, $\bar{z}_{S_{xy}}$ is the mean in of the kernel. Here we found the estimated variance of the noise by finding the variance in an sub image which taken to be the top part of the image which has mostly low frequency in the original image. The variance was found by using equation(15).

$$\hat{\sigma}_\eta^2 = \sum_{i=0}^{N-1} \frac{(r_i - \bar{r})^2}{N-1} p_s(r_i) \tag{15}$$

where r is the intensity value and $\bar{r}$ is the mean intensity value which is given by equation(16),

$$\bar{r} = \sum_{i=0}^{L-1} r_i p_s(r_i) \tag{16}$$

where L is the max intensity value, (8-bit yield $L = 256$) and $p_s(r_i)$ is the probability estimates of the intensity values.

It looks better, but we could probably do better with some other technique. If we look at the histogram in figure(10) we might suspect that we have salt & pepper + gaussian noise.

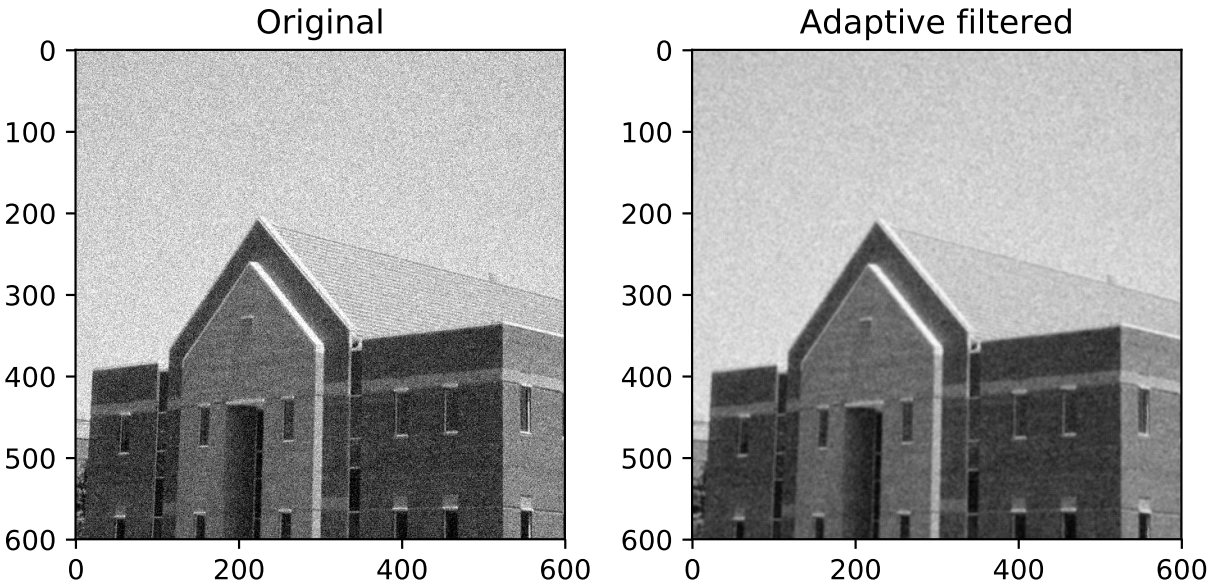

Figure 13: Filtered gaussian noise image, here we applied the adaptive filter discussed above with kernel 4x4

Since its suspected that there might be salt & pepper + gaussian noise we also have the alpha trimmed mean filtered image shown in figure(14),

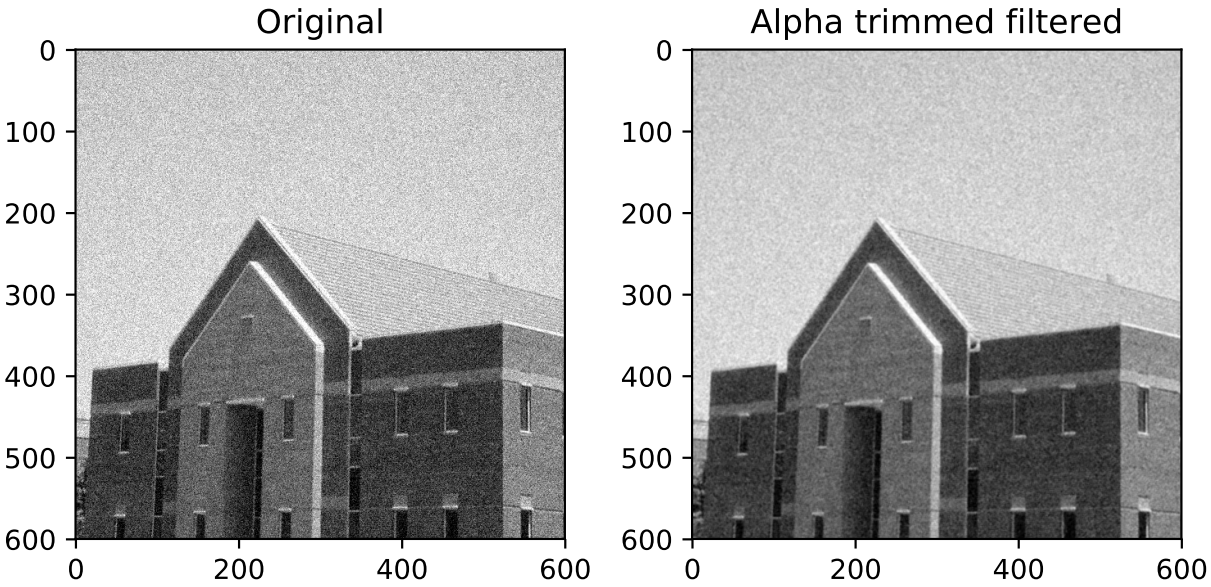

Figure 14: Alpha trimmed with kernel 3x3, d = 2. This image looks almost the same as the other adaptive filter, but slightly sharper

.

Alpha trimming is done by removing the $d/2$ of the first and last intensities that has been ranked from lowest to highest inside the kernel(like median ranking, except not taking the median). Then we take the mean of the rest and assign this new value to our new image which will be our filtered version.

3.3 3

The procedure for filtering in frequency domain is,

- 1. Obtain the image f(x,y) which is of size MxN.
- 2. Calculate new dimensions P and Q, P = 2M, Q = 2N.
- 3. Pad image i(x,y) to dimensions PxQ with zeros-, mirror-, or replicate padding. mirror padding will mirror the pixels at the borders. Pad the image at the bottom and right side to make it easier to slice out the image later.
- 4. Shift the image to center it in frequency domain, this makes it easier to do the multiplication right. This is done by multiplying the image as such $i(x,y)(-1)^{x+y}$.
- 5. Compute the DFT to obtain I(u,v).
- 6. Obtain a symmetric filter function in the frequency domain, H(u,v) of dimension PxQ. This should also be centered at P/2 Q/2! If the filter is made in spatial domain do the same procedure as the image.
- 7. Filtering in spatial domain is done by convolving, which is the same as multiplying in the frequency domain. Filter the image by forming the product G(x,y) = I(u,v)H(u,v)(elementwise product).
- 8. To obtain the filtered image in spatial domain we do the *inverse* discrete Fourier transform (IDFT), we now have g(x,y).
- shift the image back to original position by $g(x,y)(-1)^{x+y}$ and take the real part/absolute value.
- 9. Slice out the image by taking out the top left part of the image with size MxN. We now have the filtered image $\hat{i}(x,y)$.

