Image Analysis

Lab 2: 2-D Fourier Transform

Student: TRAN Gia Quoc Bao, ASI

Professor: HENG UY Chhayarith

Date: 14 February 2020

Introduction: The objectives of this lab work are to observe some properties of the 2-D Discrete Fourier Transform (DFT) and use it for filtering.

Preparation before the lab:

The traditional (1 dimension) DFT of x[n], n = 1, 2, 3, …, N-1 that we used for discrete signals:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}$$

To **extend** mathematically the concept of monodimensional Fourier transform into 2 dimensions, as the 2 directions are linearly independent of each other, we naturally have the following formula for an MxN image:

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} F(x,y) e^{-j2\pi\left(\frac{ux}{N}+\frac{vy}{M}\right)}$$

This is a map between the spatial domain (like the time domain for signals) to the frequency domain. So for each pair (u, v), we sweep through the image and calculate the corresponding sum, as we did with the signal. The slowest varying component (u, v) = (0, 0) corresponds to the average intensity in the image. High and low frequencies correspond to the fast and slowly varying patterns. But for visuality, in MATLAB we will need to do some shifting after doing the 2D-FFT to place the low frequencies at the center.

Normally, what we do with a signal is first to study its DFT in order to spot the special frequencies. For example, if we find the DFT of the signal which is the sum of 2 sin waves, we will have 2 peaks (we know they are Dirac) at these frequencies. So we know these special components. Then, we can use some filters to separate them. With images, we are going to do basically the same thing:

Textures are the periodic patterns in an image, which means they correspond to different frequencies. The Fourier Spectrum is well suited to describe the directionality of textures because it allows us to see these special frequencies. We need to find them first before we can do anything to them.

It is useful to **filter** images, as sometimes we do not want some values of frequencies. For example, there are fast varying patterns at the edges of an image, or they can be caused by noises. After using the DFT (possibly after some pre-processing), we use low-pass filters to remove these. Then we can switch back to the spatial domain to obtain the processed image. Low-pass filtering is smoothing while high-pass filtering is sharpening the image.

I have this comparison:

Signals ↔ Images

Time (t) ↔ Space (x, y)

1D DFT from the time to the frequency domain ↔ 2D DFT from the spatial to the frequency domain

Then:

Signal processing ↔ Image processing

We know that the DFT of an edge (square pulse) is a sinc of which the bandwidth is inversely proportional to the pulse's width. So for large edges, we have small-bandwidth sinc functions who lie 90 degrees to its corresponding edge.

For example, with the Lena image, we have a line in the DFT corresponding to the pattern from the woman's hat to her hair, her face and hair.

Figure 1. The Lena image and its corresponding DFT

Computer session:

I. Fourier analysis of images:

1. Familiarizing with the Fourier domain:

First, to save time and code lines, I made a MATLAB function that will be used on each of the images:

```
function [RealPart, ImagPart, Magnitude, Phase] = ImageDFT(I, nameOfImage)
% Calculate the DFT and shift it to the center
ImageDFT = fftshift(fft2(I));
% Calculate the real part, imaginary part, magnitude, and phase
RealPart = real(ImageDFT);
ImagPart = imag(ImageDFT);
Magnitude = log(1 + abs(ImageDFT)); % this one is in log scale
Phase = angle(ImageDFT);
% Plot all the the real part, imaginary part, maginitude, and phase
figure();
subplot(221);
imshow(RealPart, []);
title("The real part of the DFT of the " +nameOfImage);
```

Figure 2. The function to calculate and display an image's DFT

When we give it an image and its name, this will calculate and shift the 2-D DFT, then display the real part, imaginary part, magnitude (in log scale) and phase of the image in 4 different windows. It also displays the magnitude separately for us to see better, as this is quite complex. First, I considered the case where **w is divisible by 128**. Let's take w = 32.

This is the result obtained of the 2-D DFT:

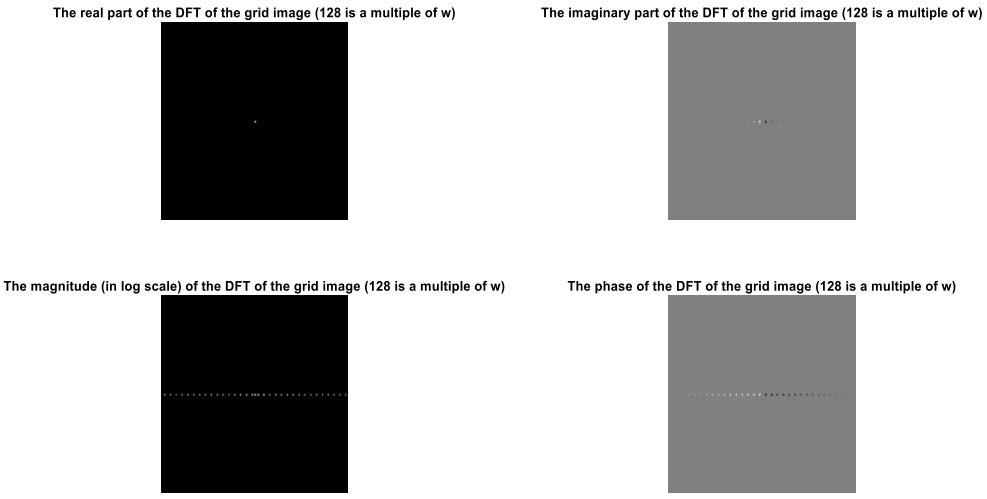

Figure 3. The DFT of the grid image where w is divisible by 128

What I got for the magnitude was some dot in the horizontal direction. This is because in the image if we go in this direction, we will obtain a periodic square pulse, of which the DFT is the sinc function. The reason why these are dot is that the image does not change in the vertical direction, so no pattern in this direction.

Then, I consider **w not divisible by 128**, for example, in this case, w = 30:

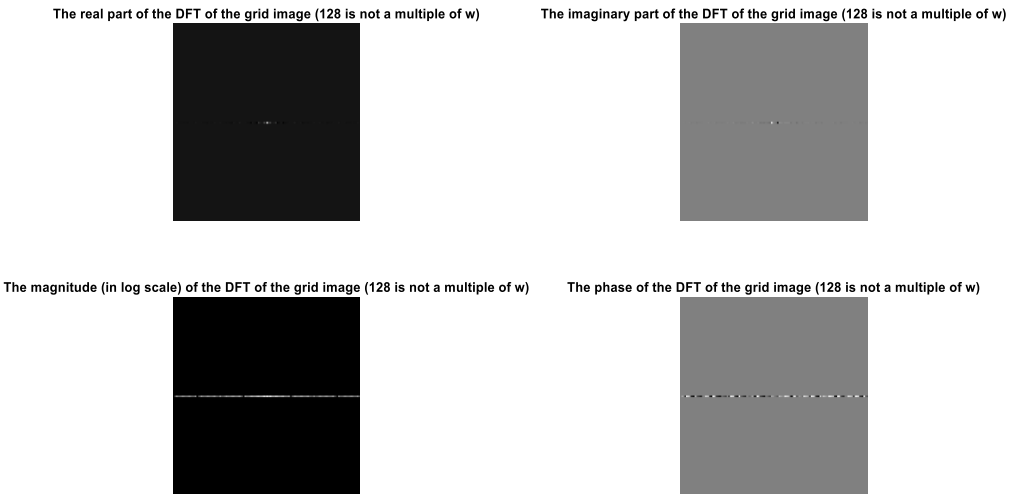

Figure 4. The DFT of the grid image where w is not divisible by 128

I note that this time the last stripe is cut, which affects the frequency response, so the dots are longer in length. The following image is for comparison:

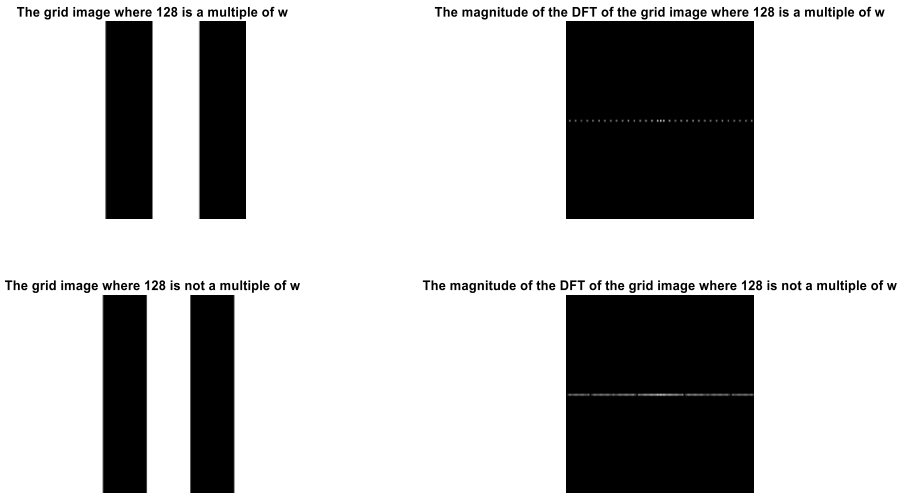

Figure 5. Comparison of the 2 cases of the grid image

I then created 2 discs of different sizes, $r_1 = 25$ and $r_2 = 50$. For the first one I got this result:

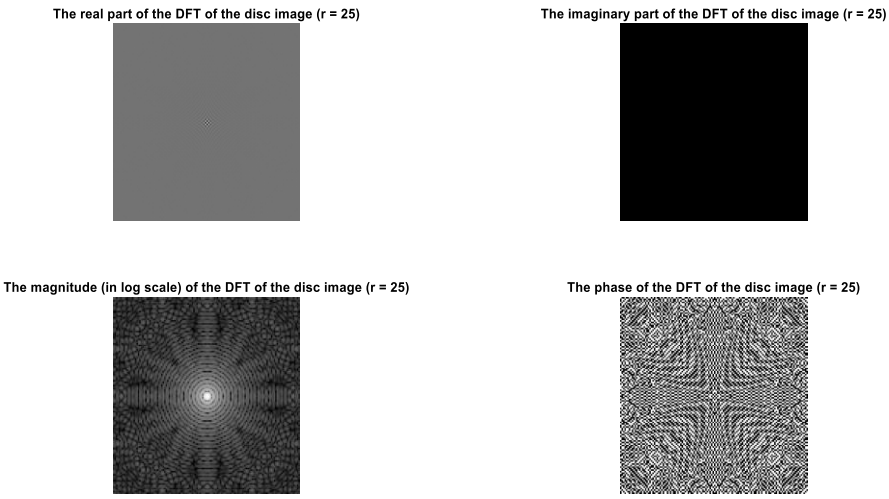

Figure 6. The DFT of the disc image r = 25

This time, as we have a multidirectional pattern (the same in all directions), the DFT magnitude is circular. For the bigger circle we also have symmetry about the center point:

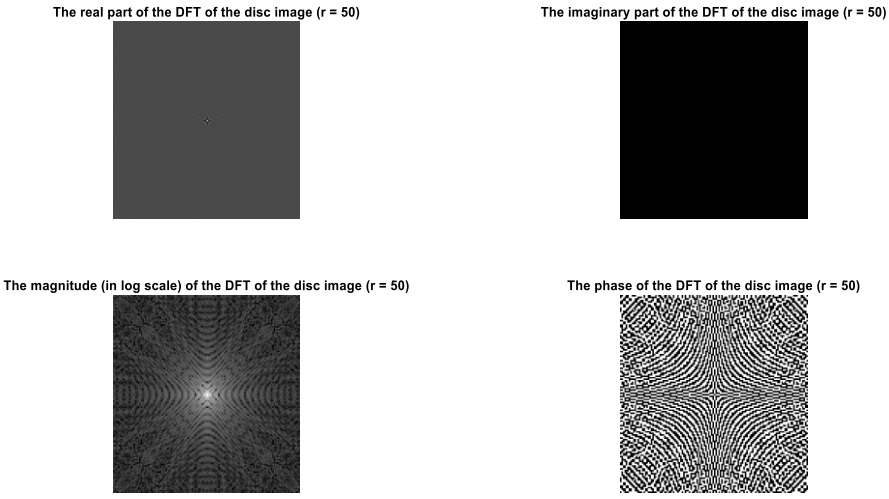

Figure 7. The DFT of the disc image r = 50

To compare, we see that if the circle is bigger, the image is less similar to a pattern, so the rings that represent the frequency patterns fade away. If the circle is too large that the image is almost white, we have no patterns at all.

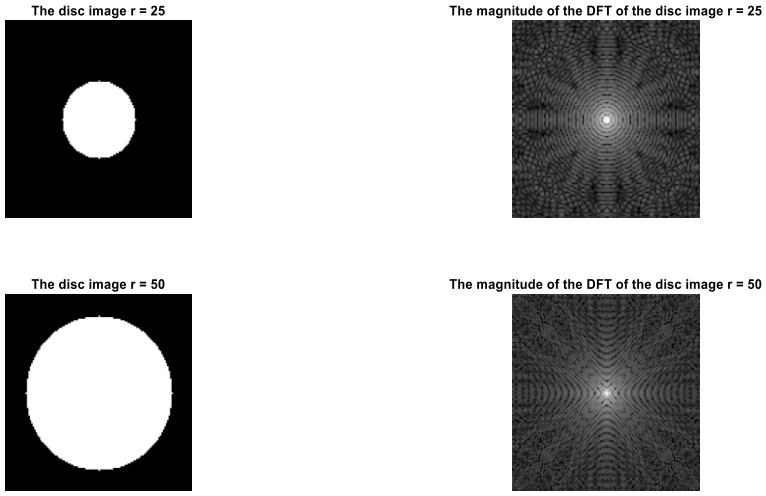

Figure 8. Comparison of the 2 cases of the disc image

I used the function on 3 different real images: the cameraman, the Lena, and the city image. Because displaying all the 4 elements is too space-consuming, here I only show the magnitude:

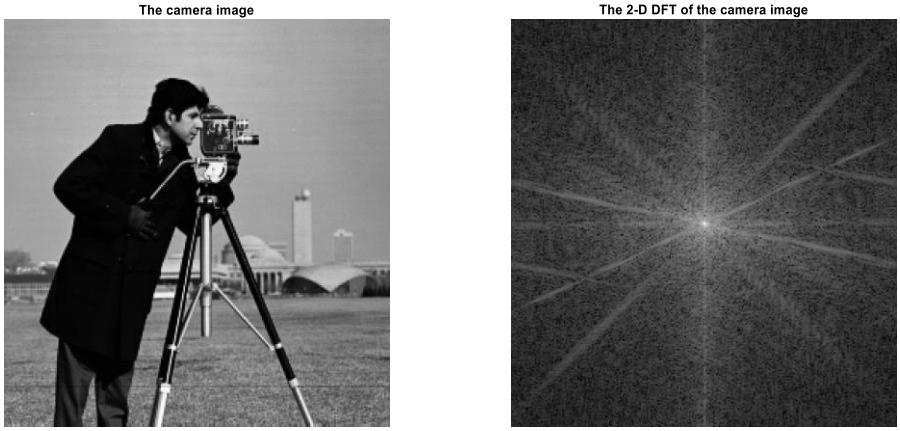

Figure 9. The Cameraman image and its corresponding DFT magnitude

This result is explained by the patterns in the image: the "\" pattern caused by the man's body and the camera's left leg, the "/" by the camera's right leg, the "|" by the ground and the buildings behind, and the "---" by the camera's middle leg.

Figure 10. The Lena image and its corresponding DFT magnitude

For the Lena image, as I explained in the preparation part, we have some patterns with my woman's hat and hair.

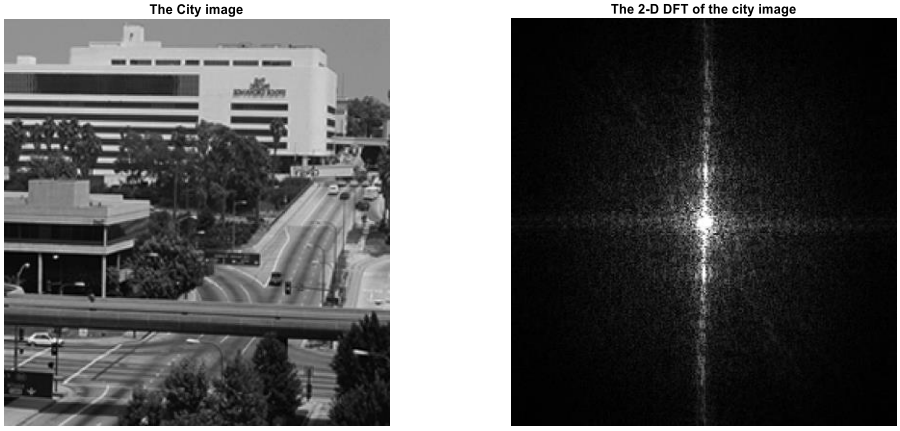

Figure 11. The City image and its corresponding DFT magnitude

The vertical and horizontal patterns cause by the roads and the buildings are the reason for the 2 lines in the DFT.

Lastly, I applied the function on 2 textures.

