

Lab 3

Basic Image Processing Algorithms
Fall 2016

Lab 3: Fourier Transformation

- implementing Discrete Fourier Transformation
- deadline (if you do not finish until the end of the lab): **23:59, 05/10/2016**;
please send it to: bipa2016fall@gmail.com
with the subject: **LAB3**

Discrete Fourier Transform

- ◉ We sample one period of the Fourier transform in evenly spaced frequencies:

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

The size of the image in the spatial domain is $N_1 \times N_2$

The size of the image in the frequency domain will be the same: $N_1 \times N_2$

$$X(k_1, k_2) = X(\omega_1, \omega_2) \Big|_{\omega_1 = \frac{2\pi}{N_1} k_1, \omega_2 = \frac{2\pi}{N_2} k_2}$$

$$k_1 = 0, 1, \dots, N_1 - 1$$

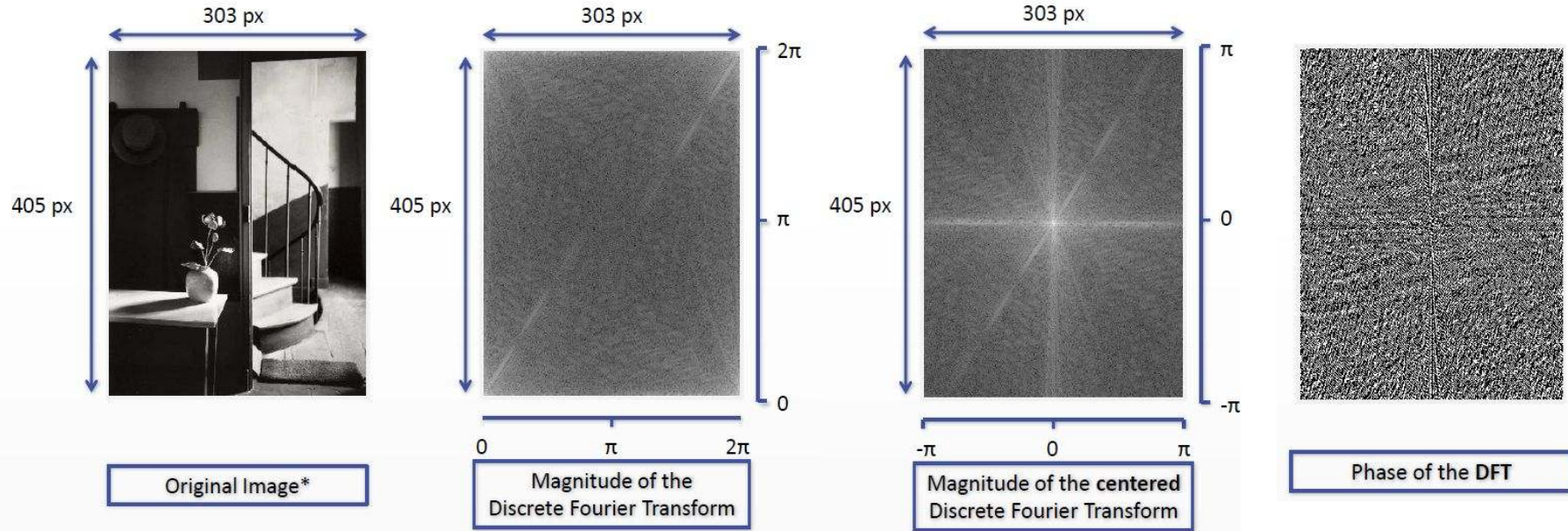
$$k_2 = 0, 1, \dots, N_2 - 1$$

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N_1} k_1 n_1} e^{-j\frac{2\pi}{N_2} k_2 n_2}$$

Only one period is kept

$$x(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) e^{j\frac{2\pi}{N_1} k_1 n_1} e^{j\frac{2\pi}{N_2} k_2 n_2}$$

