Image Analysis

Lab 3: Image filtering, edge detection, object detection

Student: TRAN Gia Quoc Bao, ASI

Professor: HENG UY Chhayarith

Date: 21 February 2020

Introduction: The objectives of this Lab are evaluating the performance of image denoising, edge detection and object detection by phase correlation.

Preparation before the lab:

A histogram is a graphical representation of intensity by pixel. It plots the number of pixels for each value of intensity. Clusters of intensities are grouped by the number of pixels with that intensity. For example, with an 8-bit image, we have 28 = 256 different intensity values so it would have 256 columns. Each column indicates the number of pixels with that intensity value. So this approach is statistical (frequentist).

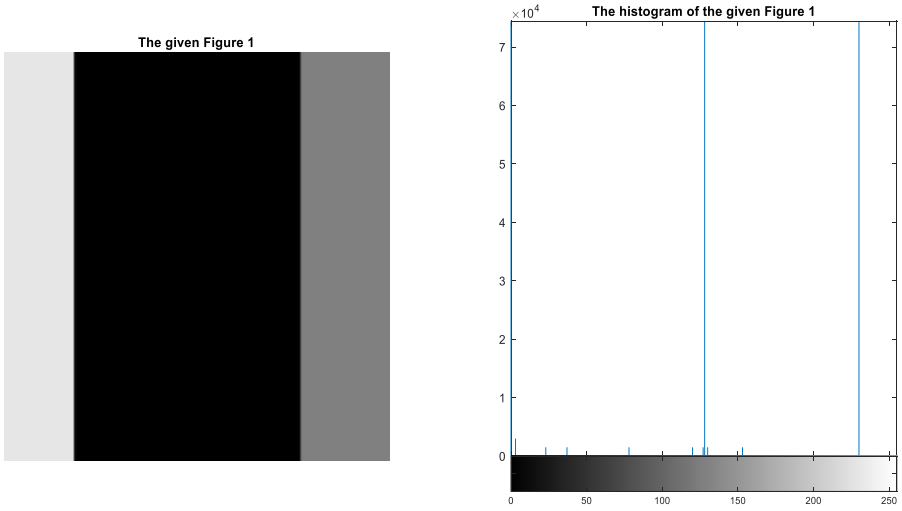

Figure 1. The sample image's histogram

The histogram has 3 tall columns corresponding to the 3 most popular intensities (0 and the other two values). Besides, we have some very small columns which represent the other values at the transitions or edges.

When we add some Gaussian noise with a standard deviation of 1:

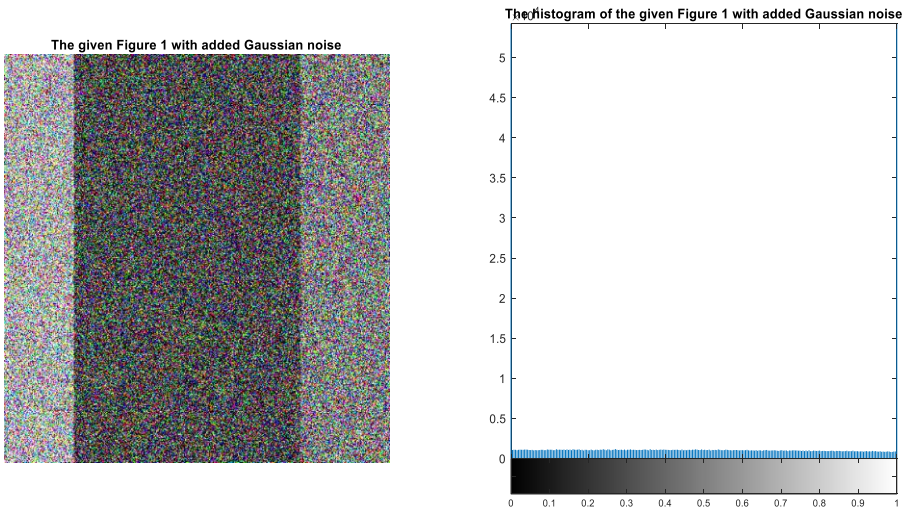

Figure 2. The image and its histogram after we add some Gaussian noise

The histogram has a lot of seemingly equal columns, as the effect of the noise.

When we add some salt & pepper noise:

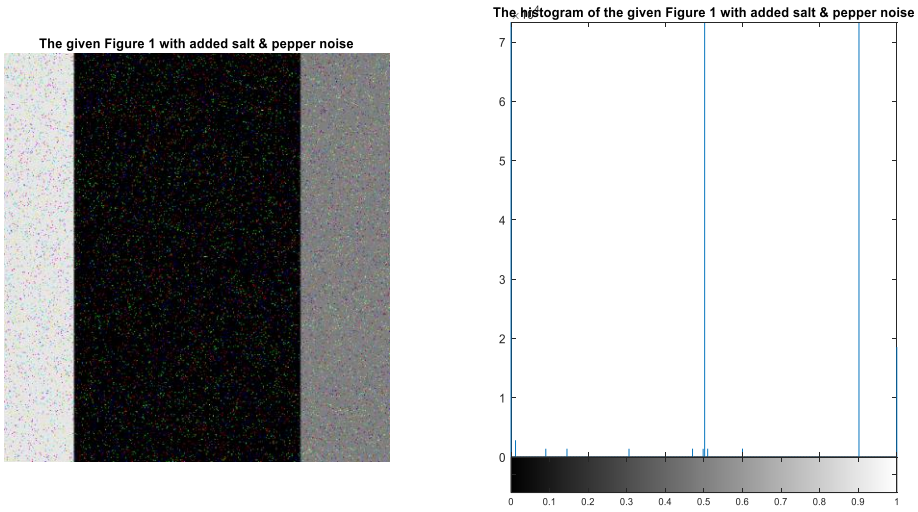

Figure 3. The image and its histogram after we add some salt & pepper noise

The histogram has the two extreme columns raised up as the result of the noise.