When zero padding it does not matter how we apply them around the image, what matters is how many zeros. When using the discrete Fourier transforms we run into an periodicity problem. This will in effect make our signal/image repeat every $1/(2\Delta t)$ in frequency domain (we use 1D for simplicity) where Δt is the sampling rate in time. As mentioned earlier convolution in time is multiplication in frequency domain. When convolving we shift our filter back and flip it 180°. In equation(17) we have the continues definition of the convolution.

$$x(t)\bigstar h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)\,d\tau \tag{17}$$

or the DFT in equation(18),

$$x[n]\bigstar h[n] = \sum_{m=0}^{M-1} x(m)h(n-m) \tag{18}$$

but since we work with the discrete version we now have this sequence in periodic intervals, this means that when we perform the convolution we do a convolution with its periodicities. Although we

can multiply them in the frequency domain we have to perform the inverse to get the image, which will in turn do the circular convolution which will yield a wrap around error. If we zero pad the signal/image in spatial/time domain we increase the distance to its periodicity such that the convolution is finished before it arrives at the periodicity generated from the DFT.

3.4 4

Since both salt & pepper noise and gaussian noise has high frequency using a gaussian low pass (see equation(11))would help removing a lot of the noise as shown in figure(15) and figure(16).

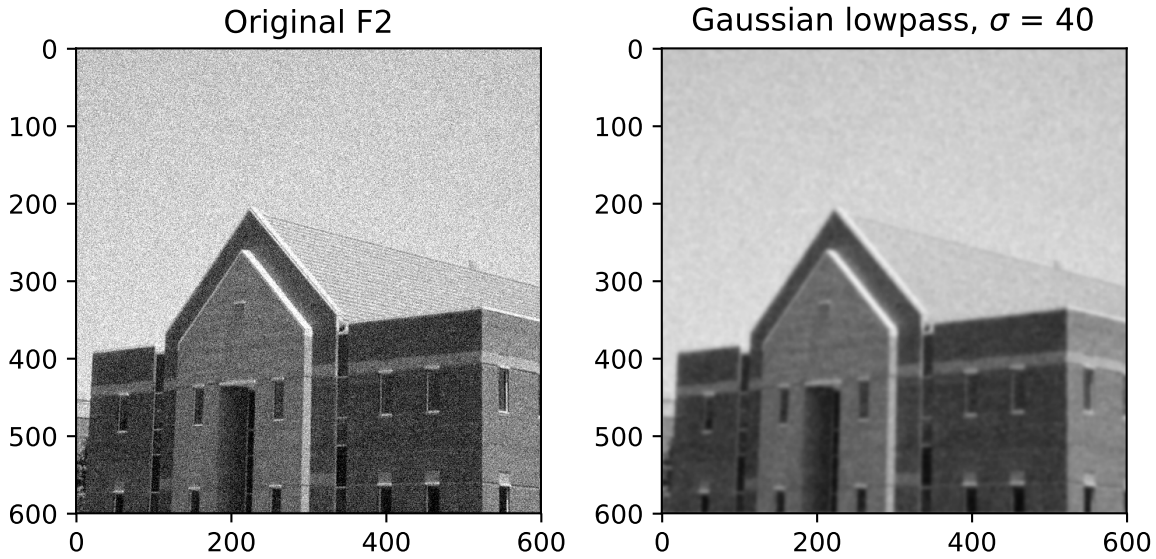

Figure 15: Original image to the left and frequency representation on the left. We can observe the two conjugated impulse representing the periodic component in the image.

Although we removed some of the noise we now have very smooth borders on the building, we could use a Butterworth filter instead of the gaussian to try and tweak with the right parameters to get sharper edges. The salt and pepper noise were easier to remove in spatial domain with the median filter as it kept the image sharp and clean. So in this case it might be more practical to remove the noise in spatial domain. For this task we just used the same gaussian lowpass filter for both images.

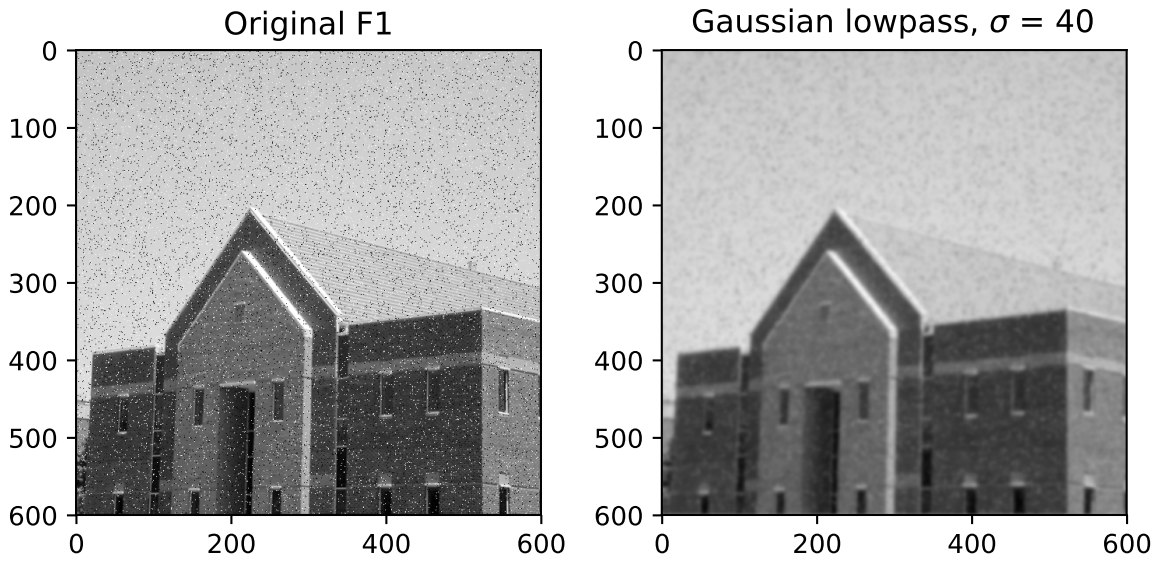

Figure 16: Original image to the left and frequency representation on the left. We can observe the two conjugated impulse representing the periodic component in the image.

In figure(17) we see the frequency domain representation of the gaussian lowpass. Here we see that around the center we have values larger then zero, and outside we have zero. This will suppress all the high frequencies and let the low frequency pass through.

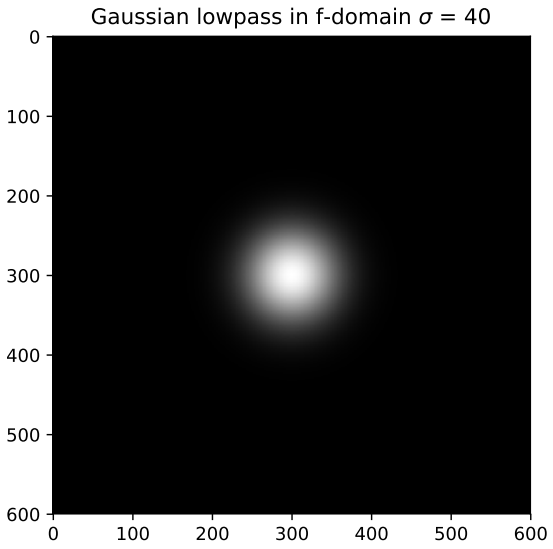

Figure 17: Frequency domain representation of the gaussian lowpass filter with cutoff = 40.