Discrete Fourier Transform



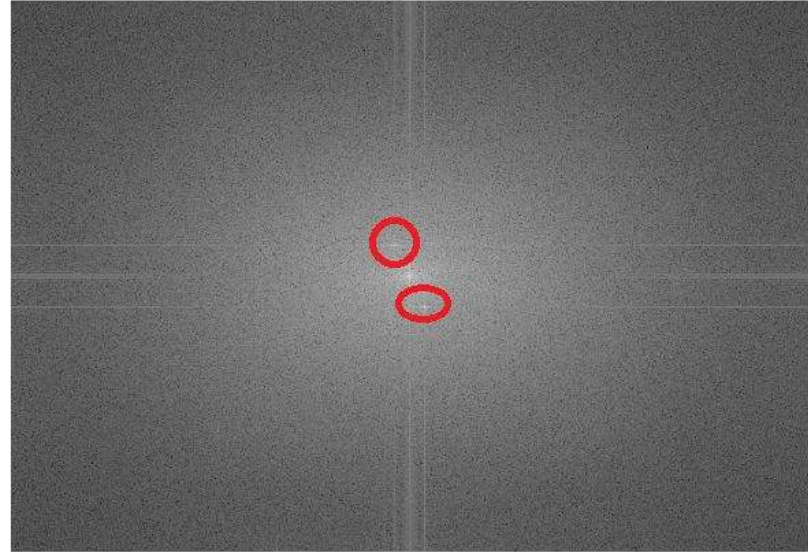
*Chez Mondrian by András Kertész (1926)

Example application I. - noise filtering

original input with noise



original magnitude of DFT



On the left: original image (Robert Capa: Lovers Parting Near Nicosia, Sicily) + some sinusoidal noise

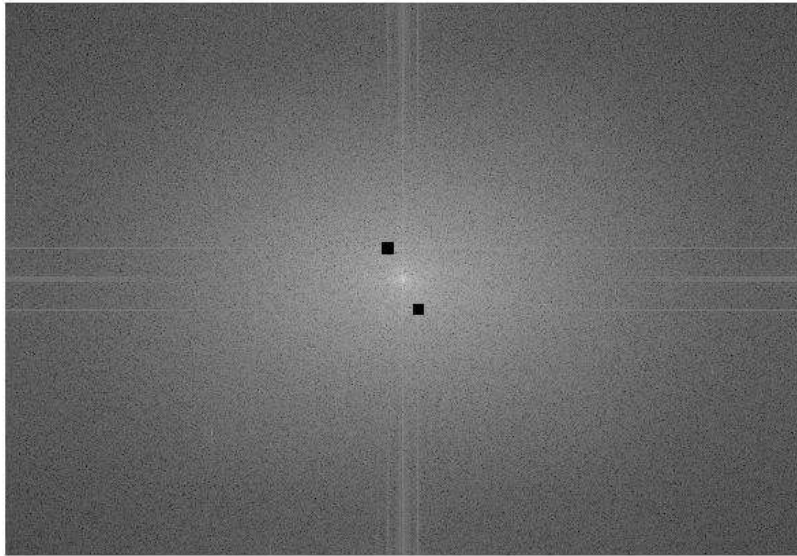
On the right: magnitude part of the frequency domain

Thanks to the regular form of the noise, we can see it as two concentrated frequency-points on the DFT-image (circled with red).

Example application I. - noise filtering

What if we hide these two, intensive-regions from the frequency-domain? (Hide:=decrease their significance, actually I set their value to complex zero, $0+0i$.) Yes, we will get rid of the noise :)

