SCC0251/0651/5830 + MAI5020 — Prof. Moacir Ponti

PAEs: Ana Letícia Vicente, Marcus Vinícius Costa, Sherlon Almeida,

Assignment 2:
Fourier Transform & Filtering in Frequency Domain

Code the assignment by yourself. Ask if you need help. Plagiarism is not tolerated.

1 Introduction

In this assignment, you have to implement functions in order to filter images in the frequency domain using the Discrete Fourier Transform. Read the instructions for each step. Use python with the numpy and imageio libraries.

Your program must have the following steps:

1. **Read** the parameters:
   a) Filename for the input image (I).
   b) Filename for the reference image (H).
   c) Filter index $i \in M = \{0, 1, 2, 3, 4\}$.
   d) Filter Parameters respective to each index;
2. **Implement** the filters below:
   a) **index i = 0** - Ideal Low-pass - with radius r.
   b) **index i = 1** - Ideal High-pass - with radius r.
   c) **index i = 2** - Ideal Band-stop - with radius r_0 and r_1.
   d) **index i = 3** - Laplacian High-pass.
   e) **index i = 4** - Gaussian Low-pass - with σ_1 and σ_2;
3. **Process** the input images:
   a) Generate the Fourier Spectrum (F (I)) for the input image I.
   b) Filter F (I) multiplying it by the input filter M_i.
   c) Generate the filtered image G back in the space domain.
   d) Compare the output image (G) with the reference image (H).
   e) (Recommended) Save each restored image for further observations;

2 Filters

Considering that the information present in images (like noise or edges) are well represented in the frequency domain (Fourier Spectrum), it is possible to design filters to remove or keep those patterns according to their usefulness for a desired task. In this assignment you will implement the following filters using mathematical functions: low-pass, high-pass, band-stop, Laplacian, and Gaussian.

Usually, higher frequencies represent the details of images, for example, the borders that highlight transitions of elements. Thus, applying a high-pass filter to the output image preserves the edges of the original image (see Figure 1).

On the other hand, low frequencies represent the "heart" of the image, however with smooth changes between the neighbor pixels. Thus, by applying a low-pass filter the output image partially removes the noise, at the same time blurs the result (see Figure 2).

Figure 1: Applying a high-pass filter.

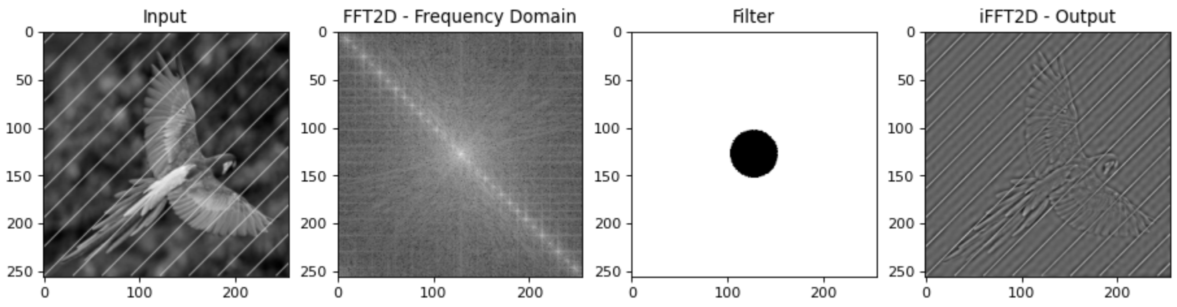
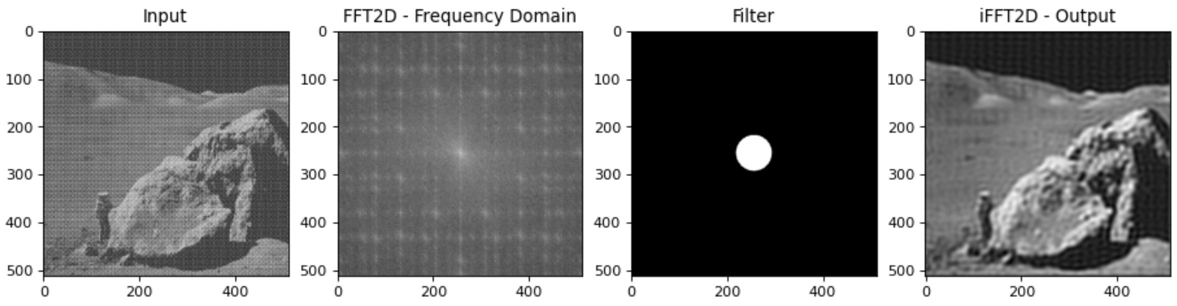


