IMAGE FILTERING AND FOURIER TRANSFORM

Authors: Ierardi Ambra, Scorrano Andrea, Trovatello Alessandro, *Università degli Studi di Genova*

There will be only *one version* of the code and the report submitted, since the group worked together over every part the assignment.

ABSTRACT: This project was focused on the implementation of functions which added noise to some given pictures to subsequently remove it, and on the computation of the *Fourier Transform* of the provided images and of some of the created filters. The objective of this lab was to improve Computer Vision skills, to acquire more knowledge about MATLAB and to complete many tasks in a pre-established time (3 weeks). All the functions were built in MATLAB and the theory behind them comes from the course's slide; however some functions were already implemented in MATLAB libraries such as *medfilt2*, used to execute the **Median Filter**, and *conv2* exploited to apply the **Convolution Operator** and many others.

KEYWORDS: Image filtering and Fourier Transform.

1 INTRODUCTION

This assignment was centered on the first basic steps to introduce the students to the subject of image processing. These operations are necessary to improve the quality of the images, which, in the Robotics Engineering field, is fundamental to achieve a good kinematics, dynamics and behaviour of the robot in the environment, based on what it sees, through the camera.
The images captured by a digital camera bring with themselves noise, which can be *deterministic*, that is the *digitalisation noise*, even if it is very low nowadays, or not, which is the noise intrinsic to any real, non ideal, signal: in particular two kinds of noise have been studied in this project, the *Gaussian noise* and the *Salt & Pepper* one. Both of them worsen the quality of the image, in different ways, and this will be further discussed in the next chapters. This explains the importance of a good filter, possibly matched to the noise.
Then, other kinds of masks were implemented, mainly in order to enhance details in the image, to make them more visible. The *Fourier Transform* of the image tells where the information is, *i.e.* which frequencies are predominant, and this is exploited to choose a proper filter, which enhances the components in which the signal is more present with respect to the noise.

2 REPORT TASKS

As first thing, running the main function, under the name **main.m**, the two images are loaded, in order to work on both of them later on. Then, the function **print_original.m** prints both pictures and their histograms, to help notice the distribution of the pixels' values before adding the noise.

2.1 Noise addition

The first task of the assignment concerned the addition of the noise, implemented in the function **add_noise.m**:

- first, it was added the *Gaussian noise* to the original image, by summing to the latter a random matrix of the same size, multiplied by the σ value equal to 20, which is the standard deviation of the probability density function;
- then, the same procedure was applied to the initial picture, by adding, through the use of masks, random black and white pixels, with density of 20%: this one corresponds to the *Salt & Pepper noise*.

In order to point out the effect of the two different kind of noise on the pixel values, the histogram was created, one for each disturb.

2.2 Noise filtering

The second task developed was the filtration of the noisy images, in order to smooth the alterations introduced in the previous point. Three different methods were tested, in order to check the effects on the each noise. Each of them was created in two sizes: 3×3 and 7×7.

- The first technique used is the $moving\ average\ filter$, implemented in the function **rm_byaveraging.m**, which consists of a square matrix of ones, averaged by the sum of each element in it. This filter takes a subset of the input matrix and produces an average of it, by convolving it with the image, and repeating this action until covering all the matrix surface;

- The second method implemented is the $Gaussian\ Low-Pass\ filter$, which performs a weighted average of the image: in the central part it is assigned the highest value, while proceeding towards the edges of the matrix the weight decreases sharply. The formula describing the Gaussian function is the following:

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{1}$$

  where σ is the standard deviation, x and y are the variables of the 2D space. The σ value was chosen according to the following rule:

$$3\sigma = \frac{SP}{2} \rightarrow \sigma = \frac{SP}{6} \tag{2}$$

  where SP is the spatial support, that is the number of pixels of the edge of the filter's matrix, in order to have a correct approximation of the Gaussian function, and to avoid the creation of a paraboloid, which is a wrong shape, since this would give the too much weight to the lateral pixels;

- The last technique applied, implemented in the function **rm_median.m**, is the $median\ filter$, which is a non linear filter that preserves the edges and discards outliers: the algorithm underlying is the analysis of the values of the image under the current position of the matrix, that consists in sorting the values in a growing scale and replacing the middle value of the matrix, in the image, with the median, that is the value in the middle of the sorted vector of numbers. This practice is useful to remove spikes, especially present in case of $Salt\ and\ Pepper$ noise, which effects have already been discussed in the previous subsection.