When we add some uniform noise with range 0.5 – 1:

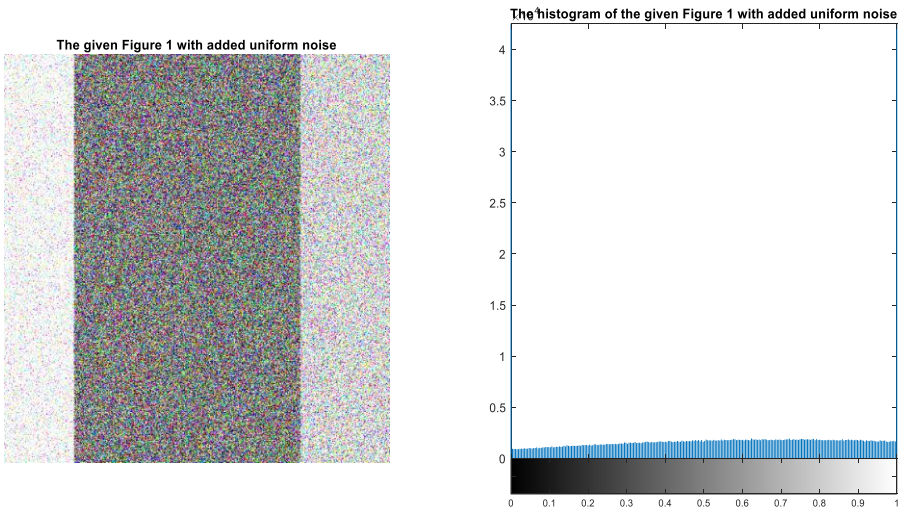

Figure 4. The image and its histogram after we add some uniform noise

The histogram has a lot of columns that are not equally tall.

To denoise these images, we can use filters. The two suggested methods are mean filtering and median filtering. Mean filtering, which uses a filter whose coefficients are the same, is the simplest among linear filtering methods. Median filtering takes the median value among the selected pixels.

Computer session:

I. Noise filtering:

1. Noise simulation:

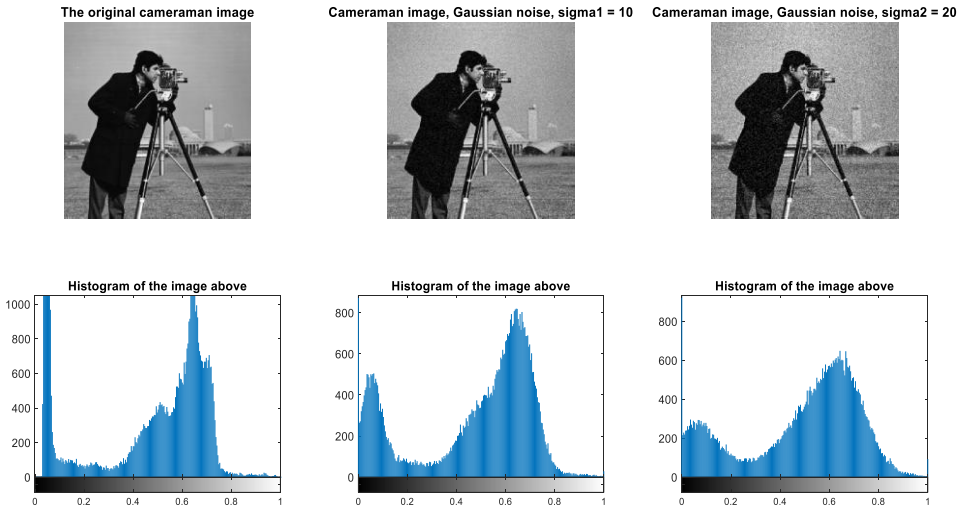

Figure 5. The cameraman image and its histogram with Gaussian noise

When we add Gaussian noise to an image, the histogram will be affected in all the intensities. As we used the normally distributed numbers (randn function), with a mean of 0, the number of intensities increased should be equal to the number of intensities decreased.

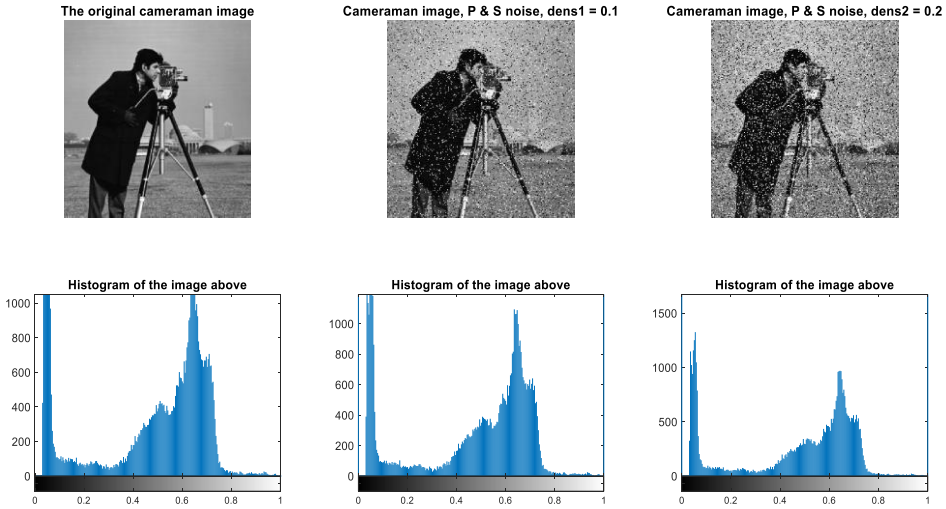

Figure 6. The cameraman image and its histogram with salt & pepper noise

For salt & pepper noise, we are adding a lot of white and black dots to the image, so we have an increase in the number of white and black pixels, not the middle ones. How covered the image is depends on our choice of density.

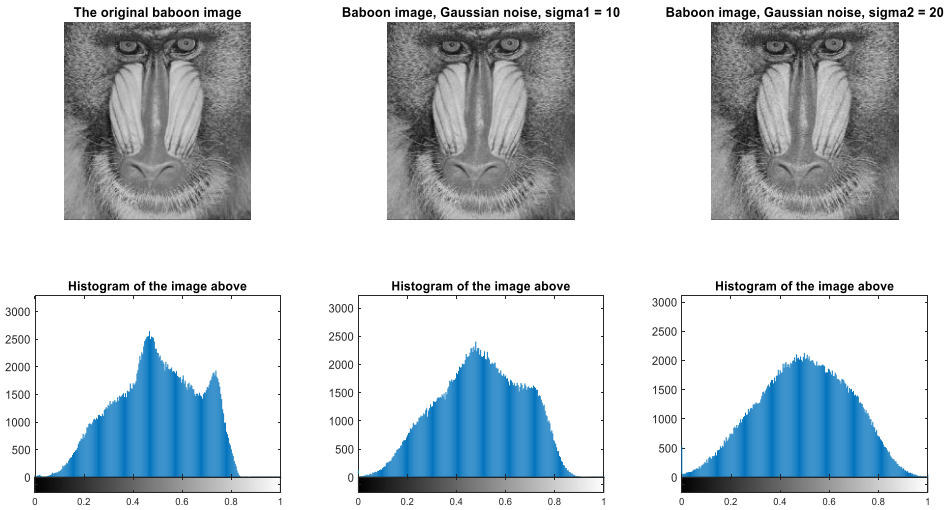

Figure 7. The baboon image and its histogram with Gaussian noise

With Gaussian noise, the histogram is redistributed and we see that there is less difference between the top and bottom values. The histogram seems smoother.