3.5 5

The noise in F3 and F4 are the same noise except one is shifted by 90°, by analyzing F4 in the frequency domain, see figure(11) we can see that there are some conjugated impulses. These impulses are shifted to ≈ +- 20 from the center, so here we have a period of about 20 pixels between the amplitudes of the noise. If we inspect the spatial domain we can also see that the lines are separated by ≈ 20 pixels between the

amplitudes so in F3 and F4 we have a frequency of 0.05.

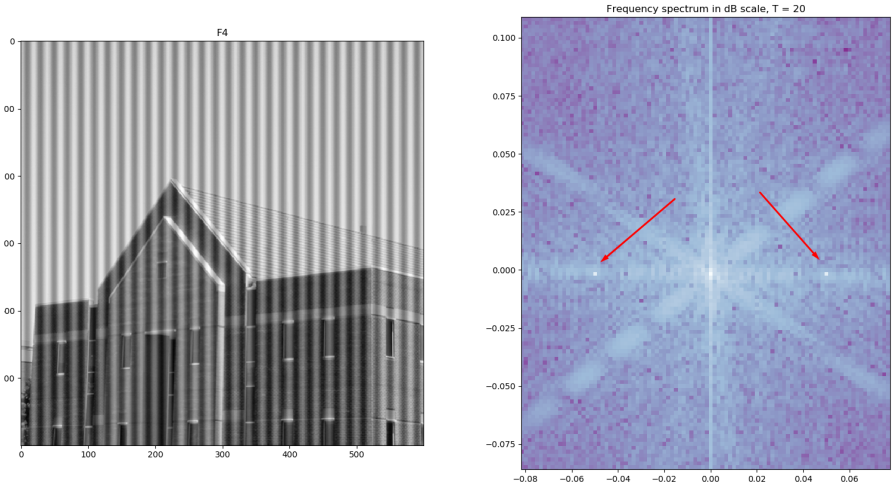

Figure 18: Original image to the left and frequency representation on the left. We can observe the two conjugated impulse representing the periodic component in the image. The impulses are at $f = -0.05$ and $f = 0.05$.

3.6 6

We can clearly see in the frequency domain of the image F5 the impulses of the periodic noise of the image in figure(19). It looks like we have 8 periodic components 4 horizontal and 4 vertical.

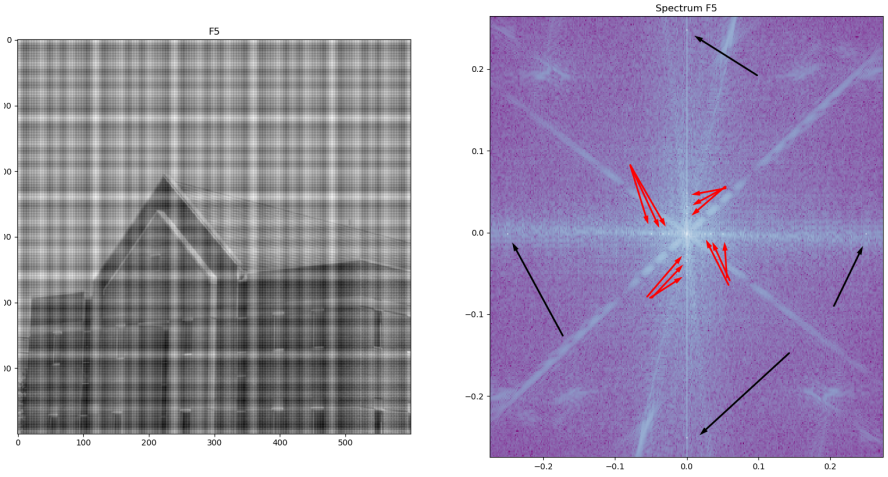

Figure 19: Left; Original image F5, Right; decibel frequency domain. The black arrows show 2 periodicities with high frequency

The frequencies are, $\approx vertical\{0.25, 0.05, 0.03, 0.02\}, horisontal\{0.25, 0.05, 0.03, 0.2\}$.

3.7 7

For filtering the images in this task we used the Butterworth notch reject filter, defined in equation(19),

$$H_{NR}(u,v) = \prod_{k=1}^{3} \left[\frac{1}{1 + [D_{0k}/D_k(u,v)]^n} \right] \left[\frac{1}{1 + [D_{0k}/D_{-k}(u,v)]^n} \right] \tag{19}$$

where D_{0k} is the cutoff frequency for each notch, $D_k(u,v)$ is the distance to the impulse to place the notch, these are symmetric so we have a $D_{-k}(u,v)$ term as well, and n is the order of Butterworth.

Removing the noise in all the images below was done with this filter (with cutoff 1 and order $n = 3$), on each of the impulses. Shown in figure(21) we have the frequency domain representation of the image F3. The result of using this notch filter on F3 is shown in figure(20).

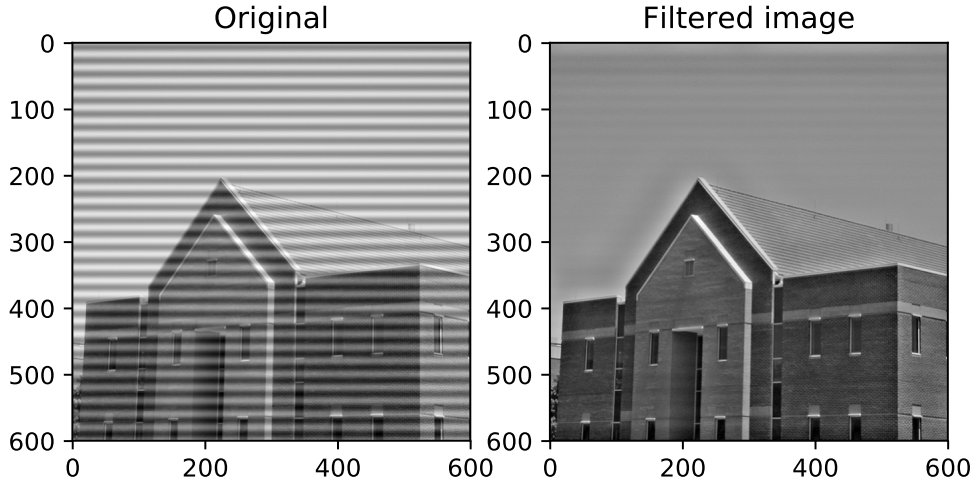

Figure 20: Left; Original image, Right; Filtered image, here we managed to remove almost all the noise. For removing the noise we used equation(19) with $D_0 = 1$ and $n = 3$. We have also sharpened using equation(21) with $k = 1$ and in our high pass Butterworth $n = 5$, $D_0 = 8$. Here we might need to reconsider the order of our filter as we have some ringing effect in the image. This is caused by have wobbles in our filter.

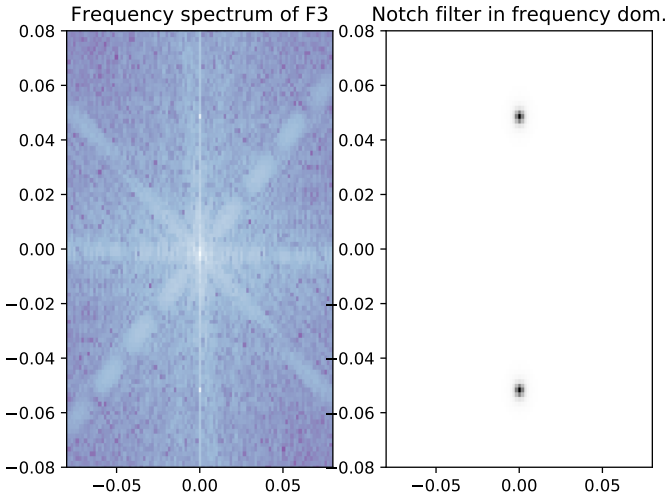

Figure 21: Left; frequency representation of image F5.png, Right; frequency representation of the Butterworth notch filter. Here we have vertical impulses since the periodic noise is in y-direction.