2.3 Practice with linear filters

In the third task, it was required to implement many different filters, each of them with spatial support of 7x7 pixels, to apply them to the images and to display them both with the function **imagesc()**, which gives a $top\ view$ prospect, and with **surf()**, that gives a $3D\ view$.

- the first one has been built in the function **no_change.m**, named this way because of the effect of the filter, since the output is equal to the input. The filter is described by the matrix A;

- the second filter implemented is the $right\ shift$ of the image, present in the file **shift_right.m**, which is implemented in the matrix B;

- the third method is essentially the same already presented in the **rm_byaveraging.m** function, since it is a $smoothing$, or $blurring$, $filter$. It is built in the file **blur_box.m**, and the matrix has the structure as the in the C one;

$$A=\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}\quad B=\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}\quad C=\frac{1}{49}\begin{bmatrix}1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\end{bmatrix}$$

- the fourth filter implemented is the $sharpening\ filter$, in the function **sharpening.m**, which highlights the differences of the image with the local average, in order to use them in the next point to accentuate the details. The matrix is built by summing two matrices equal to the first linear filter implemented, and then subtracting a matrix made as in the previous step. The filter will have this shape:

$$\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}+\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}-\frac{1}{49}\begin{bmatrix}1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\end{bmatrix}$$

- the last method implemented is present in the file **details.m**: it takes the input and subtracts to it the smoothed image, built in the second point of this subsection. Then, in the **sharp.m** function, the output of the previous function, which are the $details$ of the image, is multiplied by a weight α, to enhance it, and then it is summed to the original photography, to $remark\ the\ details$.

2.4 Apply the Fourier Transform (FFT)

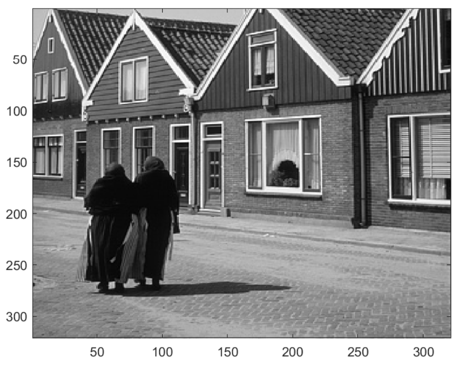
In this last task, it was required to apply the $Fourier\ Transform$, with the MATLAB command $fft2()$, which provides the $2D$ fast Fourier Transform (FFT) of the input, to different images:

- first, in the **fft_image.m** function, it is computed the FFT of the original image, then its absolute value with the Matlab command abs, in order to display it;
- then, it is computed the transform of the Low-Pass Gaussian filter, with σ=5 and spatial support 101x101, again with the function **fft_image.m**;
- as last point, in the file **fft_sharp_custom.m**, has been created a filter of 101x101 pixels performing the sharpening of the image, and then it was transformed, and its absolute value was computed.

All these transformed images have been plotted using the following command: $log(fftshift(image))$, where $image$ is the input to be plotted, $i.e.$ the matrix of the picture transformed, $fftshift()$ shifts the zero-frequency components to the center of the plot and $log()$ computes the logarithm of the argument, in order to have a clear vision of what is happening in the frequencies of interest.

3 RESULTS

Looking at the histograms of the original images, it is possible to see how the pixels are more or less uniformly distributed, except for a dark peak (at low intensity) in the $i235.png$ image, due to the presence of the two black figures in the foreground.


(a)

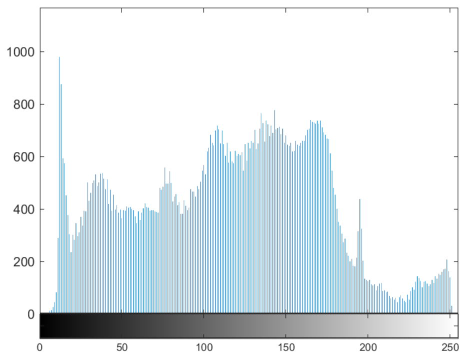

(b)

Figure 1: (a) i235 Image. (b) i235 Histogram.