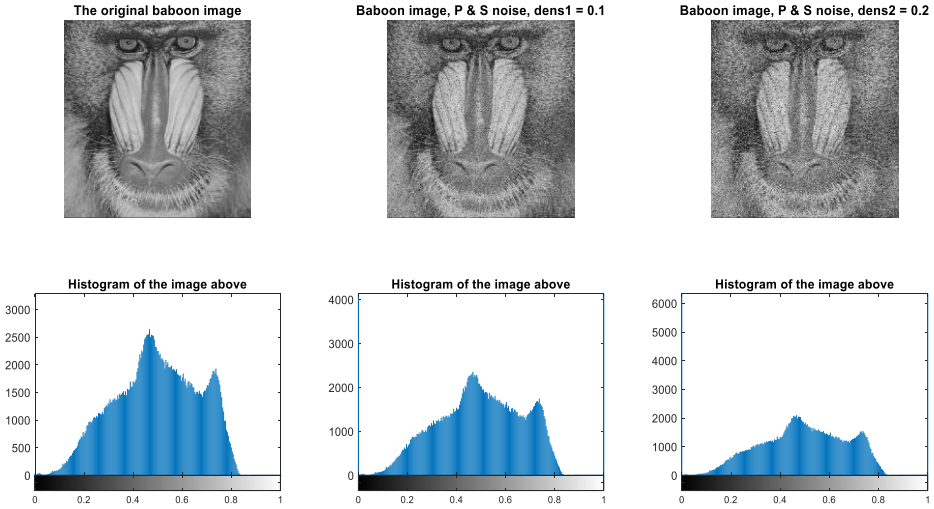

Figure 8. The baboon image and its histogram with salt & pepper noise

With salt and pepper noise, the middle range of the histogram is decreased (as they are replaced) while the 2 extreme values are lifted up.

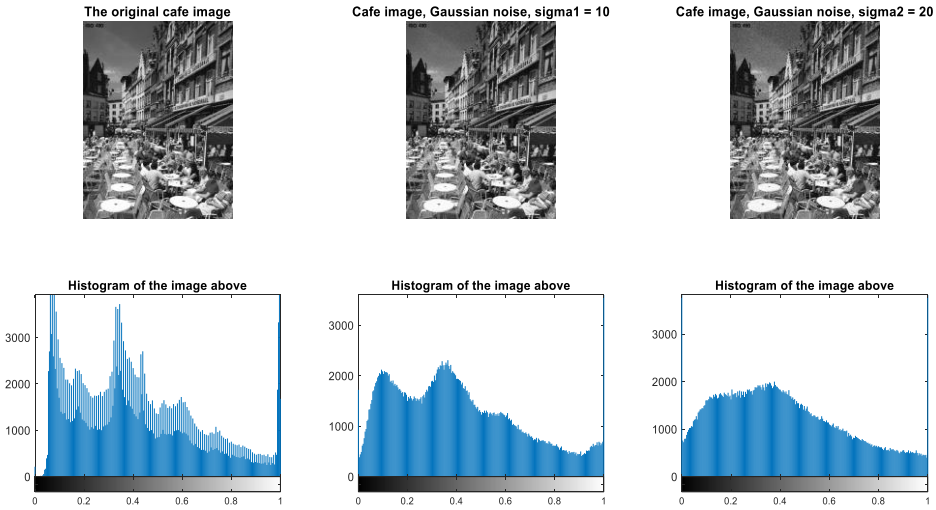

Figure 9. The café image and its histogram with Gaussian noise

Especially for this image, the great difference between the columns that lie next to each other is canceled by the Gaussian noise. When the noise has a high standard deviation (widely spread) we see no peaks in the histogram.

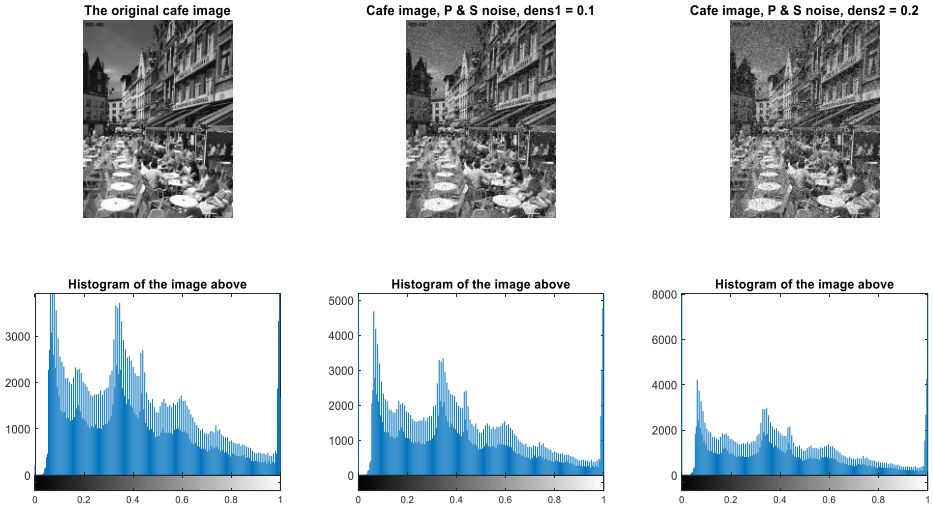

Figure 10. The café image and its histogram with salt & pepper noise

For the salt and pepper noise, the 2 sides get very high. Also, the difference between the columns is not canceled (as they all get shorter, and since is noise is random they should get shorter by the same amount).

2. Denoising:

In this part, I used 2 types of filters: the mean filter and the median filter.