modified magnitude of DFT



filtered output



Example application I. - noise filtering

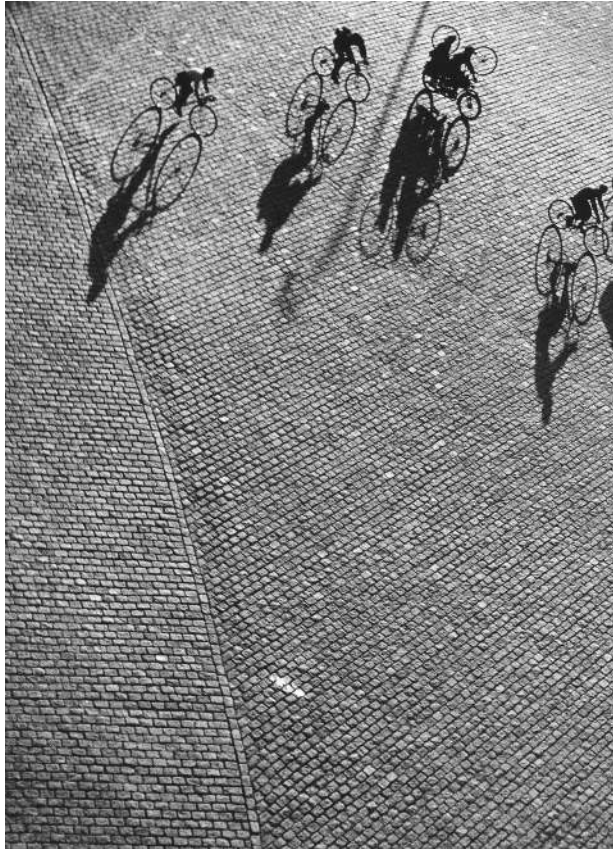
part of orig. input



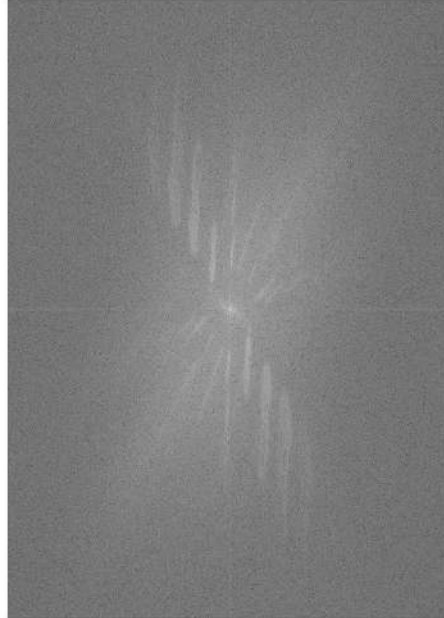
part of filt. output



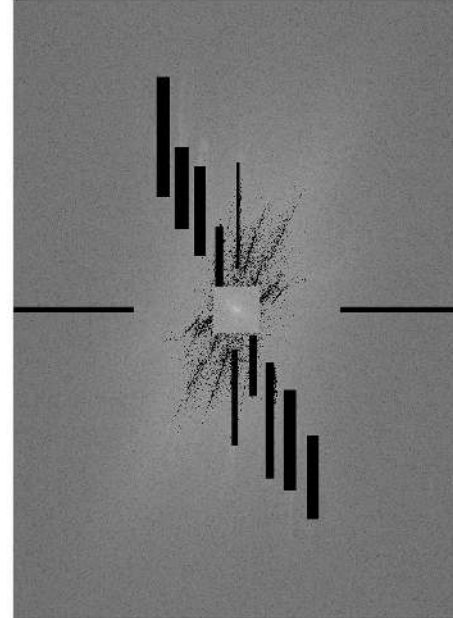
Example application II. - noise filtering



original magnitude of DFT



modified magnitude of DFT



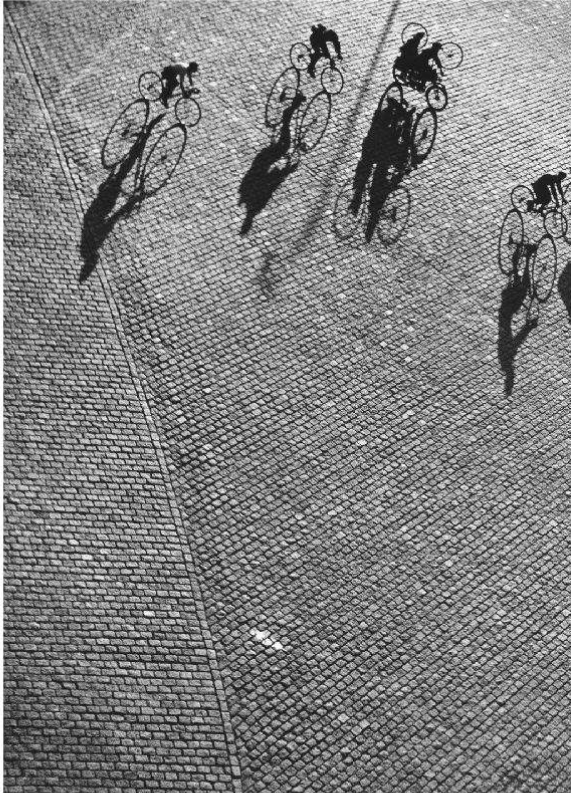
On the left: Lucien Herve: Paris Sans Quitter Ma Fenetre (Les Cyclistes)