After applying notch rejection and sharpening we transformed the intensities to 0-255 using equa-

tion(20).

$$g(x,y) = \frac{image - min\{image\}}{max\{image - min\{image\}\}}(L-1) \tag{20}$$

where $L = 256$ in this case.

The sharpening is done by the high-frequency-emphasis filtering in equation(21),

$$g(x,y) = \mathscr{F}^{-1}\{[1 + kH_{HP}(u,v)]F(u,v)\} \tag{21}$$

Here $\mathscr{F}^{-1}$ is the inverse Fourier transform, k gives control over the proportion of high frequencies. When we applied this we used $k = 1$ and g(x,y) is the sharpened image, and F(u,v) is the frequency domain of the noise filtered image.

As the high pass filter we used the Butterworth high pass filter given by equation(22 ,

$$H_{HP} = \frac{1}{1 + [D_0/D(u,v))]^{2n}} \tag{22}$$

Here $D(u,v)$ is the distance from the center, D_0 is the cutoff frequency and n is the order of the filter, the higher this order is the closer to ideal filter. The closer the filter is to the ideal, the more ringing effect we will have.

Image F4.png is almost the same noise, but shifted by 90°, so here we flip our previous notch filter and apply and then sharpen with equation(21), the result is shown in figure(22).

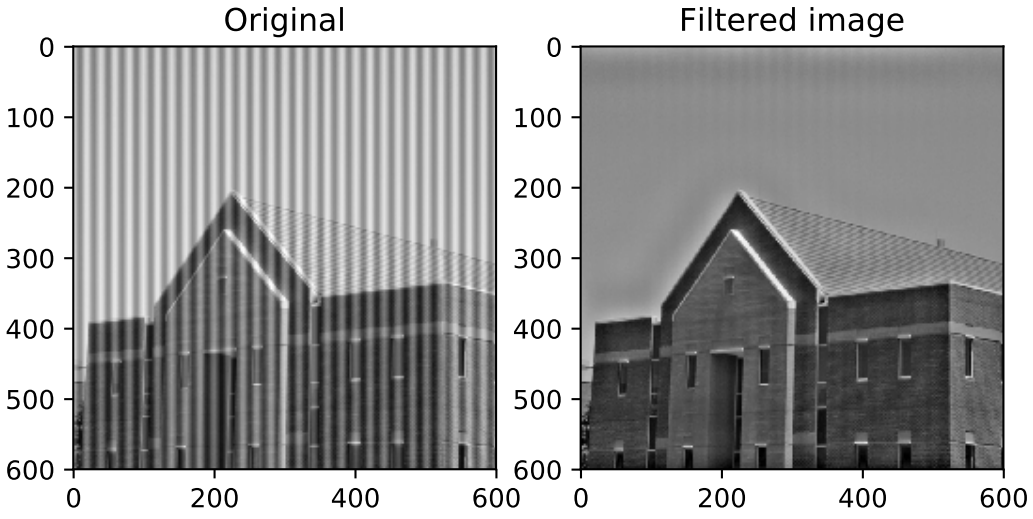

Figure 22: Left; Original image, Right; Filtered image, here we managed to remove almost all the noise. For removing the noise we used equation(19) with $D_0 = 1$ and $n = 3$. We have also sharpened using equation(21) with $k = 1$ and in our high pass Butterworth $n = 5, D_0 = 8$.Here we might need to reconsider the order of our filter as we have some ringing effect in the image. This is caused by have wobbles in our filter.

Removing the noise in image F5 was done by using a Butterworth notch filter (with cutoff 1 and order $n = 3$), on each of the 8 impulses shown in figure(25). As we can see in figure(24) we have removed almost all the noise, we have some ripple effects in some areas, most likely because we used notches which were to idealized, so here we should have tried other parameters to tune it better.

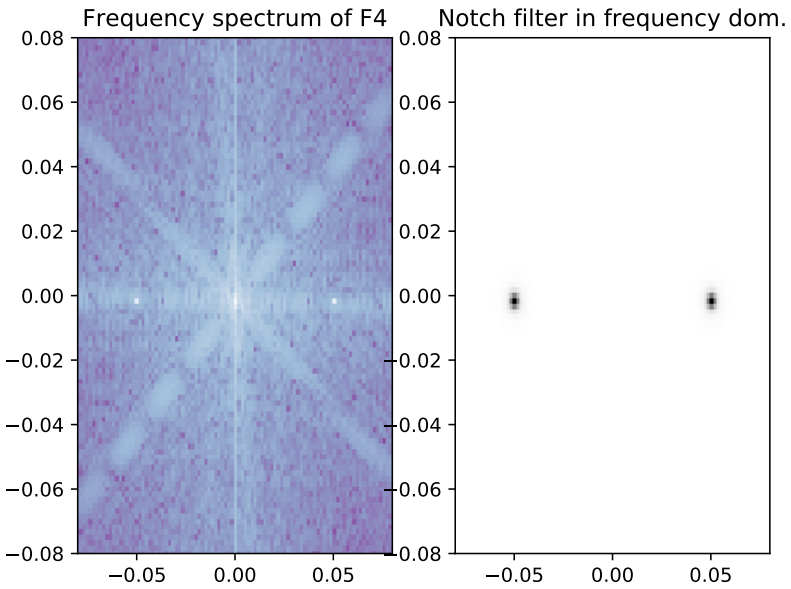

Figure 23: Left; frequency representation of image F4.png, Right; frequency representation of the Butterworth notch filter. Here the impulses are in horizontal direction since the noise is periodic in x-direction.

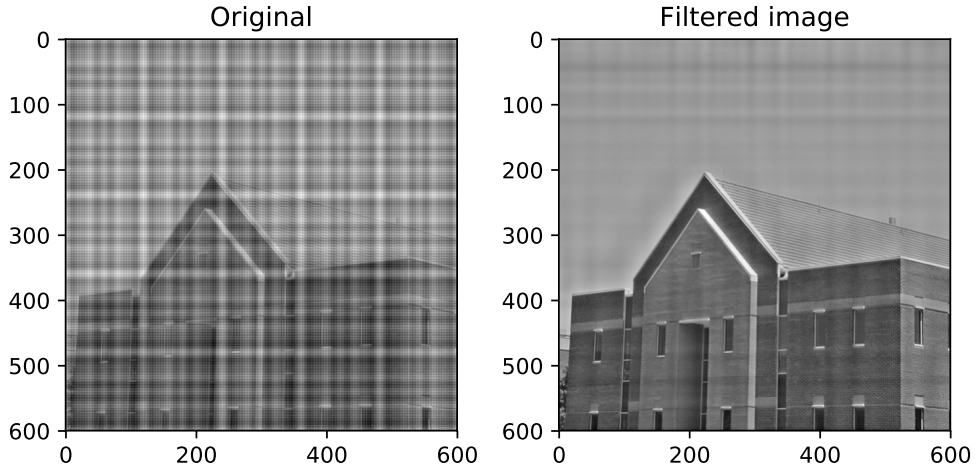

Figure 24: Left; Original image, Right; Filtered image, here we managed to remove almost all the noise. For removing the noise we used equation(19) with $D_0 = 1$ and $n = 3$. We have also sharpened using equation(21) with $k = 1$ and in our high pass Butterworth $n = 5$, $D_0 = 8$.Here we might need to reconsider the order of our filter as we have some ringing effect in the image. This is caused by have wobbles in our filter.