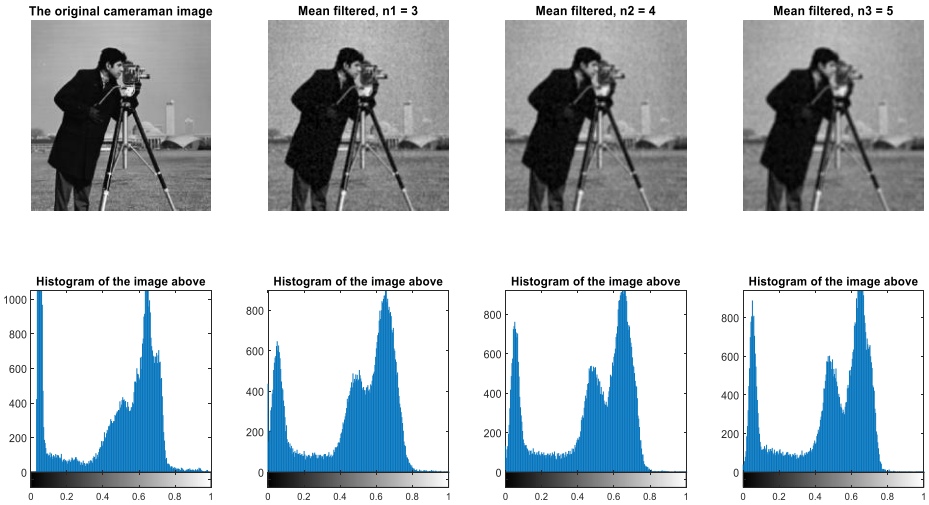

Figure 11. Cameraman image, Gaussian noise, mean filtered

For the mean filter, it uses the convolution between the image and a uniform matrix. For example with a 3x3 filter:

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

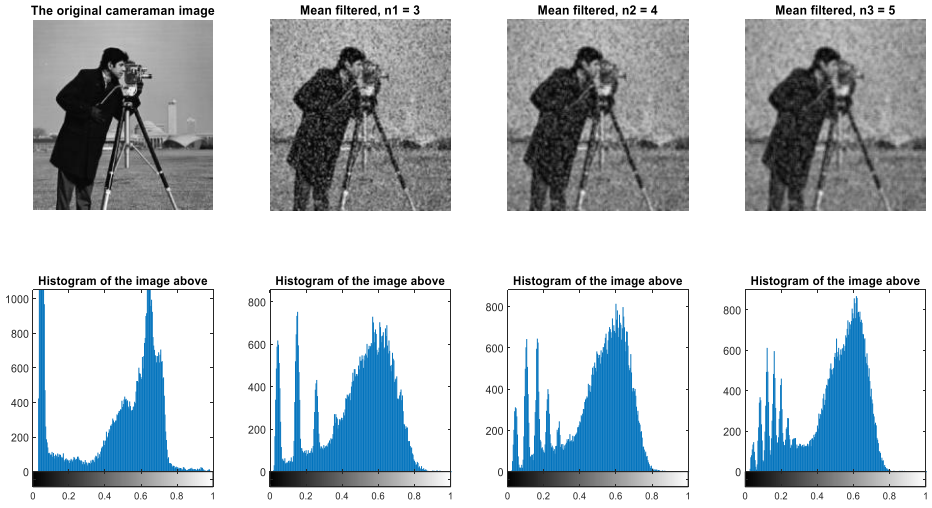

Figure 12. Cameraman image, salt & pepper noise, mean filtered

For the mean filter, this literally means that we sum the pixels and its neighbors, then divide by the total number of pixels. The result is by definition the mean of the values.

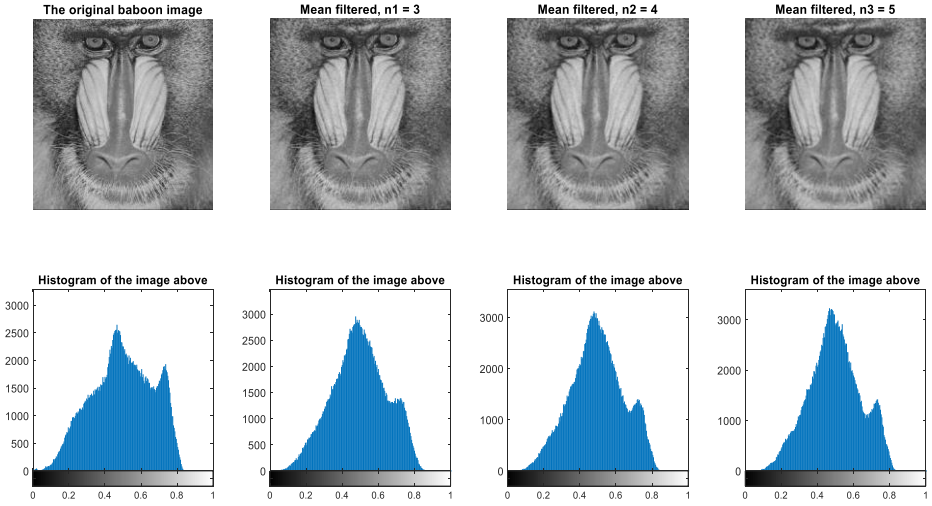

Figure 13. Baboon image, Gaussian noise, mean filtered

So this will reduce the difference between the near pixels. We see that this filter was effective as it was able to separate the intensities in the histograms.

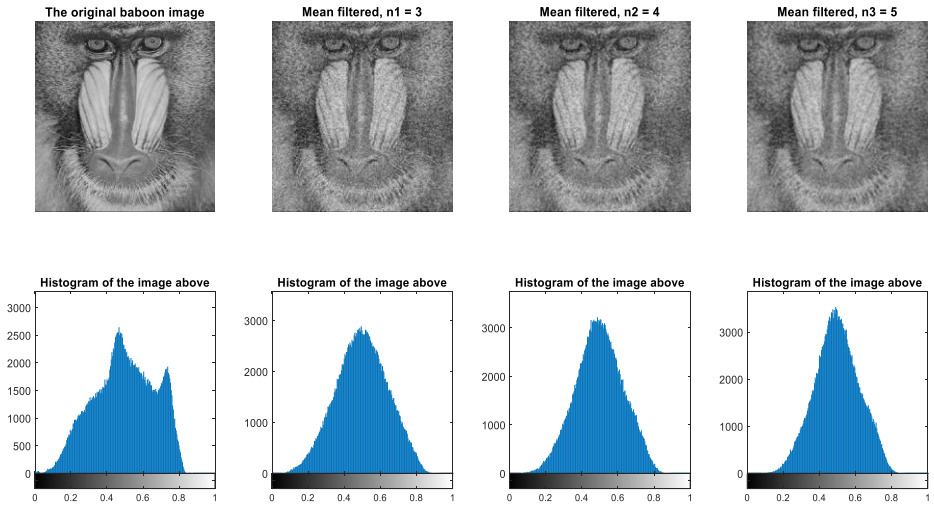

Figure 14. Baboon image, salt & pepper noise, mean filtered

I tried using 3 different mean filters with 3 different sizes: 3x3, 4x4 and 5x5 to compare their performance. The bigger the filter, the more pixels of medium intensity levels.