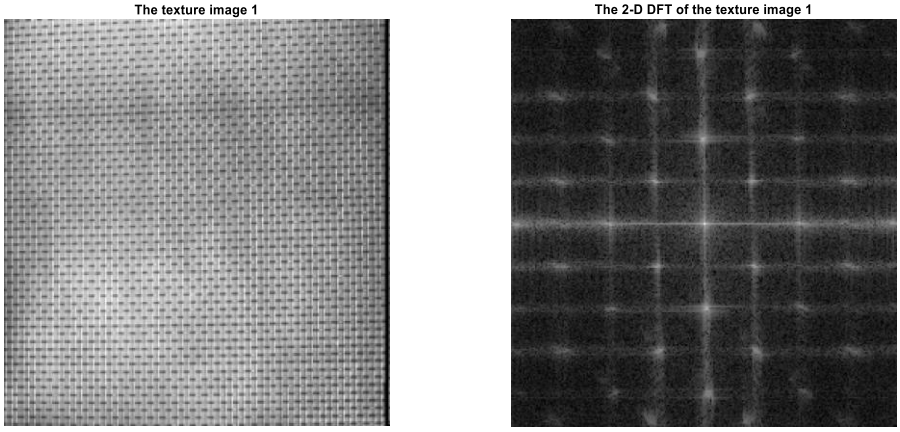

Figure 12. The texture image 1 and its corresponding DFT magnitude

For this, we have a lot of dots and stripes in both directions. If we look closely, we will see that the frequencies are in multiples, like f, 2f, 3f,… (like harmonic), so we have a lot of lines intersecting each other in the DFT.

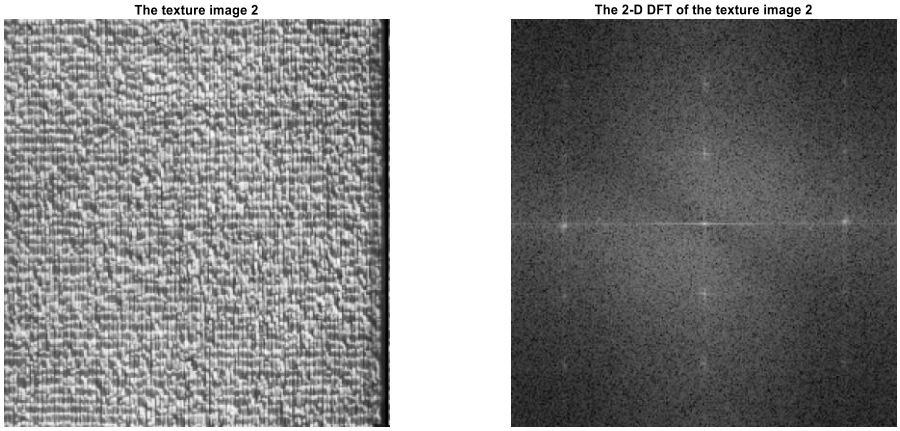

Figure 13. The texture image 2 and its corresponding DFT magnitude

For this, the pattern is less clear, and the frequency in the horizontal direction is smaller than that in the vertical direction, so the vertical lines are further away from the center than the horizontal ones (the middle is where we have the low frequencies, and the boundaries are for high frequencies).

2. Amplitude and phase information:

Shifting an image is like rotating it about an imaginary axis. So the objects in the left will go to the right, and the ones in the right edge will appear at the left edge.

Like above, I made another MATLAB function that will shift the image using the “circshift” command. The inputs are the image, the number of elements by which the image is shifted, and the image’s name.

```
function [Magnitude, Phase, MagnitudeShifted, PhaseShifted] = ImageDFTWithShift(I, shiftNum, nameOfImage)
% Calculate the DFT and shift it to the center
ImageDFT = fftshift(fft2(I));
% Calculate the magnitude, and phase
Magnitude = log(1 + abs(ImageDFT)); % this one is in log scale
Phase = angle(ImageDFT);
% Shift the image, then calculate the DFT and shift it to the center
ImageDFTShifted = fftshift(fft2(circshift(I, shiftNum)));
% Calculate the magnitude, and phase of the shifted image
MagnitudeShifted = log(1 + abs(ImageDFTShifted)); % this one is in log scale
PhaseShifted = angle(ImageDFTShifted);
% Plot the maginitude, and phase to compare
figure();
```

Figure 14. The function to shift an image and compare the DFT

Then I applied this on the Lena image by shifting it by 100 pixels to obtain the following results:

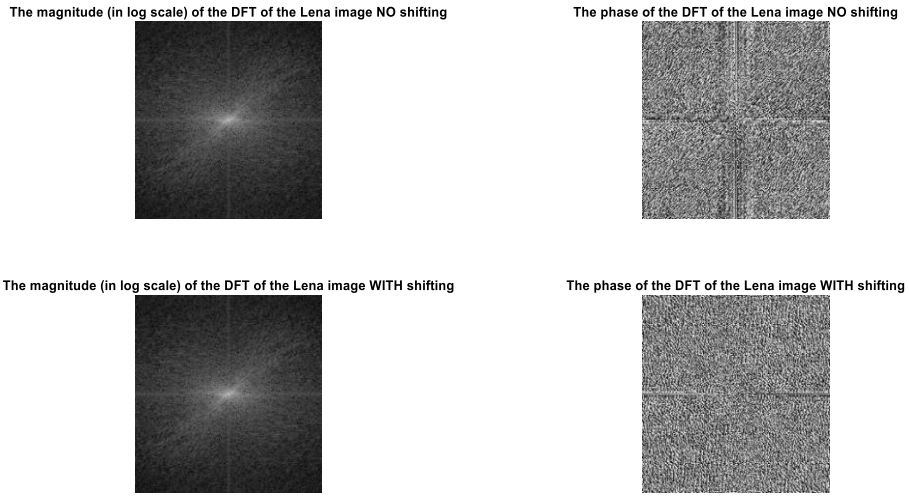

Figure 15. Comparison of the Lena image before and after shifting

We have **no clear changes in the magnitude** because the patterns (contours) keep the same direction. For example, if we have a line like this “\” in the center, and we move it to the right, it will be split into “\” and “\” on 2 sides (but shorter, of course). So the direction does not change in the magnitude of the 2-D DFT we would have this “/” for both cases.

But **the phase does change**, as we changed the geometry of the image. Like, we have some discontinuities in the middle (where we shift the edges). So we had an idea about which kinds of information are stored in which part of the DFT.

I also created a function to reconstruct the original image from its DFT. It will give us the image if we give it a DFT in the form of magnitude and phase, by using the ifftshift and ifft2 respectively.

```
function [ImageDFTInverse] = ImageDFTInverse(Magnitude, Phase, nameOfImage)
% Calculate the IDFT after shifting back the low frequency parts to the edges
I = zeros(size(Magnitude));
for m = 1:size(Magnitude)
    for n = 1:size(Magnitude)
        I(m, n) = Magnitude(m, n)*cos(Phase(m, n)) + j*Magnitude(m, n)*sin(Phase(m, n));
    end
end
ImageDFTInverse = ifft2(ifftshift(I));
% Plot the image
figure();
imshow(ImageDFTInverse*255, []);
title("The " +nameOfImage+ " recovered from its DFT");
```

Figure 16. The function to calculate and display an image's inverse DFT from the magnitude and phase

First, I inputted the magnitude as original and the phase as a zero matrix to get this result:

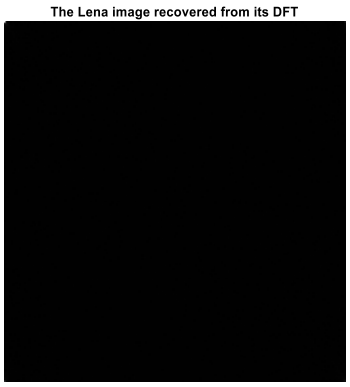

Figure 17. The Lena image recovered from the case phase = 0

If we let the phase go to 0, even if we keep the same magnitude we will get a completely dark image. Because without the phase, we have no structure, no content in the image. The information stored in the phase is the **geometrical structures**.

Then, I inputted a non zero constant magnitude and the phase as original:

Figure 18. The Lena image recovered from the case magnitude = non zero constant

If we keep the phase and let the magnitude equal to some non zero constant, we still see the image because now we have the structures it contained. We can see the boundaries which make the woman. The information stored in the magnitude is the **texture** and the **contours**.

II. Filtering in the frequency domain:

1. Filtering:

To experiment with filtering in the frequency domain, I used 3 different filters. In this domain, if we shift the DFT, the low frequencies are in the middle and the high ones are further away. The first filter was an isotropic low-pass filter, of which the function in the frequency domain is:

$$H_1(u,v) = \begin{cases} 1, & D(u,v) \leq D_{01} \\ 0, & D(u,v) > D_{01} \end{cases}$$

In which D_{01} is a cut frequency in the form of a radius that we need to decide. This filter keeps unchanged the frequencies equal to or lower than a certain value while eliminates completely the ones higher than this value.