After removing the noise with the notch filter, we made a edge mask using Butterworth with cutoff 7 and order $n = 5$.

In figure(25) we see the dB scale frequency spectrum and the Butterworth notch rejection filter applied. Her we can see the black dots applied right on top of the impulses generated from the periodic noise.

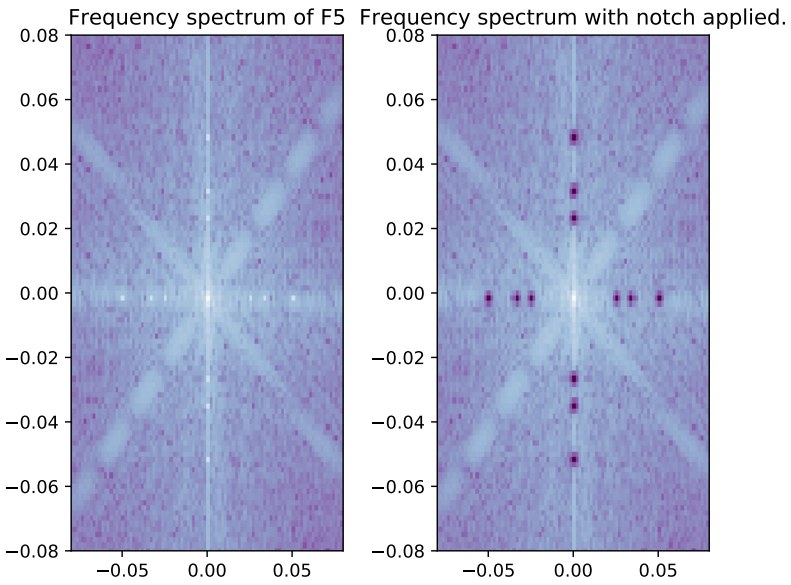

Figure 25: Left; frequency representation of image F5.png, Right; frequency representation with the Butterworth notch reject filter applied.

4 Part D

4.1 1

The degradation space is not space invariant as the degradation has a radial dependency, this is the case because the degradation is caused by rotational motion, where its center is still and the further out in radial direction yield stronger circular degradation. If we would slice out some part of the image and find the degradation function in that area and then generalize it for the whole image we would not necessary get a good result.

4.2 2

Since this image has rotational motion blur the degradation space is variant, we therefore would want to transform the image to polar form so that we have constant change in one direction. In this image we know that we have a rotation of $\theta = \frac{\pi}{4}$ since the image has been taken with 3 hours exposure time. Since a full rotation is 24 hours we will have rotated $3/24$ of a full cycle which is $1/8$ so $\theta = \frac{2\pi}{8} = \frac{\pi}{4}$ and multiplying this with the amount of pixels in x-direction gives us the total displacement in x, displacement $= (1/8) * M$. Changing coordinate system to polar is shown in figure(26), for this we used the cv2.linearPolar library in python.

The polar coordinate system is now spatial invariant and we can now start filtering.

The filter we will be using is the wiener filter as shown in equation(23),

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right] G(u,v) \tag{23}$$

Here $\hat{F}(u,v)$ is the filtered image, $H(u,v)$ is the degradation function which in this case is the motion blur in x-direction given by equation(24), K is the inverse signal to noise ratio which is constant. This

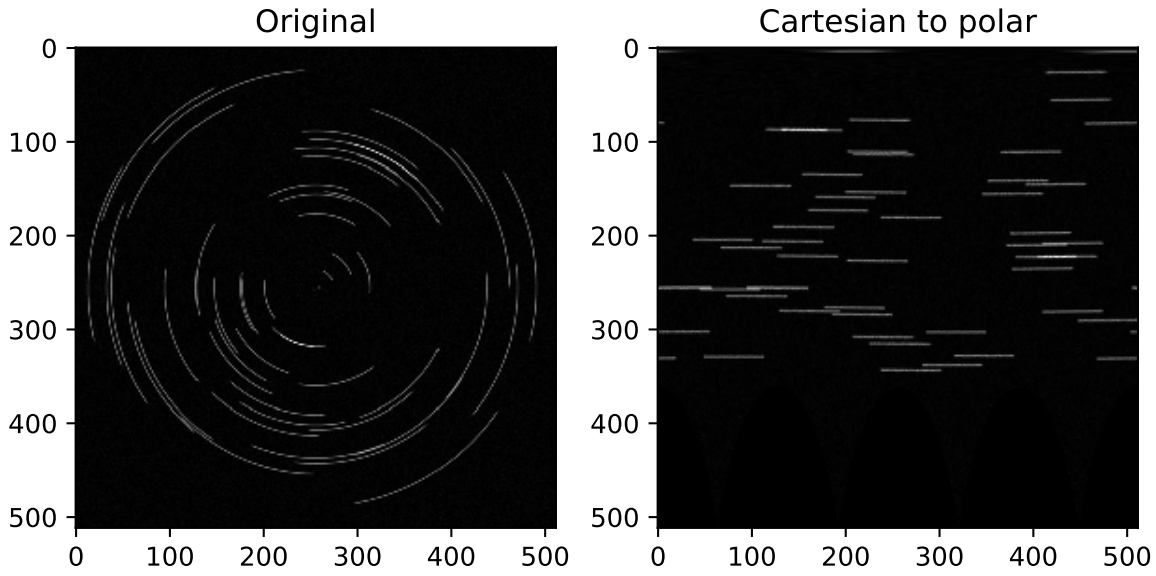

Figure 26: The original image transformed to polar coordinates. We can observe that all the stars have same length in θ direction.

parameter is usually unknown so we have to tune it until we get something that works best. Signal to noise ratio is the ratio of how strong our signal is compared to the noise. If this ratio is low we have strong signal, but if it is very high we then have lost a lot of the signal information to noise. In our case we have a strong signal so we have a low value K, $K = 0.001$. To use this filter right we have to assume that the noise is constant, which we can assume here because it looks like our noise is *white noise* which has the same power over all frequencies.

$$H(u,v) = \begin{cases} 1, if\ u = 0 \\ \frac{1}{a}\frac{\sin(\pi a u/M)}{\sin(\pi u/M)}, if\ u \neq 0 \end{cases} \tag{24}$$

In equation(24) a is the number of pixels the image has been displaced.

4.3 3

After we applied the Wiener filter in polar coordinates we got the results shown in figure(29) and figure(30). At the first attempt we used u and v to go from $0 - M$, but then we got some complications because the function we have in equation(24) is as shown in figure(27), this function is symmetric around zero. The Dirichlet kernel will flip if you traverse to far in either direction, so by going from $-M/2$ to $M/2$ we fixed the problem we had. The K value was chosen because we had a lot of noise in our image (high noise to signal ratio), but we had to try different values as the real value was not known. In figure(31) we have not squared the insensity values as we did in figure(30) to show how squaring suppresses some of the noise around the stars.