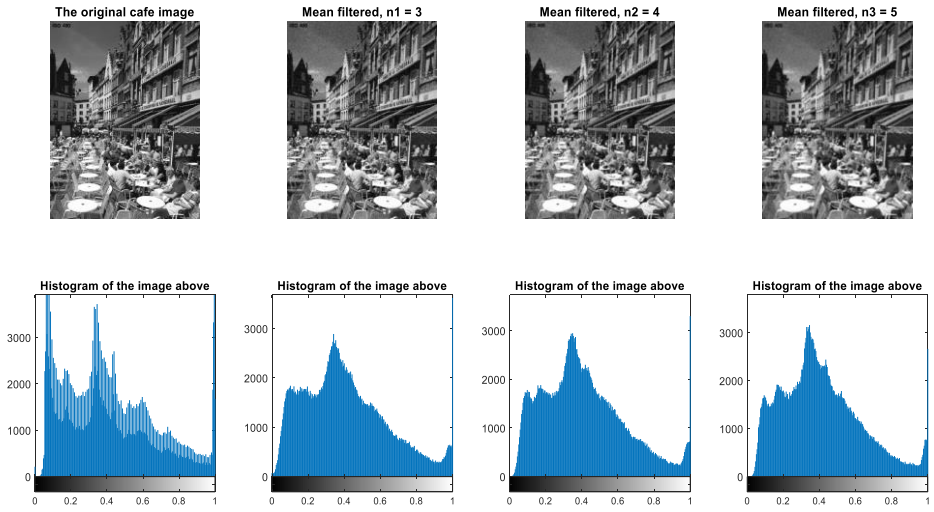

Figure 15. Café image, Gaussian noise, mean filtered

The reason is that when we have a big filter, we take more pixels into our calculations. Also, any single pixel can be taken a higher number of times as we move the filter across the image, as the size increased.

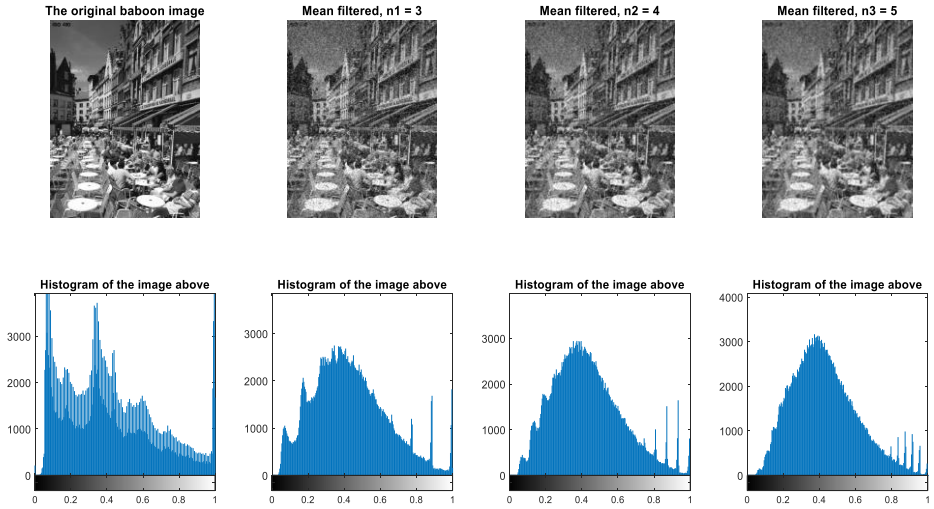

Figure 16.Café image, salt & pepper noise, mean filtered

My observation is that this filter seems effective for Gaussian noise, as the images look more similar to the original ones. And this is less effective for salt & pepper noise. The choice of filter size depends on the image: for the first 2 images, we do not have too many details, so a bigger filter seems better. But the last image has a lot of details, and a big filter may make it too blurred to see anything.

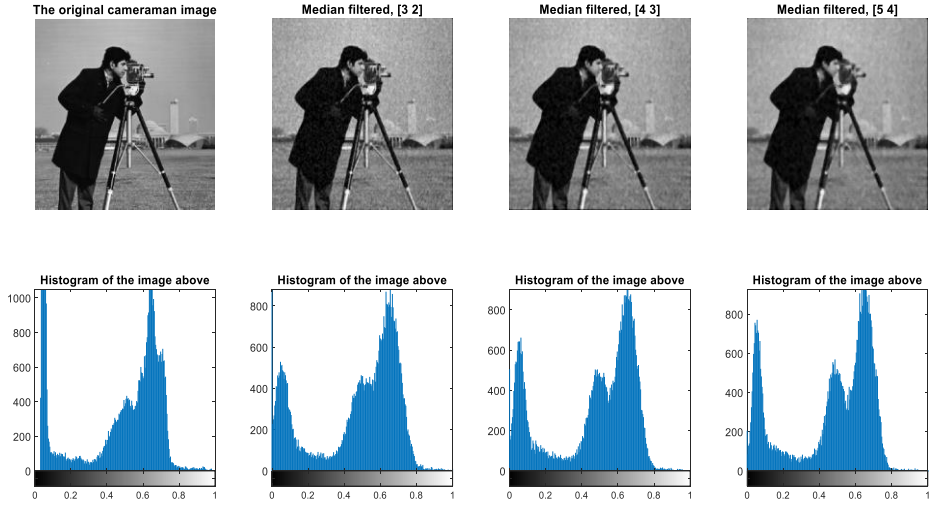

Figure 17. Cameraman image, Gaussian noise, median filtered

For the median filter, it takes the median of the values (by definition, after we put the values in ascending order, the median is the middle value in the case we have an odd number of values, and the average of the 2 middle values otherwise).

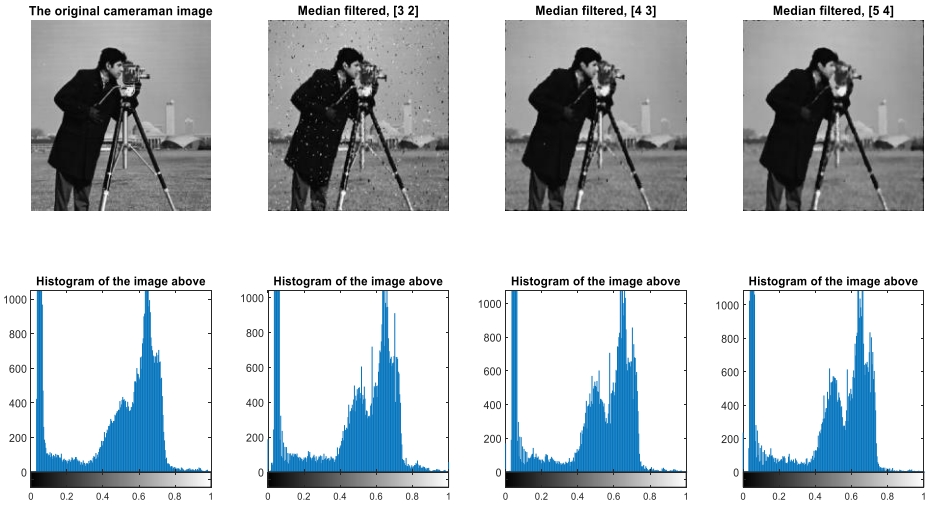

Figure 18. Cameraman image, salt & pepper noise, median filtered

So this will reduce the difference between the near pixels. We see that this filter was effective as it was able to separate the intensities in the histograms.