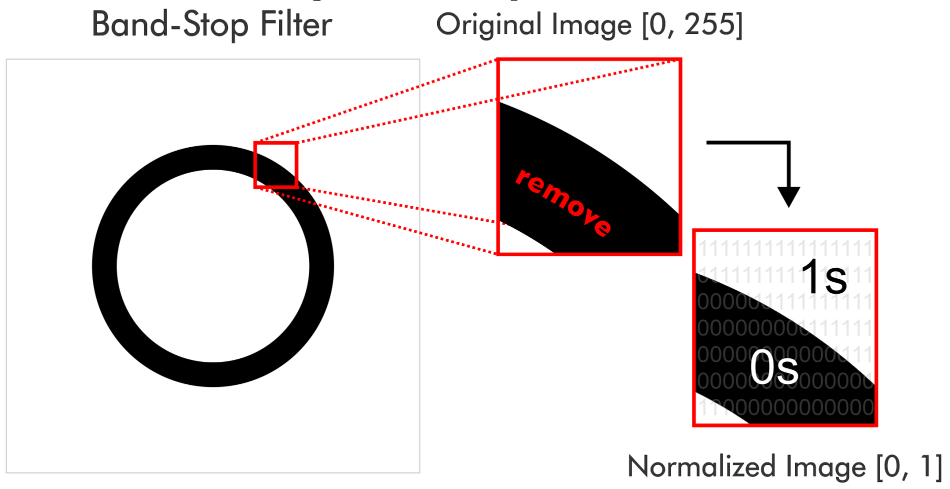
Figure 2: Applying a low-pass filter.



It is important to remember that the same result of a convolution in the spatial domain can be obtained by a simple multiplication in the frequency domain. Thus, the generated filters must have the same dimensions as the input image.

Also, the filters must be represented by an [0, 1] interval, where values close to 0 "remove" undesired frequencies while values close to 1 "keep" desired frequencies (see Figure 3). To perform the multiplication, you can use the np.multiply(...) function between F(I) and the filter M_i.

Figure 3: Filter representation.



2.1 Ideal Filters

An ideal low-pass filter (ILPF) is a 2D filter that allows all frequencies within a circular region with a given radius (r) from the center of the image to pass without attenuation, and blocks all frequencies outside this region. On the other hand, an ideal high-pass filter (IHPF) does the opposite, preserving frequencies outside the circle defined by a radius (r), removing frequencies within the circle.

Also, it is possible to select an specific interval to filter (keep or remove), such as band-pass and band-stop (also known as band-reject), which select a range of frequencies defined by different radius (r_0 and r_1). Both algorithms can be created from operations with low-pass and high-pass filters. For example, assuming the radius $r_0 > r_1$, you could subtract a ILPF with radius r_0 from another ILPF with radius r_1 to get a band-stop filter.

The filters above can be mathematically (see Equations 1-4) and visually (see Figure 4) described as follows:

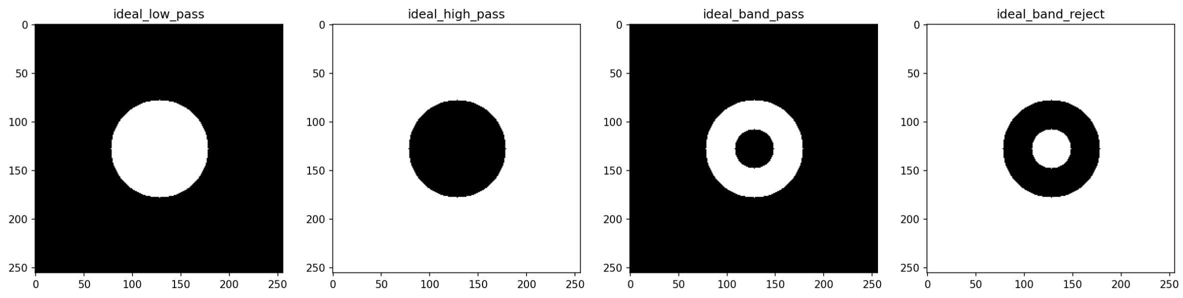
$$D(u,v) = \sqrt{(u - \frac{P}{2})^2 + (v - \frac{Q}{2})^2} \tag{1}$$