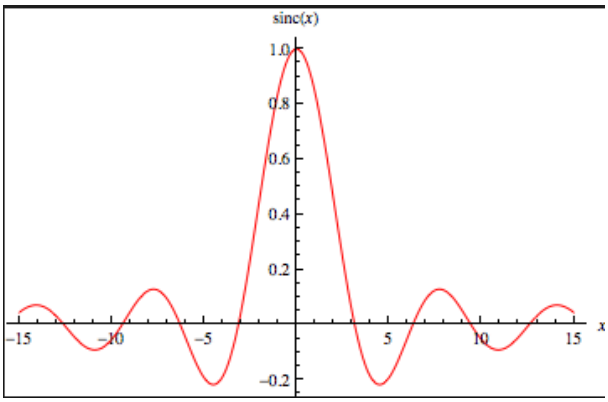

Figure 27: A Dirichlet kernel.

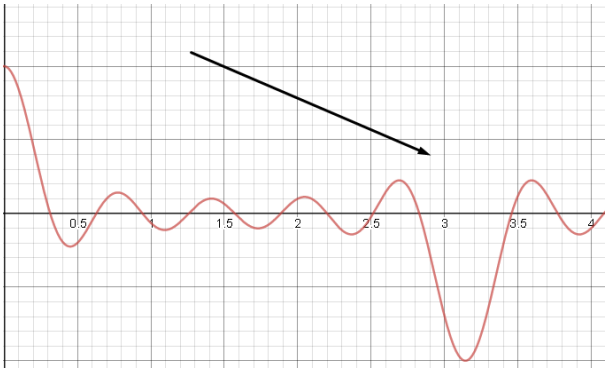

Figure 28: Dirichlet kernel which gets flipped as we traverse the axis.

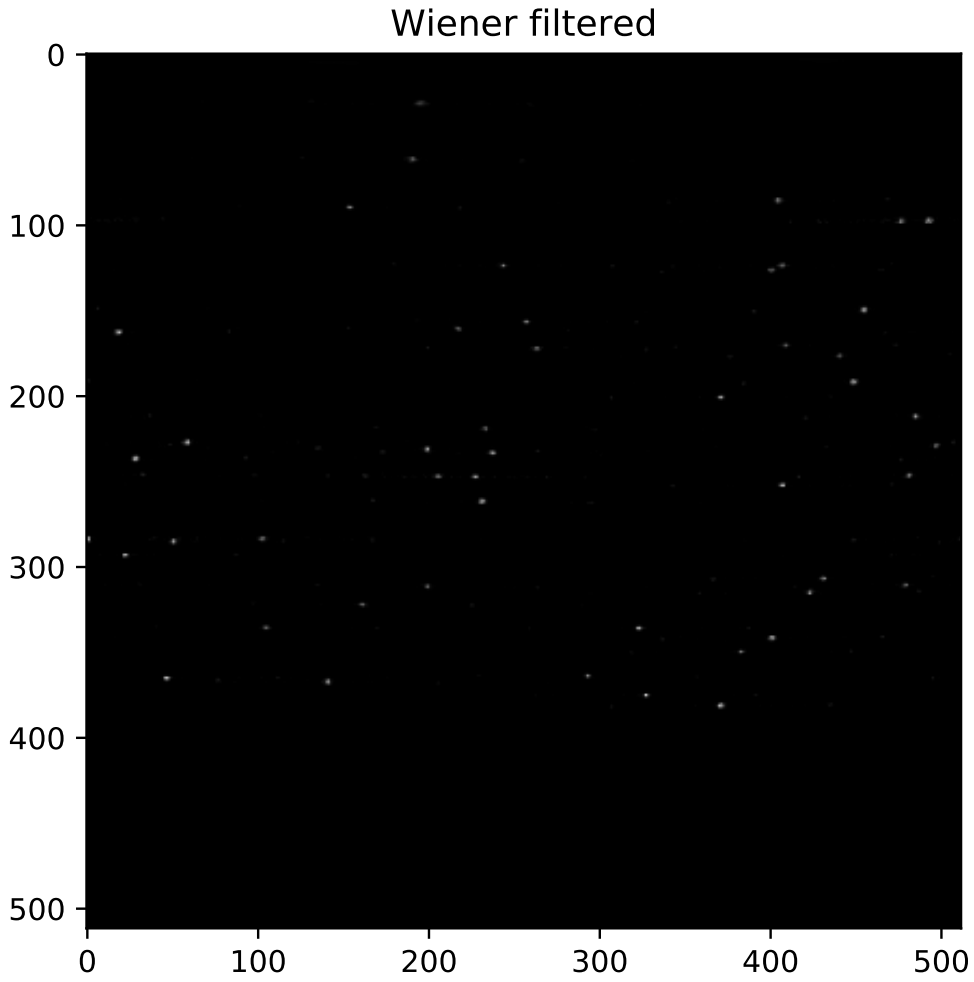

Figure 29: Applied Wiener filter with $K = 0.001$ and $a = (1/8) \cdot M$, where M is the number of pixels in x-direction, in this case $M = 512$. The values have been squared so that we decrease the intensity of the noise(the noise has low power).

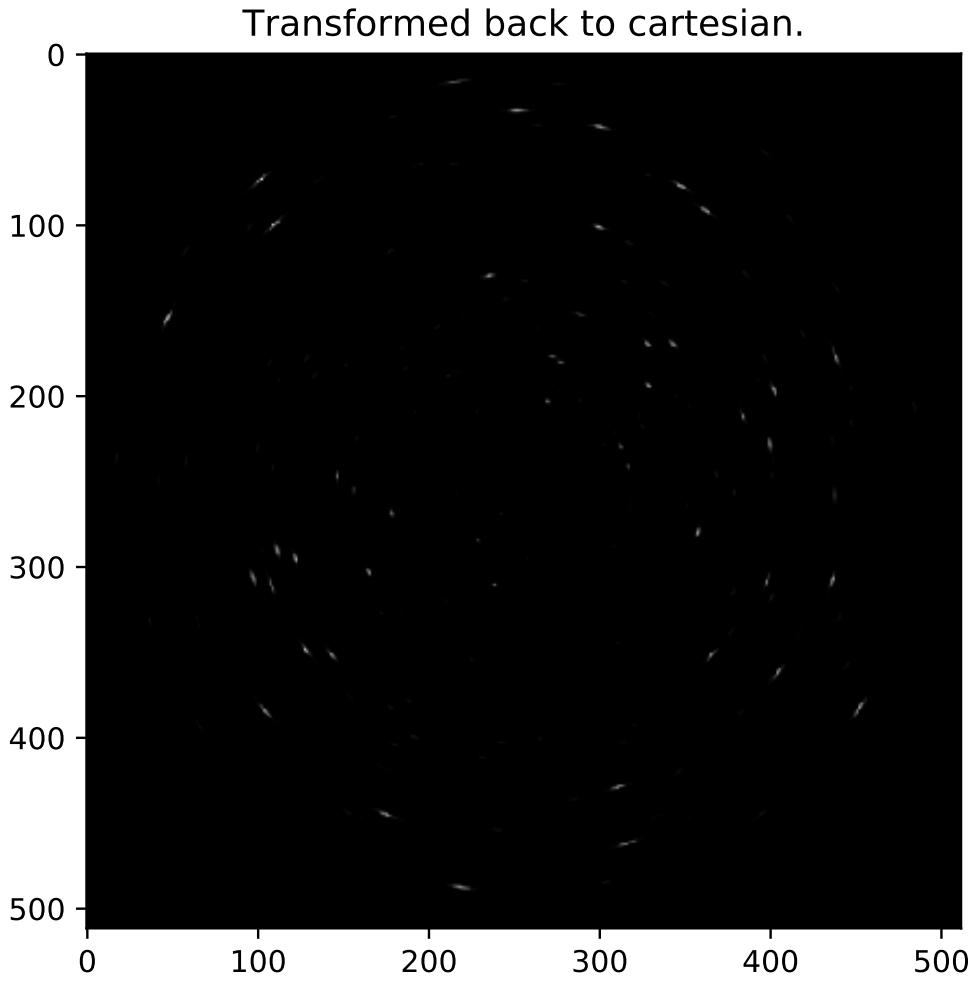

Figure 30: Changed figure(29) back to cartesian coordinates, we can now observe the night sky without the pesky earth ruining it for us.