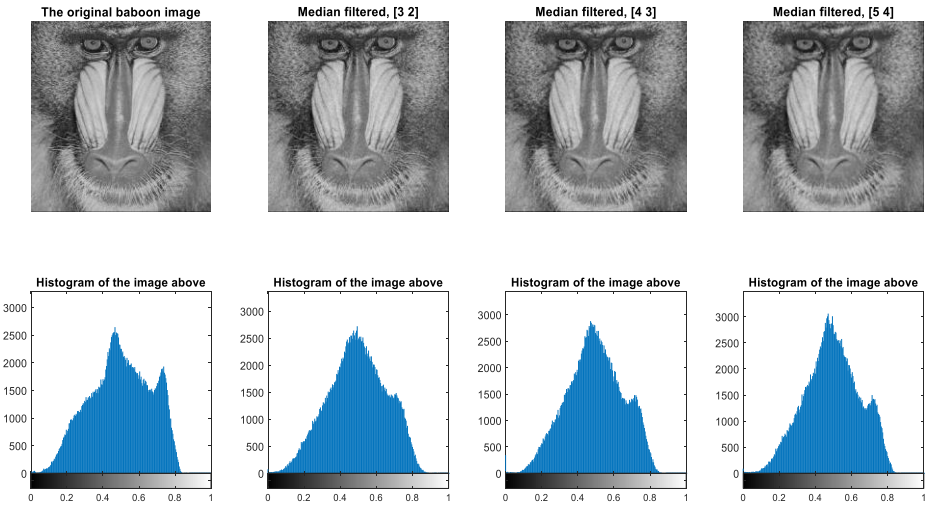

Figure 19. Baboon image, Gaussian noise, median filtered

I tried using 3 different median filters with 3 different sizes: 2x3, 3x4 and 4x5 to compare their performance. So each time I would take the median value among 6, 12, and 20 numbers.

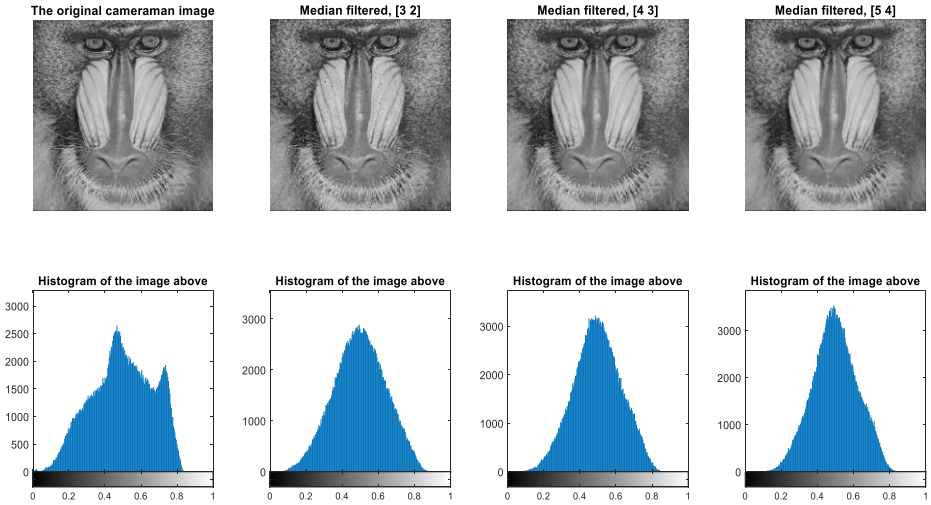

Figure 20. Baboon image, salt & pepper noise, median filtered

As we have an even number of pixel values, this corresponds to taking the average of the 2 middle intensities. The bigger the filter, the more pixels of medium intensity levels.

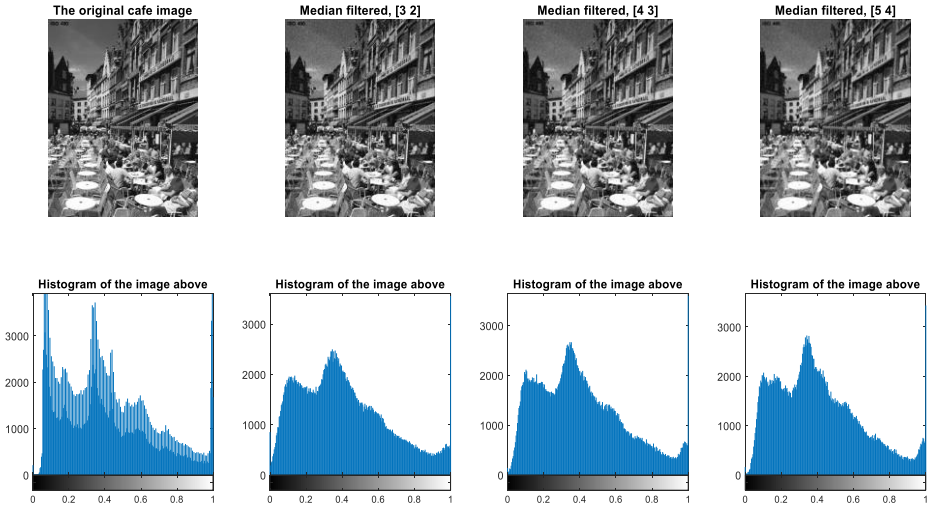

Figure 21. Café image, Gaussian noise, median filtered

The difference between this and the previous filter is that when we have a big filter, we take more pixels into our calculation, but the median will not vary much. Because we only care about the middle values when we take medians, not the small and large ones. In the previous filter, the average value must change accordingly when we move the filter.