3.1 Noise addition

The noises applied were the $Gaussian$ and the $Salt\ \&\ Pepper$ one. When the distortions were applied to the image, it was evident that the $Gaussian\ noise$ was affecting the image less then the $Salt\ \&\ Pepper$ one. Indeed, it is possible to see that the latter replaces pixels randomly with white or black ones and the disturb is more visible to the human eye. Looking the histograms of the pictures with $Gaussian\ noise$, it can be noticed that the pixels' values appear to form a sort of Gaussian curve, which makes sense because of the definition of $Gaussian\ noise$, that is a phenomenon with uniform probability density distribution; while looking at the histogram of the image affected by the $Salt\ \&\ Pepper\ noise$, there are two peaks at the extremities, which correspond to the 0 and 255 values, due to the new black and white pixels, introduced by the algorithm.

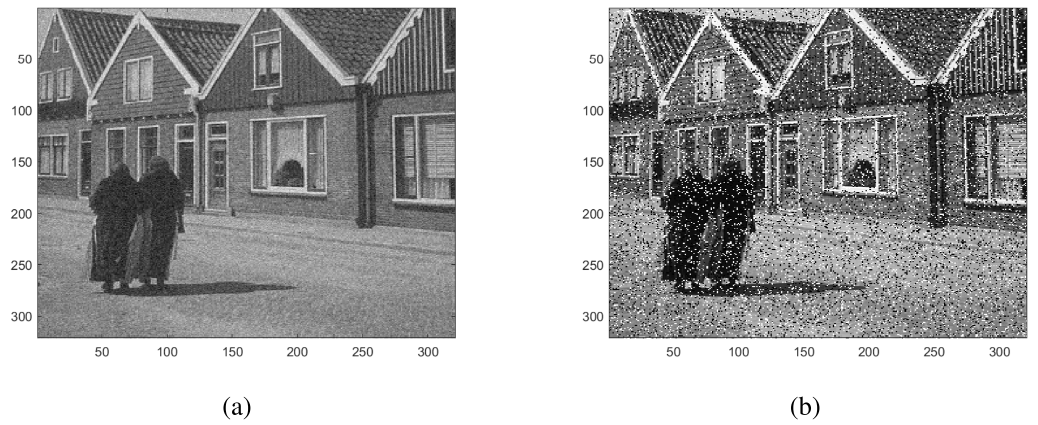


Figure 2: (a) $Gaussian\ Noise$ applied to the Image. (b) $Salt\ \&\ Pepper\ Noise$ applied to the Image

3.2 Noise filtering

Once added the $Gaussian$ and $Salt\ \&\ Pepper\ noise$, the next part of the assignment was focused on the implementation and application of different filters, in order to remove the disturb and obtain an acceptable image.
The first filter made is the **moving average**, which consists in a flat surface, since all of the values are equal. Once done the convolution between image and the filter, different results came out, based on the spatial support.

- The average filter with a $smaller\ spatial\ support$ (3x3) preserves more details of the original image compared to the former filter. This kernel is not enough for the $Salt\ \&\ Pepper\ noise$, while it is quite efficient in case of $Gaussian\ noise$.
- The average filter with the $larger\ spatial\ support$ (7x7) has a more significant blurring effect on the image. This means that details are further lost, and the image appears more uniform and less detailed. This size is good to remove $Gaussian\ noise$, while in case of $Salt\ \&\ Pepper$ one it is not sufficient, even if the effect is better with respect to the 3x3 kernel.

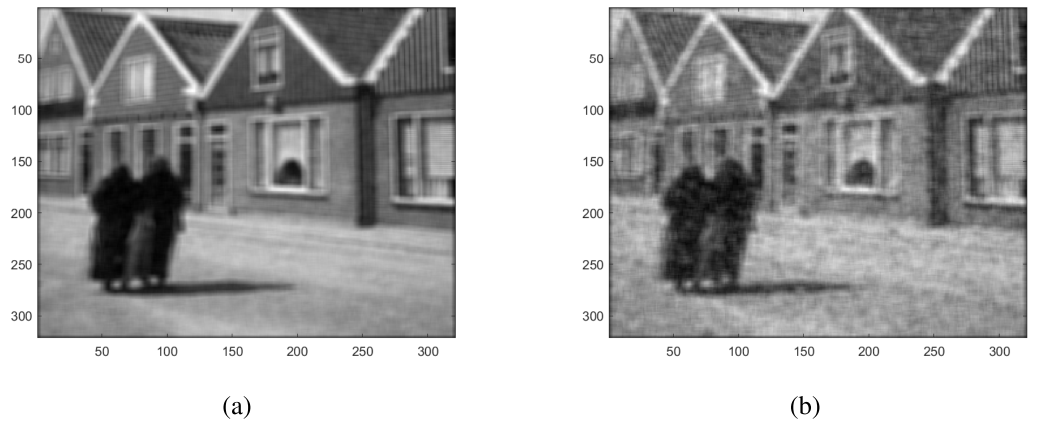


Figure 3: (a) Average filter 7x7 on $Gaussian$ Noisy Image. (b) Average filter 7x7 on $Salt\ \&\ Pepper$ Noisy Image.

