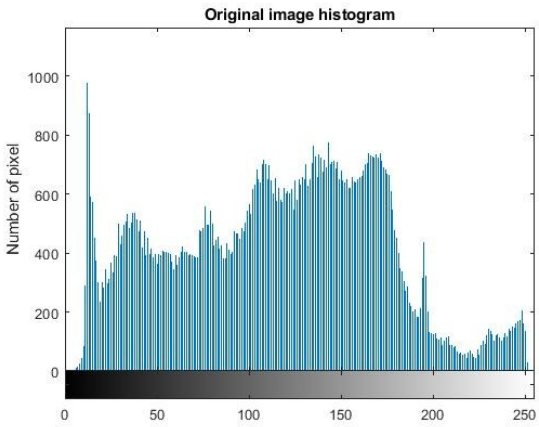
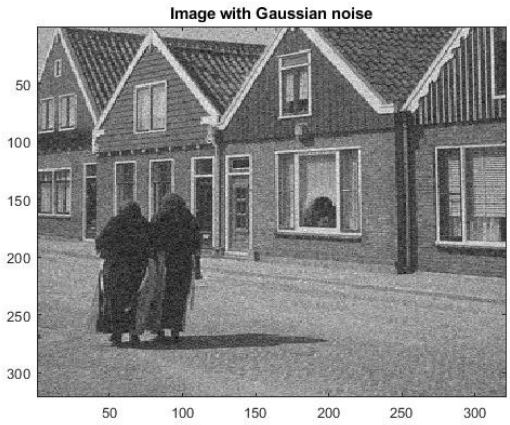
Image filtering and Fourier Transform

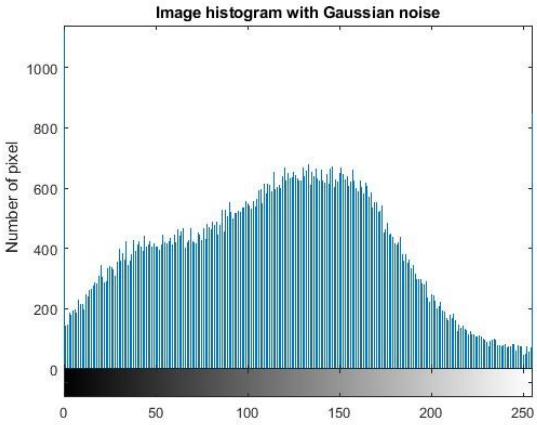


The aim of this lab session is to modify noisy images filtering them and study the Fourier transform.

We have two functions that add noise:

Gaussian noise (noiseGauss.m)

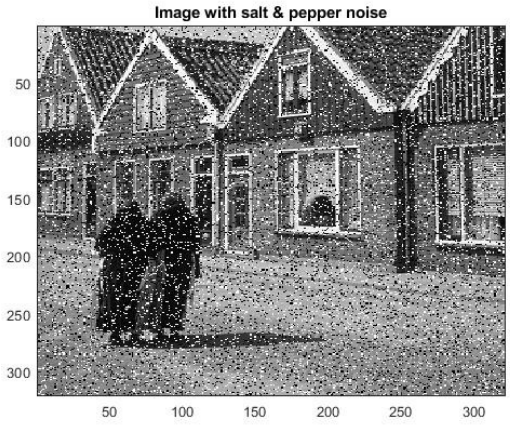


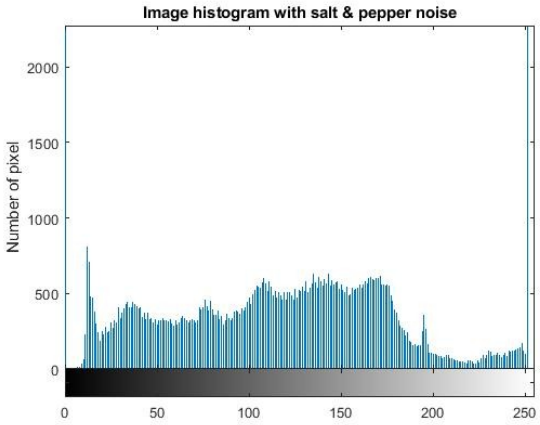


For this we use the function randn() (multiplied for the standard deviation) to compute the noise matrix that will be added to the image.