On the center: magnitude part of the frequency domain

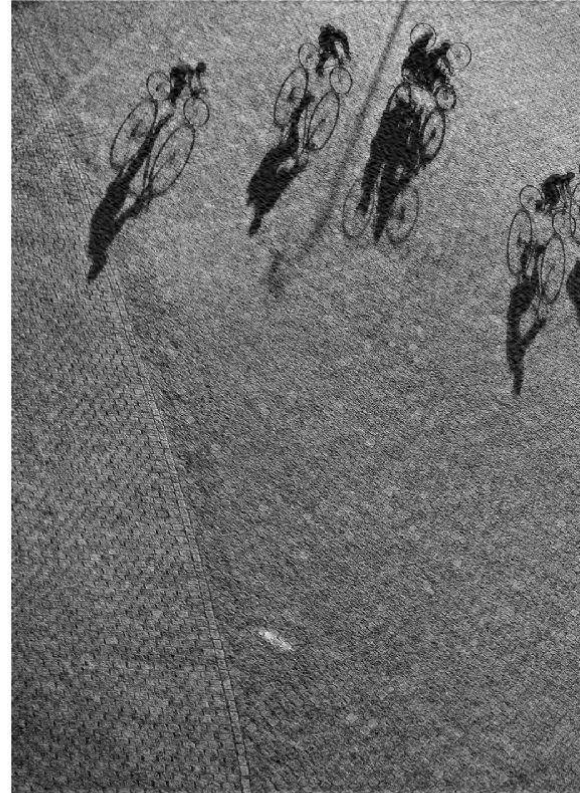
On the right: modified magnitude (again, the spec. values are replaced with complex zeros)