The second filter implemented is the **Gaussian low-pass**: in this case it can be noticed that the surface is no more flat, but it has the shape of the Gaussian function, and it varies depending on σ and on the size of the filter. As in the previous case, two different spatial supports were used: 3x3 and 7x7.

- The $3x3$ Gaussian filter has a smaller kernel: this means that it affects fewer pixels in the image and has a gentler blurring effect. It preserves more details with respect to the 7x7 filter but is less effective in noise reduction. In case of the $Salt\ \&\ Pepper\ noise$, this filter is almost useless, while it is efficient for the $Gaussian\ noise$;

- The $7x7$ Gaussian filter is more effective in noise reduction and blurring, and consequentially tends to lose more details. This filter is even more smoothing in case of Gaussian noise, while in the other case it is still not so useful.

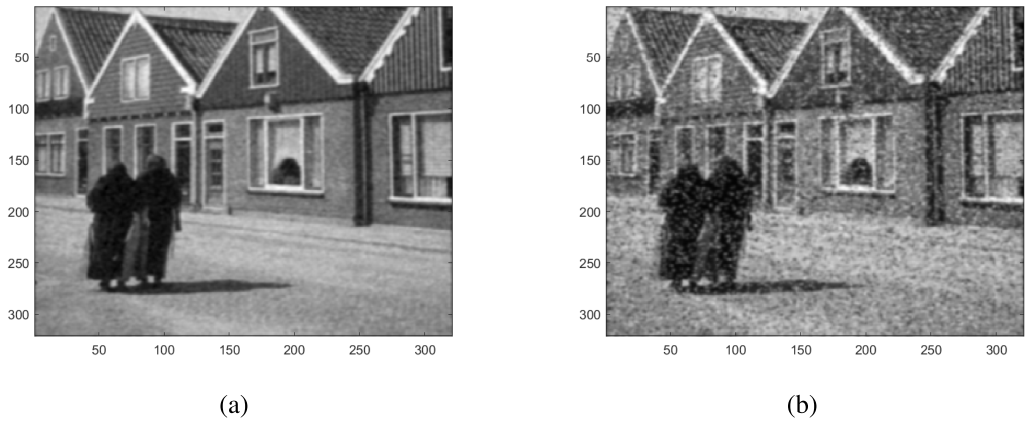


Figure 4: (a) Gaussian filter 7x7 on $Gaussian$ Noisy Image. (b) Gaussian filter 7x7 on $Salt$ & $Pepper$ Noisy Image.

The third filter used is the **median**, which wasn't plotted because the median filter changes continuously, so it is not unique. This filter is very good to correct the $Salt$ & $Pepper$ $noise$, while for the $Gaussian$ one it is still good, but the previous ones are better.

- The median filter with a $smaller$ $spatial$ $support$ (e.g., 3x3) is less effective to remove noise but preserves finer details in the image. The resulting image has less smoothing effect and may still contain some minor interference. This size is good, bus some unexpected white or black pixels still persist in case of the $Salt$ & $Pepper$ $noise$.
- The median filter with a $larger$ $spatial$ $support$ ($e.g.$, 7x7) is more effective at removing larger and more uniform noise in the image. This results in a smoother, less noisy image but with some loss of fine details. This kernel is better, since it removes more wrong pixels, together with the smoothing of the image, which is effective for the $Gaussian$ $noise$, even tough the other two filters are better in this case. For the $Salt$ & $Pepper$ $noise$, this kernel is very accurate, since the image has no more black and white pixels where unexpected.