Fabio Fusaro
Davide Torielli
Luca Morando

Salt & Pepper noise (noiseSP.m)

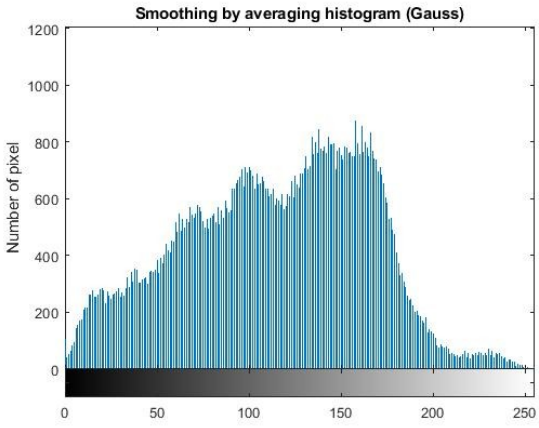




For the Salt & Pepper we use the function sprand() to compute the values. The result is passed to full() to have the values stored as a matrix (because sprand() store values as vector).
Two masks are used to put values of the noise matrix either to 0 or 1 before adding the real salt & pepper noise to the image.

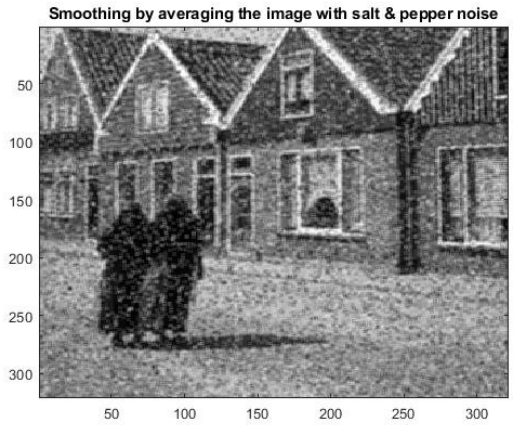
To remove these noises, we have three types of filters:

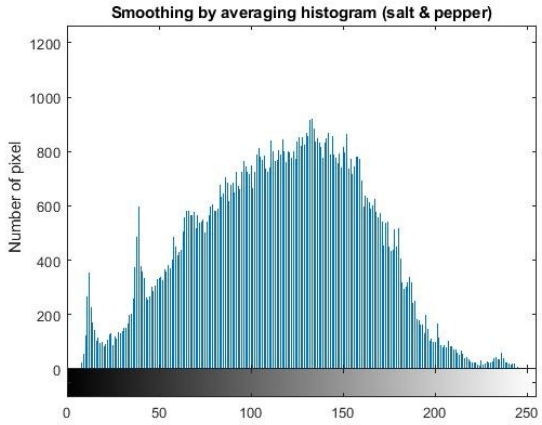
Moving Average filter (filterMovingAverage.m)



This filter is a box filter: a matrix of all ones multiplied by 1/filterSize (in this case the filterSize is 3).
This is made for give same weight to the neighbours of the central pixel. Then this matrix is convoluted with the original image.
With this the image will be more uniform but less nitid (as it would be a little out of focus). The more the filterSize is, the more the image is blurred because the average is made among more pixels.

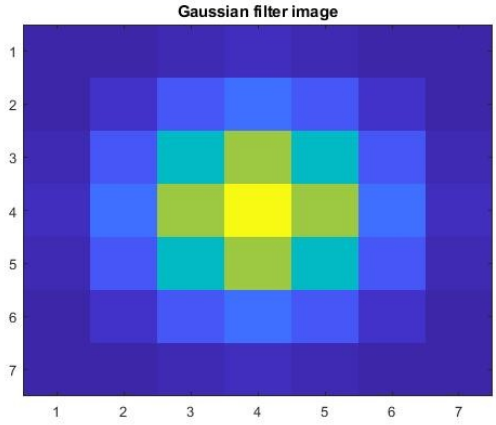
Fabio Fusaro
Davide Torielli
Luca Morando