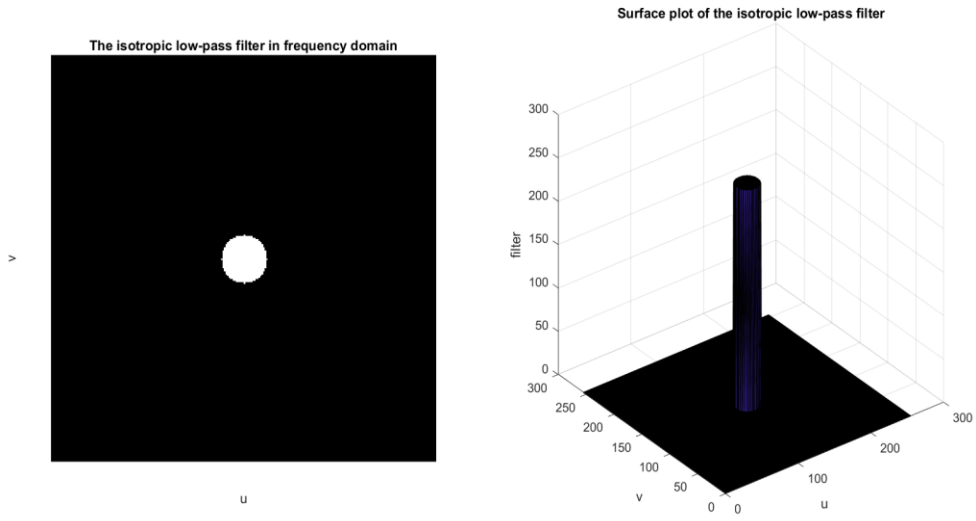

Figure 19. The isotropic low-pass filter's DFT and surface plot

I used D_{01} = 15. In the frequency domain, this corresponds to a white circle of a certain radius. When we perform filtering in the frequency domain, we need to multiply the filter which the image's DFT, because convolution in the spatial domain becomes multiplication in the frequency domain. I got this result for the Cameraman image:

Figure 20. The Cameraman image isotropically low-pass filtered

We see that the elements in the image of high frequencies, like the buildings behind and the camera's middle leg, are filtered away as we cut the high frequencies. The image is much blurrier, which means that it has been affected a lot by the filter, which means this image has a lot of patterns which high frequencies.

An isotropic high-pass filter is similar to this one: we cut completely the low frequencies and do not touch the high ones.

$$H_2(u,v) = \begin{cases} 1, & D(u,v) \geq D_{02} \\ 0, & D(u,v) < D_{02} \end{cases}$$

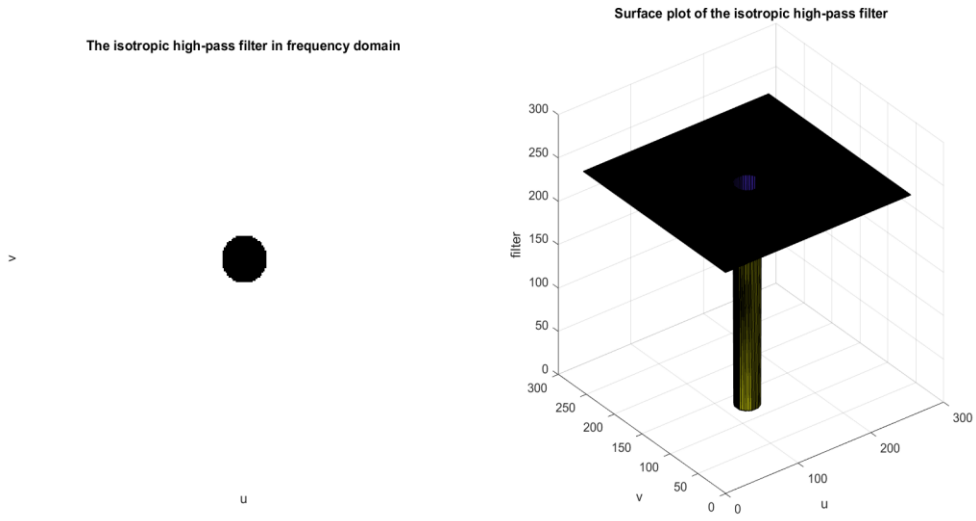

Figure 21. The isotropic high-pass filter's DFT and surface plot

I used D_{02} = 15. In the frequency domain, this corresponds to a black circle of a certain radius. The frequencies near 0 from all directions are cut completely.