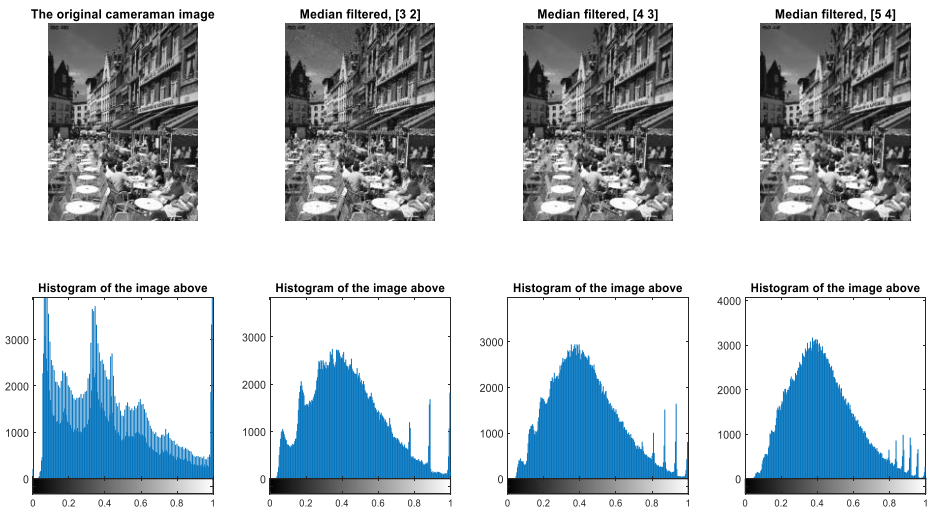

Figure 22. Café image, salt & pepper noise, median filtered

My observation is that this filter seems effective for salt & pepper noise, as the images look more similar to the original ones. And this is less effective for Gaussian noise. The choice of filter size depends on the image: the bigger the filter, the higher the middle intensities. This effect is the reverse of the effect caused by salt & pepper noise.

3. Quality assessment:

We cannot compare the sizes of the filters. We can only compare the types of filters.

I applied the filters like above onto the images and calculated the average Peak Signal-to-Noise Ratio (PSNR) defined by:

$$PSNR = 10 \log_{10} \left(\frac{d^2}{MSE(I_{in}, I_{den})} \right)$$

Where MSE means the mean square error between the original and denoised images, calculated by the function *immse*. When the PSNR is high, it means there is little difference between the 2 images as MSE is small, and so the images are very similar. We also consider the images' range of intensity values in the criterion, here d = 255 for an 8-bit image.

With the values obtained, I **observed** that: For Gaussian noise, a mean filter is better than a median filter (higher PSNR). For pepper noise, a median filter is better.

Explanation: the Gaussian noise has a mean of zero and does not have outliers, so a mean filter is better as it takes into account all the pixels. For the pepper noise, it's like we add the dots and black points to the image, so we have outliers instead of distribution. If we calculate the average it's not near to the gray color of the original image. If we take the median, it includes sorting the dots, then taking the middle value, and this value is close to the gray background color.

A weakness of this quality index is that it is not close to human perception. Which means the results given by it and what we see can have some difference.

II. Edge detection:

To find the edges in an image, we can use the **Sobel filter**. Gradients can be taken in each direction, for example in the vertical direction we have:

$$filterSobel_{verticalGradient} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Which will detect the horizontal changes or edges. This transpose of this, when applied to an image, will give us the vertical edges. The mean square of these two results is the gradient's magnitude.

First, I applied the Sobel filter on the images:

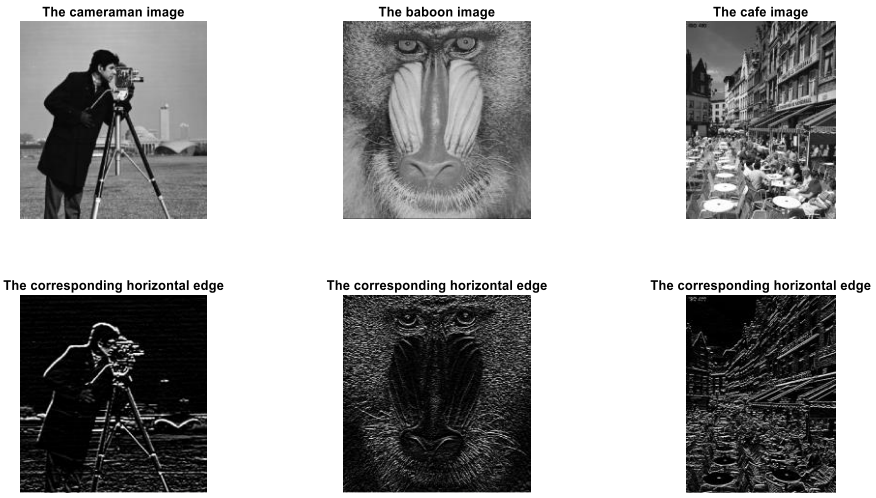

Figure 23. The 3 images Sobel filtered horizontally

We can see that the horizontal lines in each image are kept while the vertical ones are filtered. The clearest results can be seen in the last 2 images: the baboon image has a lot of vertical lines, so we can no longer see the baboon's face after filtering. The café image has a lot of straight lines, and we see that only the horizontal ones are kept.

Then this is the result when applying the transpose Sobel filter on the images:

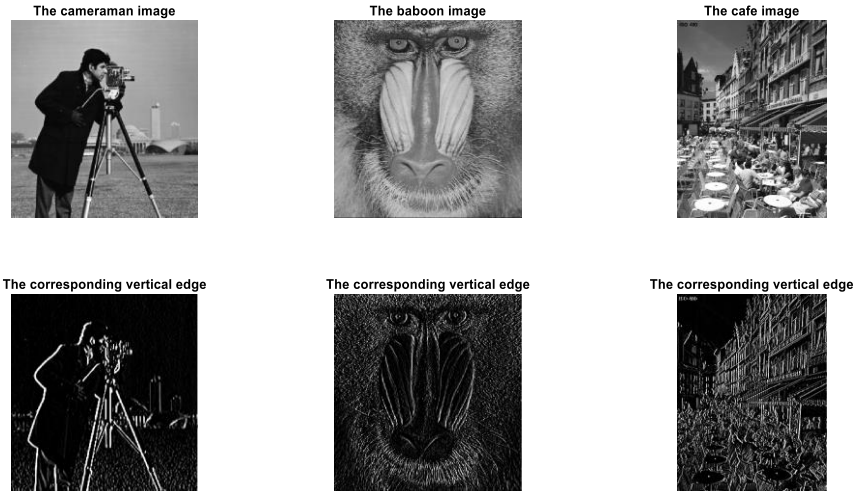

Figure 24. The 3 images Sobel filtered vertically

We see that this time the vertical edges are reserved. We can see the baboon's face and the vertical straight lines in the last image.