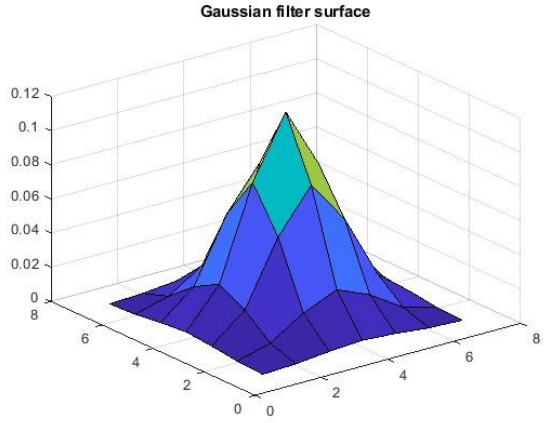


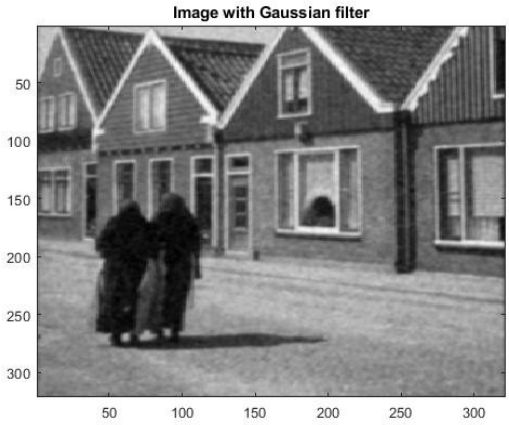


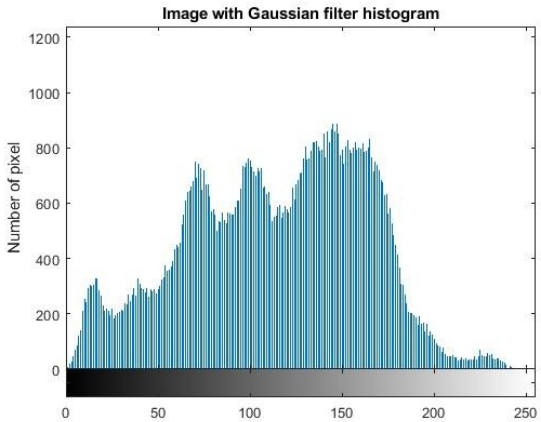
If we apply this filter to remove the Salt & Pepper noise, we clearly see that the noise still affect the image. For this noise we need another filter: the median one.

Low-Pass Gaussian filter (filterGauss.m)





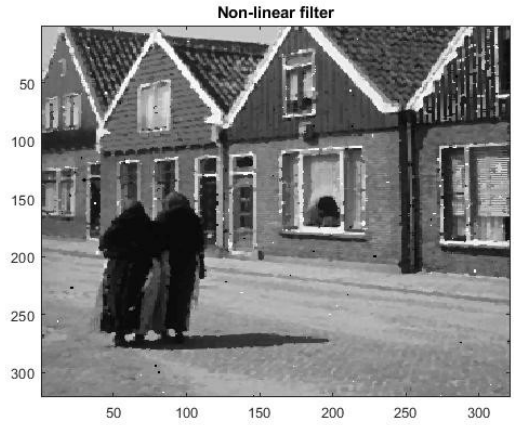


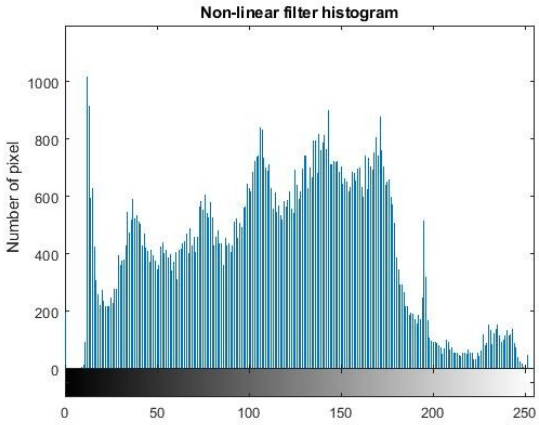


Fabio Fusaro
Davide Torielli
Luca Morando

The filter matrix for the gaussian filter has values that are bigger near the center and smaller in the border. So it gives more weight to the center neighbours when we make the convolution. Then the matrix is convoluted like the moving average method.
In the photo above we use a filter of 7x7 size.

Median Filter (filterMedian.m)





Differently from the other two filters, this is a non linear filter. It tends to discards outliers, that are pixel that are very different from the ones near. This is particulary useful for the Salt & Pepper noise (as it is possible to see in the images), for which the two linear filter are not suitable.

Fabio Fusaro
Davide Torielli
Luca Morando

Various Linear Filters (linearFilters.m)

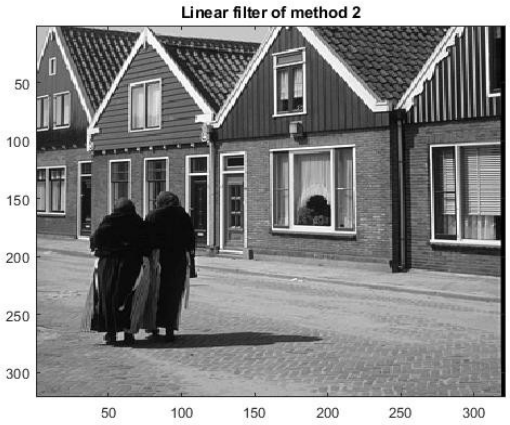
For the third point, we implement linearFilters function which filters with different method the given image. The final image is the result of the convolution between original image and the choosen filter. These three filters are not created to remove the noise but to modify in someway the image.