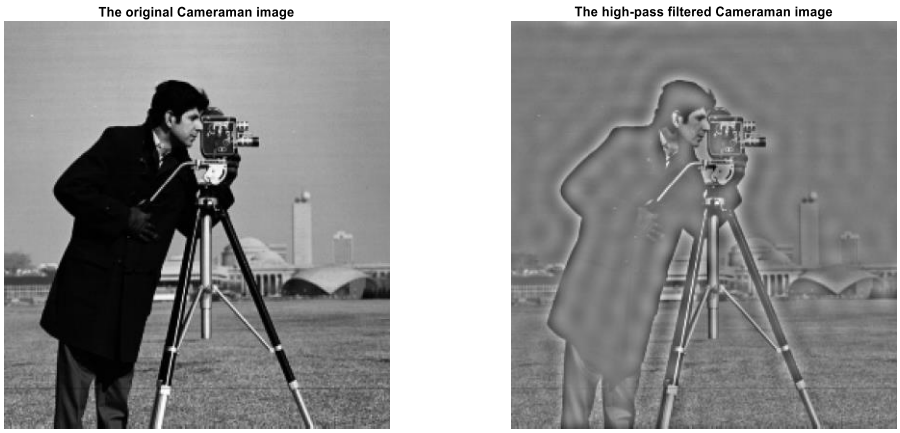

Figure 22. The Cameraman image isotropically high-pass filtered

We see that the image is not much affected, because the high frequencies corresponding to the buildings, the camera's legs,… are kept in this case.

The last filter is a Butterworth, of which the function in the frequency domain:

$$H_3(u,v) = \frac{1}{1 + \left(\frac{u^2 + v^2}{D^2}\right)^N}$$

In which D represents the cut frequency and N is the filter's order. Here I took N = 2 and D = 30.

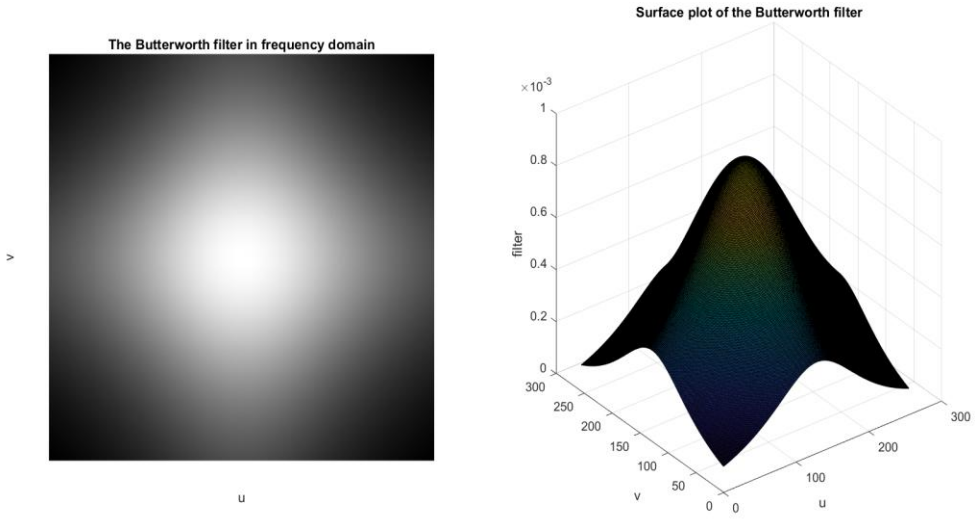

Figure 23. The Butterworth filter's DFT and surface plot

The Butterworth filter is a low-pass one, but it is a smooth function instead of a step.

Figure 24. The Cameraman image Butterworth filtered

We see that this time the image after being filtered has very few changes, as the filter is low-pass. How much the image is filtered depends on our choice of D and N. For a larger D we keep more frequencies, and for larger N the same frequency is more filtered.

2. Equivalence with convolution in spatial domain:

I made an average filter of dimension 3x3:

$$\text{filter} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

I filtered the City image in 2 different ways:

- First, take the image's and the filter's DFTs, multiply them, then inverse the result into the spatial domain. This is called filtering in the frequency domain.
- Taking the convolution between and image and the filter, using imfilter. This is called filtering in the spatial domain.

Then I compared the results obtained:

Figure 25. Comparison of the City image filtered in Frequency and Spatial domains

We see that the difference between the zones in the image is reduced because the filter took the average of the pixels. The important thing is that the filtered image in 2 cases are the same, which means that we have verified the equivalence between multiplication (filtering in the frequency domain) and convolution (filtering in the spatial domain).

I used the immse function which will calculate the average squared difference between the 2 filtered images:

immse(ICityButterworthFilteredFreq, ICityButterworthFilteredSpatial) = 0.0059

The result is not exactly zero which means there is still very little difference between the images, as the convolution is not perfect. But this is very small compared to an image of size 254x254, so the verification is still acceptable.

Conclusion: After this Lab, I understood the 2-D DFT: what it means, what information is stored in which component. I also understood that image filtering which is just like signal filtering but in space, and verified that convolution in the spatial domain is equivalent to multiplication in the frequency domain.