$$\textbf{Low-pass} \rightarrow H(u,v) = \begin{cases} 1, & \text{if } D(u,v) <= r \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$\textbf{High-pass} \rightarrow H(u,v) = \begin{cases} 0, & \text{if } D(u,v) <= r \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

$$\textbf{Band-stop} \rightarrow H(u,v) = \begin{cases} 0, & \text{if } D(u,v) \in [r_1,\ r_0] \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

Figure 4: Representation of ideal filters.



2.2 Laplacian

The Laplacian operator is a second-order derivative that is used in image processing to enhance edges and details. The Laplacian filter can be implemented as a high-pass filter to enhance edges or as a low-pass filter to smooth an image. The equation 5 is the frequency response of a Laplacian low-pass filter in the frequency domain.

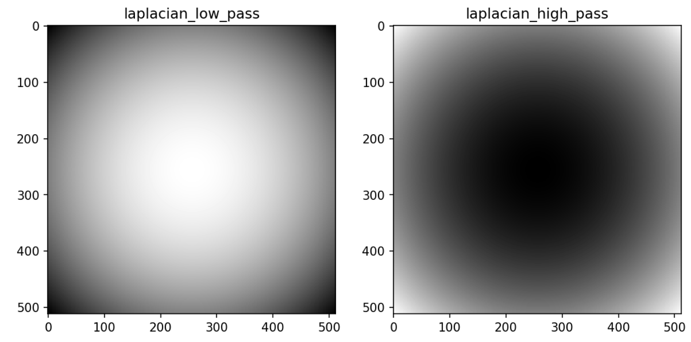
$$H[u,v] = -4\pi^2((u - \frac{P}{2})^2 + (v - \frac{Q}{2})^2) \tag{5}$$

The filter is defined by the function H(u,v), where u and v are the spatial frequency variables, and P and Q are the dimensions of the image in the x and y directions, respectively.

The Laplacian filter is a high-frequency filter that attenuates high-frequency components in the image. The equation achieves this by applying a negative value to H(u,v) for higher frequency components. The squared distance between the frequency components (u,v) and the center of the frequency domain is calculated and multiplied by a factor of $-4\pi^2$. This factor determines the rate of attenuation for higher frequency components. The farther away the frequency component is from the center, the greater the attenuation.

2.3 Gaussian

A Gaussian low-pass filter is a type of filter used in image processing to suppress high-frequency information in an image, providing a smooth alternative to the ideal low-pass filters previously



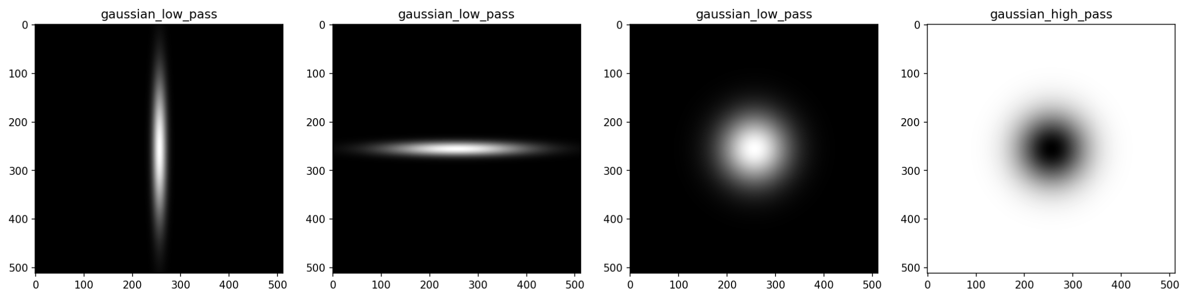
presented; the effect is effectively a "blur" and is the filter most commonly used in user interface elements.