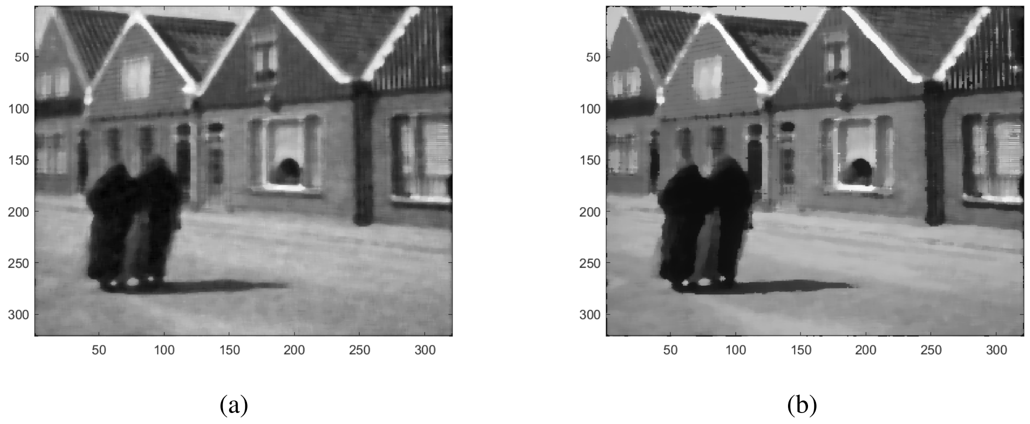
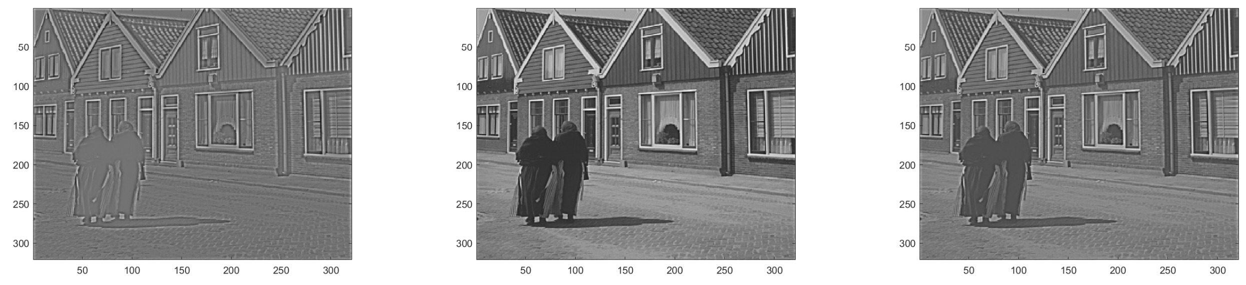


Figure 5: (a) Median filter 7x7 on $Gaussian$ Noisy Image. (b) Median filter 7x7 on $Salt$ & $Pepper$ Noisy Image.

3.3 Linear Filters

The third point of the assignment was focused on the application of $linear$ $filters$. In this step, the convolution of the original images with different matrices was implemented.

- In the first point, which was the convolution of the image with a matrix of zeros except for a one at the center of it, which essentially corresponds to a sort of $impulse$, no change is made to the picture.
- The image that was convoluted with a matrix with a 1 on the right returns as a result an image $shifted\ to\ the\ right$ by one pixel.
- The image convoluted with a matrix whose components are ones, normalized by the area of the filter, named also as $box\ filter$, results in an image with a $blurred\ effect$.
- To obtain a $sharpened\ picture$ where the discontinuities in illumination are underlined and increased, the image was convoluted with the matrix resulting from the summation of the double of the matrix at the first point and the opposite of the matrix implemented at the previous point.
- In the last point of this task, the goal was to $enhance\ the\ details$ of the image: this has been possible by subtracting from the original image the smoothed, blurred one. Once the details were extracted from the original image, it was possible to remark these ones by multiplying them by a factor α and adding the result to the original image. Increasing α, it is visible how the details are accentuated, up to overwhelm the original image. In the code it is possible to observe two cases: α=0.5 and α=1.5. In the first case the details are more evident with respect to the original image, while in the second one the authentic picture's colors are almost buried because of the enhancement of the details.



(a) (b) (c)

Figure 6: (a) Details of the Image. (b) Result of adding original image and details of the image moltiplied by α. (c) Result of adding original image and details of the image moltiplied by 3α

3.4 Apply the Fourier Transform

The last step of the assignment was based on the application of the FFT (Fast Fourier Transform) to the original images, to the Low-Pass Gaussian filter with σ=5 and spatial support equal to 101x101, and to the sharpening filter, and showing the magnitude of the relative results.