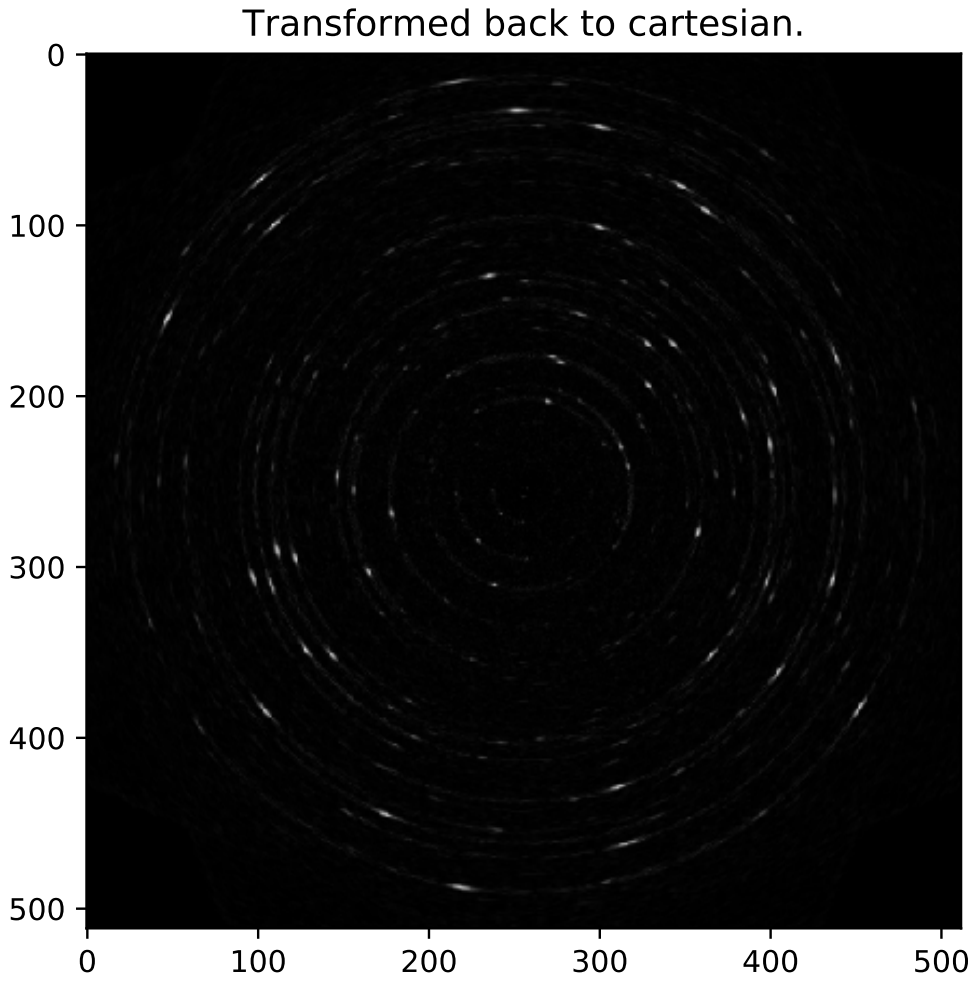

Figure 31: If we do not square the image we get this, here we can see the lower intensity noise around the stars.

5 Appendix

For more in-depth code see extra files.

```
        import numpy as np
        import matplotlib.pyplot as plt
        import os
        from PIL import Image
        import cv2
        from scipy.signal import convolve2d

```

Figure 32: Imports

```
        def gaussian_lp(image, sigma):
        """Gaussian low pass filter, sigma defines the radius around the centered
    frequency(cutoff)"""
        row, col = image.shape
        H = np.zeros((row, col))
        for y in range(row):
            for x in range(col):
                D = np.sqrt((y-int(row/2))**2 + (x-int(col/2))**2)
                H[y,x] = np.exp(-D**2/(2*sigma**2))
        X = np.fft.fftshift(np.fft.fft2(image))
        Y = np.fft.fftshift(X*H)
        y = np.fft.ifft2(Y)
        return np.abs(y), H

# Fourier, shift absolute value and decibel
Y2 = 10*np.log10(np.abs(np.fft.fftshift(np.fft.fft2(F4))))

# Get frequencies sampling 1
N2 , M2 = Y2.shape
Y2freqN = np.fft.fftshift(np.fft.fftfreq(N2, 1))
Y2freqM = np.fft.fftshift(np.fft.fftfreq(M2, 1))
# Fourier transform image
spectrum = np.fft.fftshift(np.fft.fft2(F5))
# dB scale
spectrum1 = 10*np.log10(np.abs(spectrum))

# get shape
N , M = spectrum1.shape
# Find right frequency, sampling rate 1
HfreqN = np.fft.fftshift(np.fft.fftfreq(N, 1))
HfreqM = np.fft.fftshift(np.fft.fftfreq(M, 1))