The equations 6 and 7 represents this filter. The standard deviation, σ, determines the width of the Gaussian curve, with a smaller value of σ resulting in a sharper filter.

$$x = (\frac{(u - \frac{P}{2})^2}{2\sigma_r^2} + \frac{(v - \frac{Q}{2})^2}{2\sigma_c^2}) \tag{6}$$

$$H(u, v) = e^{-x} \tag{7}$$

The frequency domain representation of the Gaussian filter is expressed in terms of the frequency coordinates in the row and column directions, denoted by $u - P/2$ and $v - Q/2$, respectively. The filter response at each frequency coordinate is determined by evaluating the Gaussian function at that coordinate. The exponent in the equation is the squared distance of the frequency coordinate from the center of the filter, divided by twice the squared standard deviation in the corresponding direction. Note that you can subtract the filter from a fully blank filter to get a high-pass version.



3 Discrete Fourier Transform (DFT)

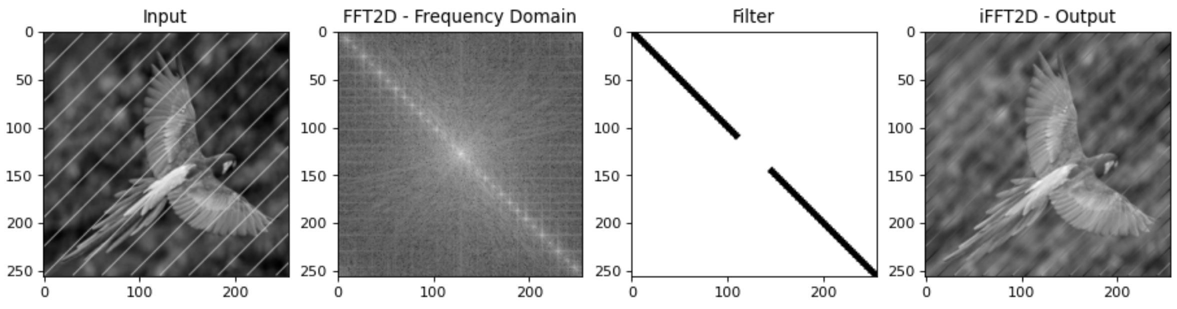
Performing a transformation in the original domain of an image using DFT 2D changed the way of processing images. Processing images in the frequency domain is faster than its version in the spatial domain, and also it allows us to identify noisy frequencies that can be filtered easily.

Figure 5 illustrates the approach you have to implement. Given an image as input, firstly you must generate its Fourier spectrum, which will be multiplied by a generated filter to obtain a filtered image back in the spatial domain.

Initially, it is essential to understand the concepts behind the Fourier transformation and the spectrum analysis. The higher frequencies are far from the center, while the lower frequencies are close to the center.

To generate the frequency domain you can use the Fast Fourier Transform (FFT) implementation from Numpy library `np.fft.fft2(...)`, apply the shift `np.fft.fftshift(...)` to center the lower frequencies, multiply it by the filter M_i, apply the inverse shift, and get the restored image by computing the inverse FFT `np.fft.ifft2(...)`.

Figure 5: Step-by-Step to implement in this assignment.



4 Input and Output

The following parameters: Input image (I), reference image (G), filter index i from M, and the filter‘s parameters.

input	**output**

`apollo17.png`
`apollo17_ref.png`
`1`
`2.0`
	0.7079

5 Comparing against expected

Your program must compare the restored image G against reference image H. This comparison must use the root mean squared error (RMSE), as follows:

$$RMSE = \sqrt{\frac{1}{m \times n} \sum_i \sum_j (H(i,j) - G(i,j))^2}$$

where $m \times n$ is the size of the image.

6 Submission

Submit your source code to `https://runcodes.icmc.usp.br` (only the `.py` file).

1. **Use your USP number as the filename for your code.**
2. **Include a header**. Use a header with name, USP number, course code, year/semester and the title of the assignment. A penalty on the evaluation will be applied if your code is missing the header.
3. **Comment your code**. For any computation that is not obvious from function names and variables, add a comment explaining.
4. **Organize your code in programming functions**. Use one function for each filter method and a separate function for the frequency domain processing.