- It is possible to notice that the $magnitude\ of\ the\ images$ was not similar: the image **i235.png**'s absolute module has many vertical, horizontal and cross lines, other than the central peak. This can be possibly brought back to the geometrical patterns present in the picture, $e.g.$ the straight lines of the houses, windows and oblique roofs; in the **tree.png** picture, there are only two not clearly visible patterns, one central vertical line, which can be originated by the trees, and a faint horizontal one, which might be carried out by the difference between the foreground, with the lighter trees, and the background, made of darker pixels. The central white peak in the spectrum stands for the main components at low frequencies, due to the dominating constant backgrounds, such as the street in the first image and the soil or the dark, far trees in the second one. The straight lines in the spectrum might be the result of the high frequency transitions due to the straight lines in the images, $e.g.$ the tree in foreground, in the bottom part of the picture, has a darker color with respect to the grass, and the change of the tone happens abruptly, which corresponds to a sort of $step\ function$.

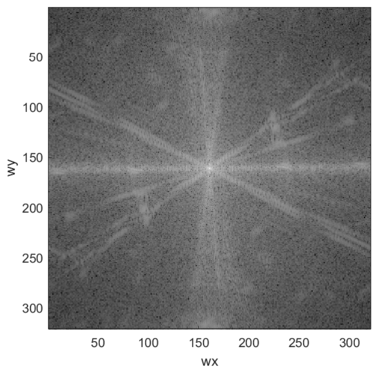


Figure 7: Magnitude of the i235 image.

- It is also possible to notice the frequency components of the $Low - Pass\ Gaussian\ filter$, mainly concentrated at the center of the graph, since it has a transition in shape limited at low frequencies, because it doesn't happen so suddenly.

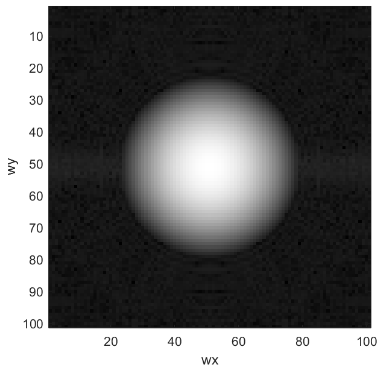


Figure 8: Magnitude of the $Low - Pass\ Gaussian\ filter$.

- In the last point, it can be noticed the $magnitude\ of\ the\ sharpening\ filter$, which has high components at high frequencies: this can be can be explained by the enhancement of the details, which results in sharp transitions between adjacent pixels.

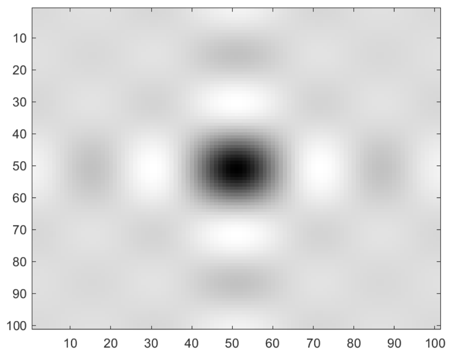


Figure 9: Magnitude of the $sharpening\ filter$.

4 SUMMARY

This lab was focused on moving the first steps in the **image processing** subject. First, the $Gaussian\ noise$ and $Salt\ \&\ Pepper$ one were $added\ to\ the\ original\ images$: displaying the image and plotting the histogram, it can be observed how the $Gaussian\ noise$ covers the image with random pixels, which just create a general distortion, while the $Salt\ \&\ Pepper$ one creates a more visible effect, since some random pixels are replaced with black or white points, and this is more evident to the human eye.
The second step had as objective the $application$ of the $principal\ filters$ to the noisy images and see the outcomes: it was possible to notice that the images with $Salt\ \&\ Pepper\ noise$ processed with the median filter resulted in a better output, meanwhile the images affected by the $Gaussian\ noise$ were better filtered using the average filter and the Low-Pass Gaussian filter, especially the latter.
The third step was focused on the implementation of $linear\ filters$ to $enhance$ the $details$, and it was possible to notice that it is better to be careful with the weight given to the details, since, if it is too high, the original colors are overwhelmed by the details ones, which are distorted.
The last part of the laboratory had as goal to see what happens when the $Fourier\ Transform$ is used on the image and some of the filters created: the remarkable result is the observation of the lines in the magnitude of the spectrum in frequency, linked to the geometrical patterns in the pictures. A possible $future\ development$ is to use the filters created in other projects such as edge or blob detection, in order to recognize the edges and the area of the objects in the images.