```

Figure 33: TASK C.4-C.5-C.6

```
        def resize_image(image, newN, newM):
            """Resizing by taking only the second pixel (50% resizing)"""
            resize = np.zeros((newM, newN))
            row, col = resize.shape
            try:
                for j in range(col):
                    for i in range(row):
                        resize[i, j] = image[i*2, j*2]
            except:
                pass

            return resize

        row, col = image.shape
        # Resize image 50%
        imagerez = resize_image(image, int(col*0.5), int(row*0.5))


        def smoothing(image, boxsize = 3):
            """
            Avarage smoothing of 2D array
            boxsize -> size of boxkernal filter
            returns filtered 2D array
            """
            boxkernal = np.ones((boxsize, boxsize))/(boxsize**2)
            result = convolve2d(image, boxkernal, mode = 'same')
            return result.astype('uint8')


        def lapsharp(image, maskret = False):
            """
            img -> 2D array
            maskret = True -> returns result and mask
            maskret = False -> returns result
            """
            #padded_image = np.pad(img, (1, 1), mode = 'symmetric')
            # lap is linear therefore;
            # lap f(x,y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x,
    y)...
            #---------------------
            c = -1 # Depends on kernel
            # make zero kernal
            lapmask = np.zeros((3, 3))

            # add values to kernel
            lapmask[0,0] = 1
            lapmask[0,1] = 1
            lapmask[0,2] = 1

            lapmask[1,0] = 1
            lapmask[1,1] = -8
            lapmask[1,2] = 1

            lapmask[2,0] = 1
            lapmask[2,1] = 1
            lapmask[2,2] = 1
            #---------------------
            mask = convolve2d(image, lapmask, mode = 'same')
            result = image + c*mask

            # Map values to 0-255
            g1 = image - np.min(image)
            g = g1/np.max(g1) *255
            g = g.astype('uint8')

            if maskret == True:
                return g, mask
            else:
                return g.astype('uint8')


    def gaussian_hp(image, sigma):
        """Gaussian high pass filter sigma defines the radius around the centered
    frequency"""
        row, col = image.shape
        H = np.zeros((row, col))
```

```
        # Make histogram
        fig, ax = plt.subplots(1,2)
        ax[0].hist(F1.flatten(), bins = 256)
        ax[0].set_title('Histogram F1')

        ax[1].hist(F2.flatten(), bins = 256)
        ax[1].set_title('Histogram F2')

        plt.savefig('noisehist.pdf')
        plt.tight_layout()
        plt.show()


        def medianfilter(image, boxsize = 3):
            """
            Uses median filter, convolution in spatial domain\n
            returns processed image.
            """
            image_padded = np.pad(image, (boxsize,boxsize) , mode = 'symmetric')
            result = np.zeros(image.shape)
            rows, cols = image.shape
            # Traverse the image and apply filter.
            for row in range(rows):
                for col in range(cols):
                    result[row, col] = np.median(image_padded[row:row + boxsize,col:col
    + boxsize].flatten())
            return result


        def adaptive_filter(image, boxsize = 3):
            """
            Adptive filtering, found variance of noise by find variance of a slice in
    the top image.
            Uses local var and local mean to set pixel value.
            """
            # Pad image
            image_padded = np.pad(image, (boxsize, boxsize) , mode = 'symmetric')
            result = np.zeros(image.shape)
            rows, cols = image_padded.shape
            # Get histogram of strip in image
            hist, bins = np.histogram(F2[0:100,:].flatten(),256)
            # Intensities
            r = np.array([x for x in range(256)])
            # Get size
            rows, cols = image.shape
            # Get probabilites
            prob = hist/(rows*cols)
            # Get mean
            m = np.sum(r*prob)
            # Get variance estimate of noise
            variance_noise = np.sum((r-m)**2*prob)

            # Traverse the image and apply filter.
            for row in range(rows):
                for col in range(cols):
                    local_var = np.var(image_padded[row:row + boxsize,col:col + boxsize
    ])
                    local_mean = np.mean(image_padded[row:row + boxsize,col:col +
    boxsize])
                    kernel_placement = image_padded[row:row + boxsize,col:col + boxsize]
                    current_val = image_padded[row,col]
                    if variance_noise > local_var:
                        result[row, col] = current_val - 1*(current_val - local_mean)

                    else:
                        result[row, col] = current_val - (variance_noise/local_var)*(
    current_val - local_mean)

            return result.astype('uint8')

            def alpha_trimmed_mean(image, d, boxsize = 3):
            """
            Alpha trimmed mean filter.
            """
            # Pad image
            image_padded = np.pad(image, (boxsize, boxsize) , mode = 'symmetric')
```

```
        def butter_notch_filter(image, sigma, notches, n):
            """Butterworth notch reject, sigma defines the radius around the notch
    frequency
            and n defines the order.(how close to ideal you want)"""
            row, col = image.shape
            res = np.ones((row, col))
            for h in notches:
                H = np.zeros((row, col))
                for y in range(row):
                    for x in range(col):
                        Dp = np.sqrt((y-int(row/2)-h[1])**2 + (x-int(col/2)-h[0])**2)
                        Dn = np.sqrt((y-int(row/2)+h[1])**2 + (x-int(col/2)+h[0])**2)
                        if Dp > 0 and Dn > 0:
                            H[y,x] = (1/(1+ (sigma/Dp)**n))*(1/(1+ (sigma/Dn)**n))
                        else:
                            H[y,x] = 0
                res *= H
            H = res
            X = np.fft.fftshift(np.fft.fft2(image))
            Y = np.fft.fftshift(X*H)
            y = np.fft.ifft2(Y)
            return np.abs(y), H

def butterworth_hp_sharp(image, sigma, n, k):
        """Butterworth high pass filter, sigma defines the radius around the centered
    frequency
        and n defines the order.(how close to ideal you want)"""
        row, col = image.shape
        H = np.zeros((row, col))
        for y in range(row):
            for x in range(col):
                D = np.sqrt((y-int(row/2))**2 + (x-int(col/2))**2)
                if D > 0:
                    H[y,x] = 1/(1+ (sigma/D)**(2*n))
                else:
                    H[y,x] = 0
        X = np.fft.fftshift(np.fft.fft2(image))
        # sharp image (High frequency emphasis filtering)
        Y = (1+k*H)*X
        Y = np.fft.fftshift(Y)
        y = np.fft.ifft2(Y)
        return np.abs(y)


```

Figure 36: TASK C7.1

```
        # Centered frequency, since notch filter have origo in center we need this to
    subtract.
        mid = 300
        notches = np.array([[330-mid, 0]])

        # Use filters
        y, H = butter_notch_filter(F4, 1, notches, 3)
        y = butterworth_hp_sharp(y, 8, 5, k = 1)

        # Convert back to 255 after masking
        y = (y-np.min(y))/np.max(y-np.min(y))*255
        y = y.astype('uint8')


```

Figure 37: TASK C7.2

```
        # Centered frequency, since notch filter have origo in center we need this to
    subtract.
        mid = 300
        # Notches to reject
        notches = np.array([[450-mid, 0], [330-mid, 0], [320-mid, 0], [315-mid, 0], [0,
    450-mid], [0, 330-mid], [0, 320-mid], [0, 315-mid] ])

        # Use filters
        y, H = butter_notch_filter(F5, 1, notches, 3)
        y = butterworth_hp_sharp(y, 8, 5, k = 1)

```

Figure 38: TASK C7.3

```
        # Transpose
        image = plt.imread(file)
        # Transform to 0-255
        image = (image-np.min(image))/np.max(image-np.min(image))*255
        image = image.astype('uint8')
        rows, cols = image.shape
        radius = np.sqrt((rows/2-25)**2 +(cols/2-25)**2)
        polar_image = cv2.linearPolar(image, (int(rows/2), int(cols/2)), radius, (cv2.
    WARP_FILL_OUTLIERS+cv2.INTER_LINEAR))

        # Flip image with transpose
        polar_image = polar_image.astype('uint8').T


        def wiener_filter(image, a, K):
            """Wiener filter, a = displacement, K = Inverse signal to noise ratio. """
            rows, cols = image.shape
            H = np.zeros((rows, cols))
            for v in range(int(-rows/2), int(rows/2)):
                for u in range(int(-cols/2), int(cols/2)):
                    if u == 0:
                        H[v+int(rows/2),u+int(cols/2)] = 1
                    else:
                        H[v+int(rows/2),u+int(cols/2)] = (1/a)*(np.sin(np.pi*u*a/cols)/(
    np.sin(np.pi*u/cols)))

            # H.H^*
            H_abs = (H*np.conj(H))
            X = np.fft.fftshift(np.fft.fft2(image))
            top = H_abs
            bot = H*(H_abs + K)
            G = (top/bot)*X
            y = np.fft.ifft2(G)

            # Return image
            return np.abs(y)


        # Apply filter and then taking it to the second power to remove some noise.
        im = wiener_filter(polar_image, a = (1/8)*cols, K =0.001)**2

        # Return image to cartesian coordinates.
        im2 = cv2.linearPolar(im.T, (int(rows/2), int(cols/2)), radius, (cv2.
    WARP_INVERSE_MAP + cv2.INTER_LINEAR))


```

Figure 39: TASK D

6 References

Richard E. Woods Rafael C. Gonzales. *Digital Image Processing.* Pearson, fourth edition, 2018.