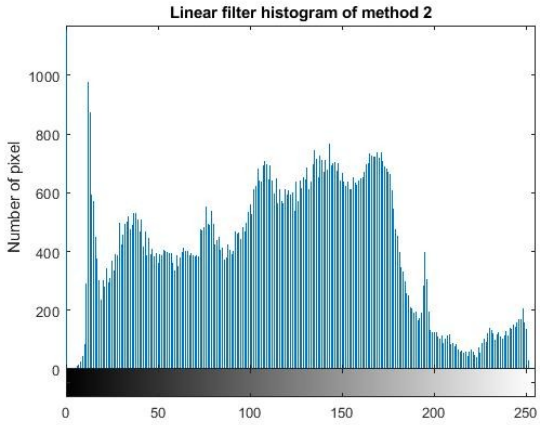
Impulse filter (method 1)

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This filter keeps the image inaltered because with the kernel we take only the central pixel when making the convolution.

Shifted left filter (method 2)



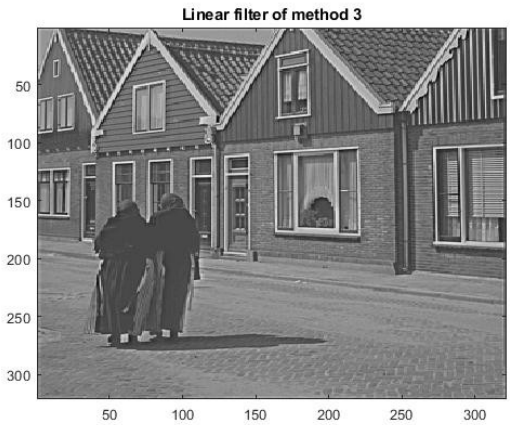


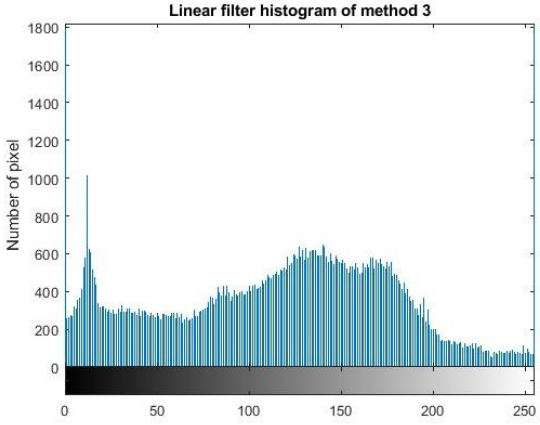
$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The kernel here has only one 1 in the middle of the first column. When we convolute, the resultant image is the original one only shifted to the left.
The more the kernel size is, the more the pixels are shifted.

Fabio Fusaro
Davide Torielli
Luca Morando

Sharpening filter (method 3)