Example application II. - noise filtering

original input

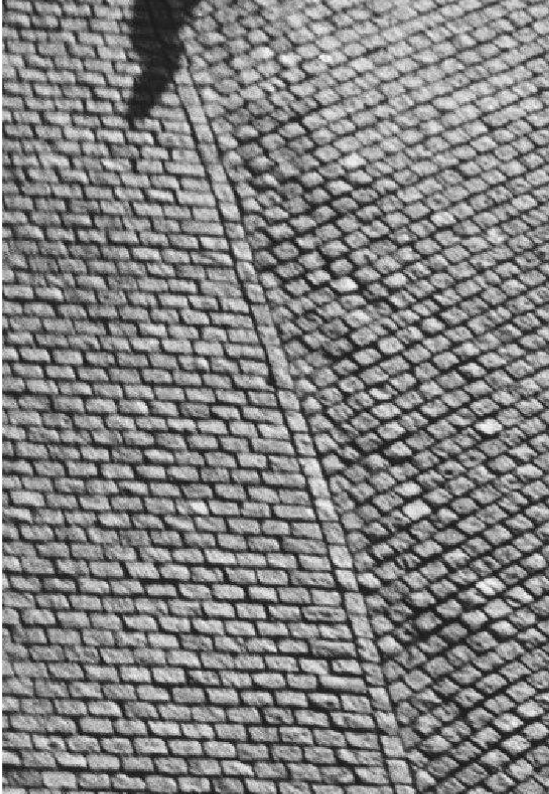


filtered output

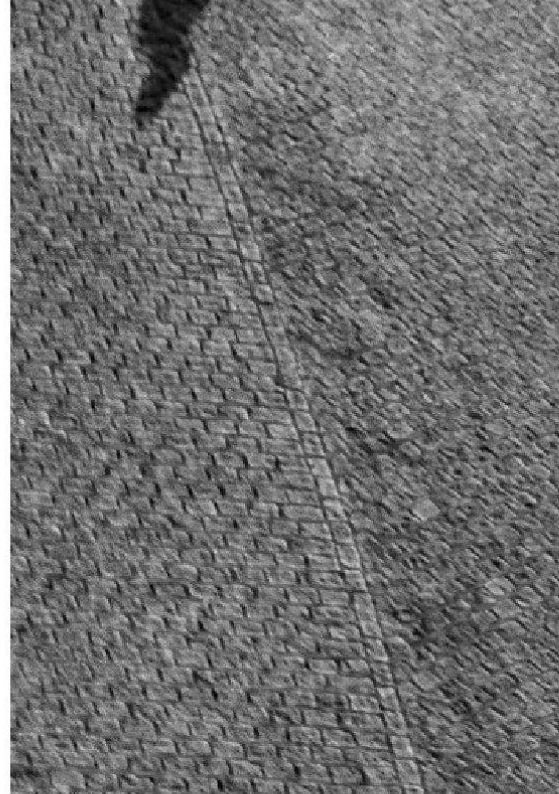


Example application II. - noise filtering

part of orig. input



part of filtered output



Exercise 1

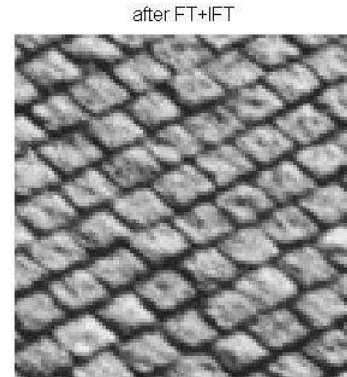
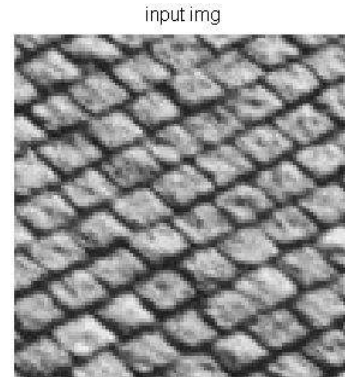
Create a function that:

- realizes the forward Discrete Fourier Transformation,
- the parameters of this function should be as follows:
 - *input1*: image matrix
 - *output*: Fourier Transformation of the image;
- it should work with grayscale images (check if the input image is grayscale),
- *note*: the naive implementation of the transformation is $O(N^4)$, do not use it on large images,
- *note*: the imaginary number in MATLAB is `i`, the built-in constant for π is `pi`, the exponential function is `exp()`.

Exercise 1

Create a script to your function that:

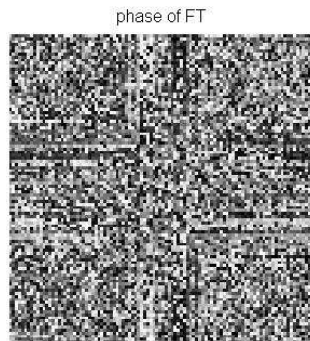
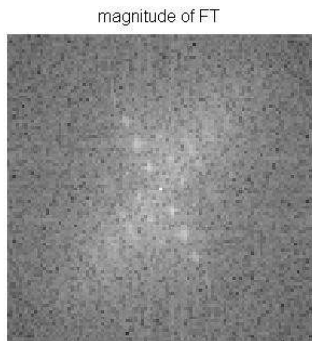
- reads in an image,
- converts it to grayscale,
- crops (**`imcrop`**) it or resizes (**`imresize`**) it to a reasonable size (lower than 100x100, eg.: given the example image LucienHerve_ParisSansQuitterMaFenetre.jpg, crop a 100x100 sized part of it with **`imcrop`** from pixels starting [430, 640]),
- calls your DFT function on the image,
- calls MATLAB's built-in inverse Fourier Transformation on the output image of your function (see **`ifft2()`**),
- displays the original input image and the output image of **`ifft2()`**, like this:



Exercise 2

Extend the above script with

- displaying and saving the phase and magnitude images of the DFT-image
 - use `fftshift()` to center the output of your DFT-function,
 - use `abs()` and `log()` functions to better visualize the magnitude-part,
 - use `angle()` function to retrieve the phase angles;



- *note:* try to avoid `log(0)`; scale the images before `imshow` (either if you give a parameter with type `uint8`, the value-range of it should be between `[0, 255]`; or if you give a parameter with type `double`, the value-range of it should be between `[0, 1]`).