Then I summarized the results in both directions by calculating the mean square of the two:

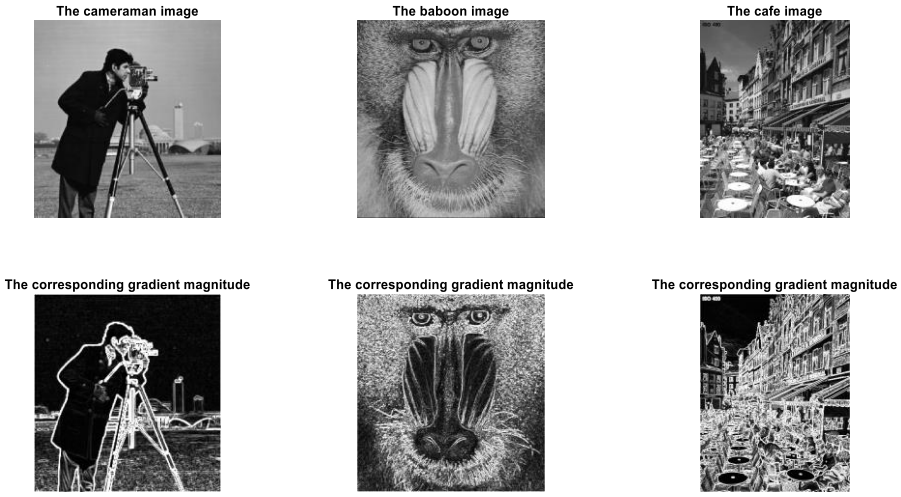

Figure 25. The 3 images' gradient's magnitudes (edges)

As we combined the horizontal and vertical edges, now we have all the edges. The clearest result is the cameraman image, where we form an edge around the man's body and the camera. So this method of using the Sobel filter can be used for **edge detection**.

III. Object detection by phase correlation:

We already know from the last Lab that the information contained in the phase of an image's DFT is the geometry in the image. So this operation called phase correlation:

$$R = F^{-1}\left(\frac{F(I_{in}) \times F^*(I_d)}{|F(I_{in}) \times F^*(I_d)|}\right)$$

Will compare the similarity in the phase part of an image's DFT with a model, reference one. As DFTs are complex numbers we can write them under the form:

$$F(I_i) = A_i e^{jP_i}, \quad i = 1, 2$$

In which A_i is the magnitude and P_i is the corresponding phase. **Mathematically**, we have:

$$\frac{F(I_1) \times F^*(I_2)}{|F(I_1) \times F^*(I_2)|} = \frac{A_1 e^{jP_1} \times A_2 e^{-jP_2}}{A_1 A_2} = e^{j(P_1 - P_2)}$$

So this matrix contains the **difference in phase** of the 2 images (phase correlation). The division by the magnitude is for the **normalization** of the result.

To apply this method to detect circles in the following Tokens image, I created a disk of radius similar to the circles (around 30 pixels). Then I shifted it to be coherent with the zero point of the token image:

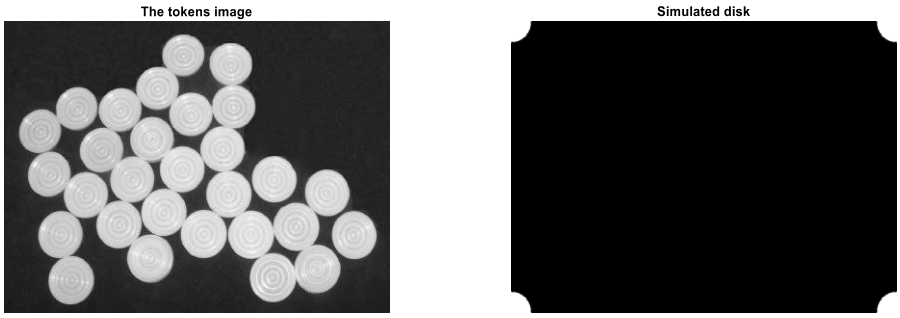

Figure 26. The Tokens image and the simulated disk

Then, I calculated the R using the formula above. To see the result clearer, I applied some thresholding on the R, at 0.015. What we have is the small dots at the centers of the circles, we can count about 29 dots 9they are not really dots, but groups of dots lying close to each other) which correspond to the 29 circles in the Tokens image.

Figure 27. Result of the phase correlation method

Then I compared the result with that given by the MATLAB built-in function *imfincircles*:

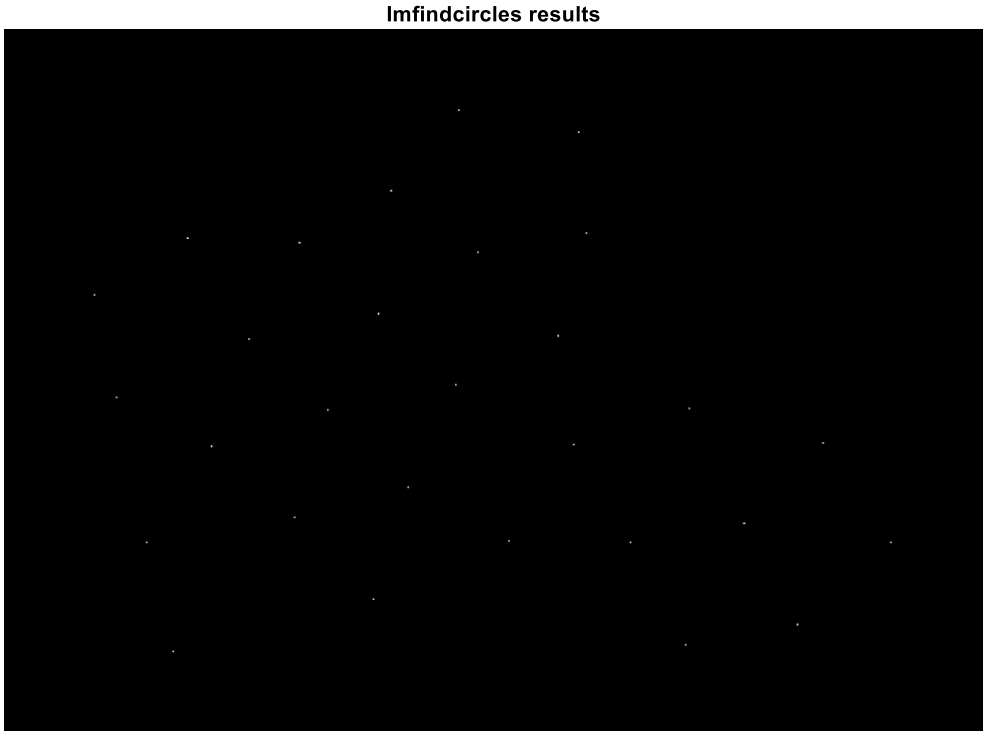

Figure 28. Result of the MATLAB imfincircles function

I had to zoom the image a little bit to see better the dots, as now the dots are in one pixel each. We see that the phase correlation was correct in comparing the images and was able to give us the locations of the centers.

Conclusion: After the Lab, I learned how to denoise an image, especially how to choose the filter effectively depending on the noise's nature. Then, I learned how to use the Sobel filter for edge detection. Lastly, objects in an image can be detected using the method of phase correlation.