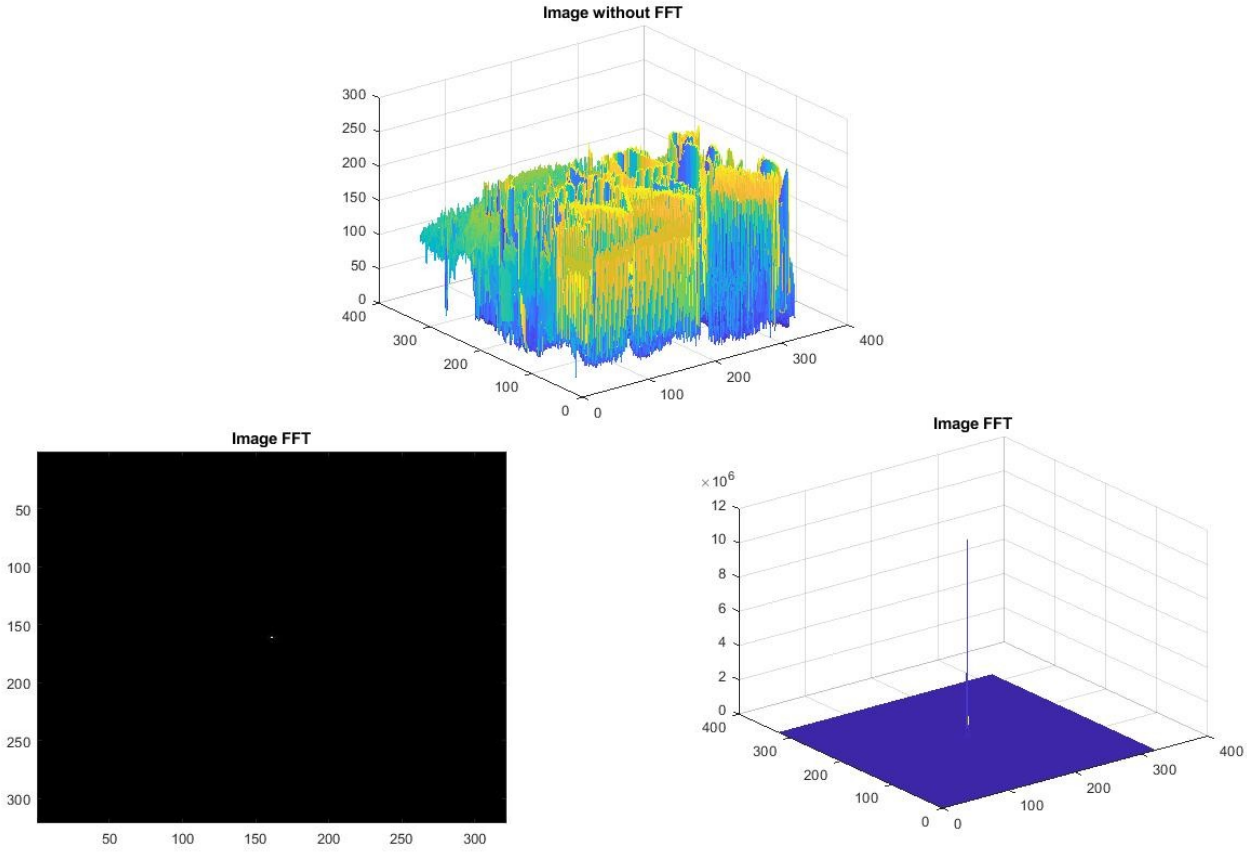


$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} - 1/9 * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This filter accentuates the differences in the center of the image. The resultant is an image with the center pixel more “nitid” than the ones nearer to the border.

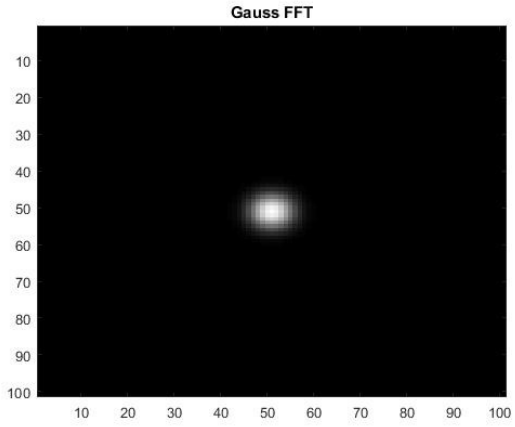
Fabio Fusaro
Davide Torielli
Luca Morando

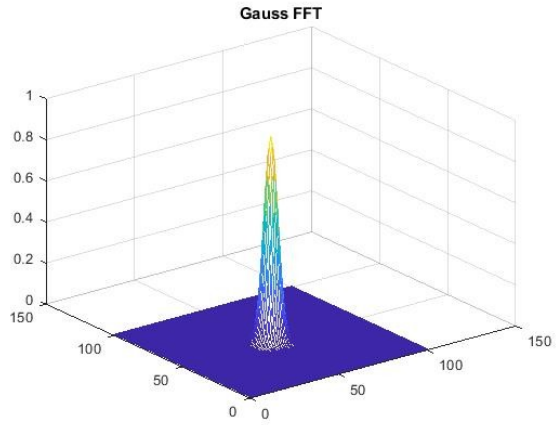
Fourier transformations



In this section we made the Fast-Fourier Transformation of the image by the function fft2( ) which has as input the original image.
Then it is necessary to use the function fifthsift( ) because it shifts the zero-frequency component to the center of the spectrum.
At this point with imagesc( ) and mesh( ) we visualize the transformed image putting as input of the function the magnitude of the spectrum.
This is similar to an impulse because the values of the image are almost uniform distribuited.

Fabio Fusaro
Davide Torielli
Luca Morando





Next we have done the fft of the gaussian filter with size of 101x101 pixel.
The filter is built by the function fspecial( ) where we have also specified the dimension and desired standard deviation. Once the filter is created we have done the transformation and then centered the zero frequency.
In the image it is possible to see the gaussian shape because the transformation is again a gaussian shape.

Fabio Fusaro
Davide Torielli
Luca Morando