Vision and Image Processing Assignment 1: filtering and edge detection

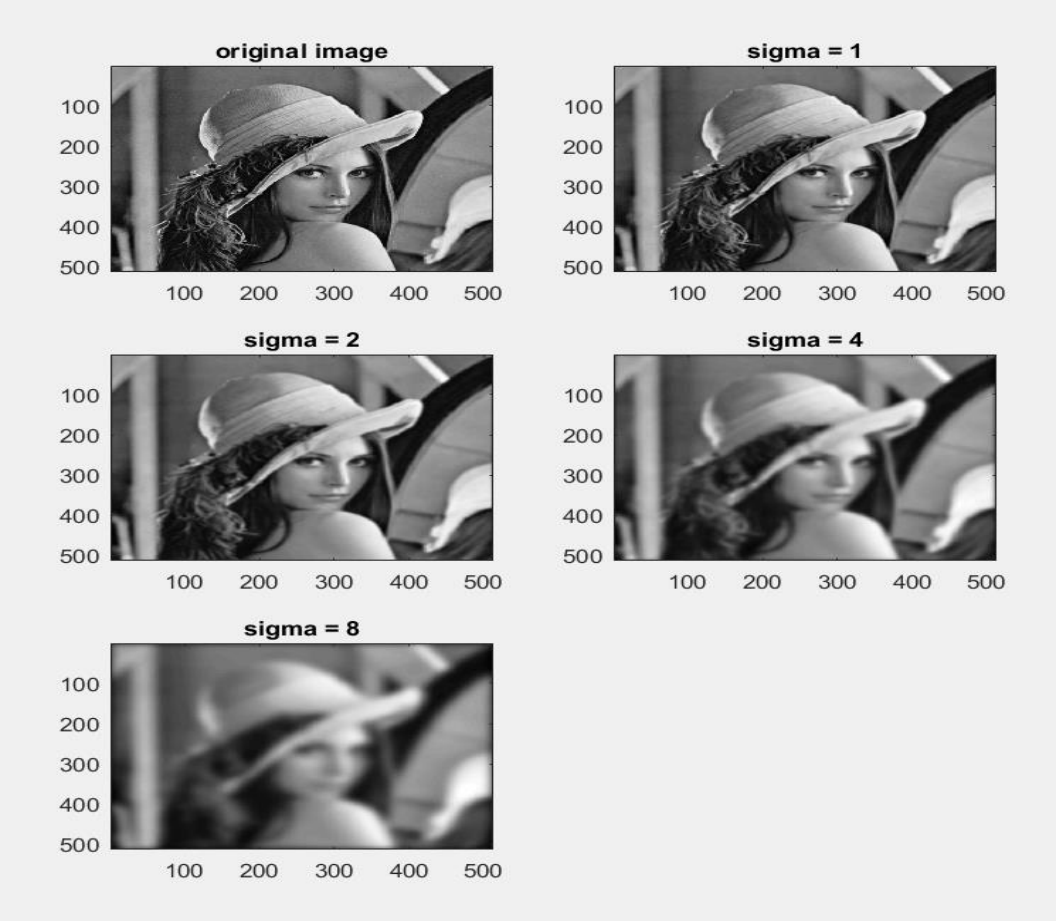
Mariuta Nicolae (rqt629) December 02, 2015

1. Gaussian filtering

For this exercise I have applied the Gaussian filtering in 2 ways:

- First, I did it by using Matlab functions for creating the Gaussian filter (by using the function *fspecial*) and then applied the filter by convolution using the function *imfilter* with the Lena image and the filter obtained previously.
- For the second part I have tried to implement filtering on my own: I have still created the filter by using the *fspecial* function of Matlab but then I used the convolution theorem shown at lecture that I get same results by transforming both matrices (the image and the filter) in Fourier domain and then multiply them. The only issues I had were because after making the IFFT the images were shifted and it took me a while to understand how to use the Matlab for shift and inverse shift but after that I have managed to obtained the same results as using the imfilter function of Matlab

I have applied the procedures as described previously with different parameters for σ and displayed the resulting images together with labels indicating the value of the parameter that was used. I obtained the following result:



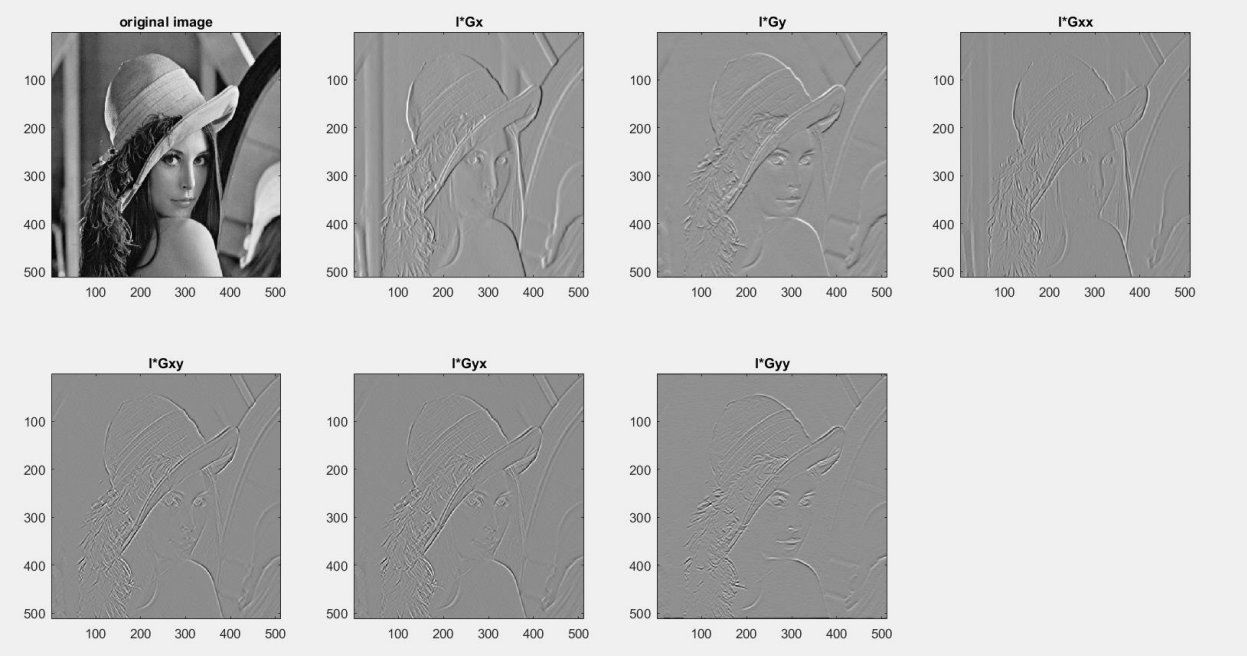
I can see that for very low values of parameter σ, there is not much change, the image is only smoothed a little bit but for values 4 and 8 of this parameter the image is very blurred. From these results I can understand that the noise is removed and some features of the image that would result in bad edge detection are also removed. For sigma 8 only few important edges are still visible: the contour of Lena and some other objects that are shown in the mage.

2. Gradient magnitude computation using Gaussian derivatives

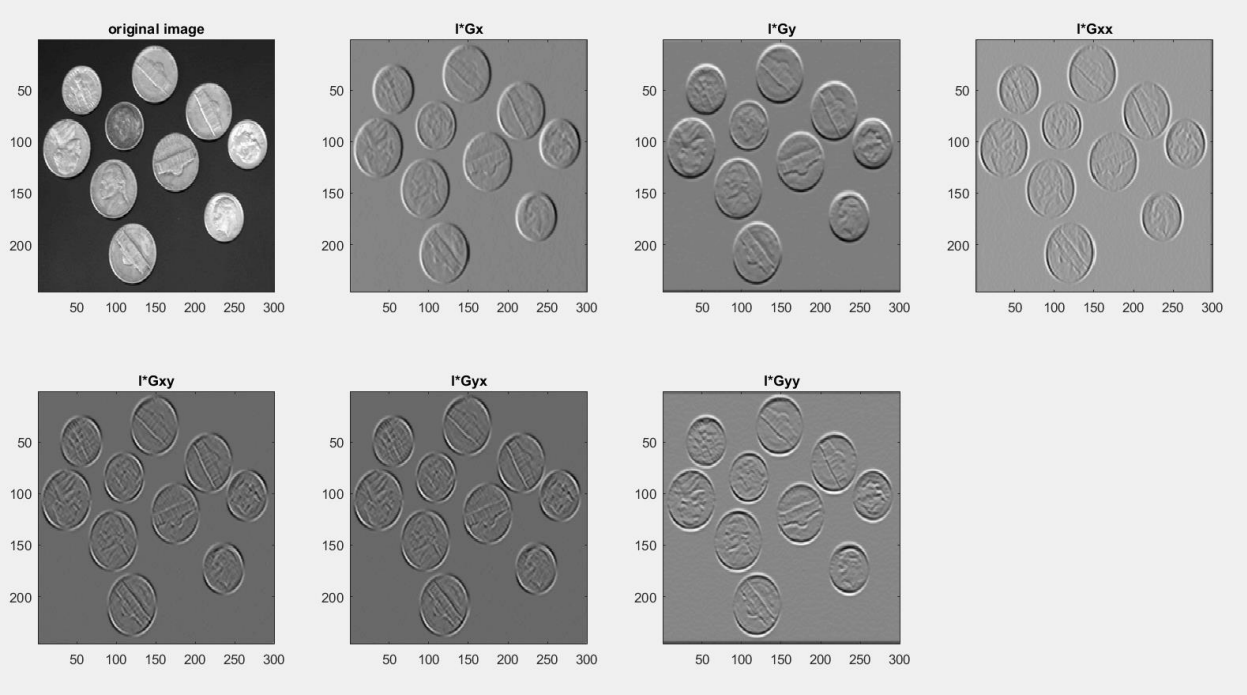
For this exercise I have only implemented the computation by using the Matlab functions for obtaining the Gaussian filters with all the values of sigma that are asked, I have then used the function *gradient* for computing the first order and second order derivatives. After obtaining these derivatives, I used the function *conv2* to make the convolution of the original image with each of the partial derivatives and displayed the results.

I did not have time to implement on my own but I think I could just compute the Gaussian derivatives or find the equations on Wikipedia and create the filters on my own for applying the convolution as I did for the previous exercise, by using the Convolution Theorem. I think this strategy would have led to longer computation times but maybe I could have understood better how to the derivatives computation works.

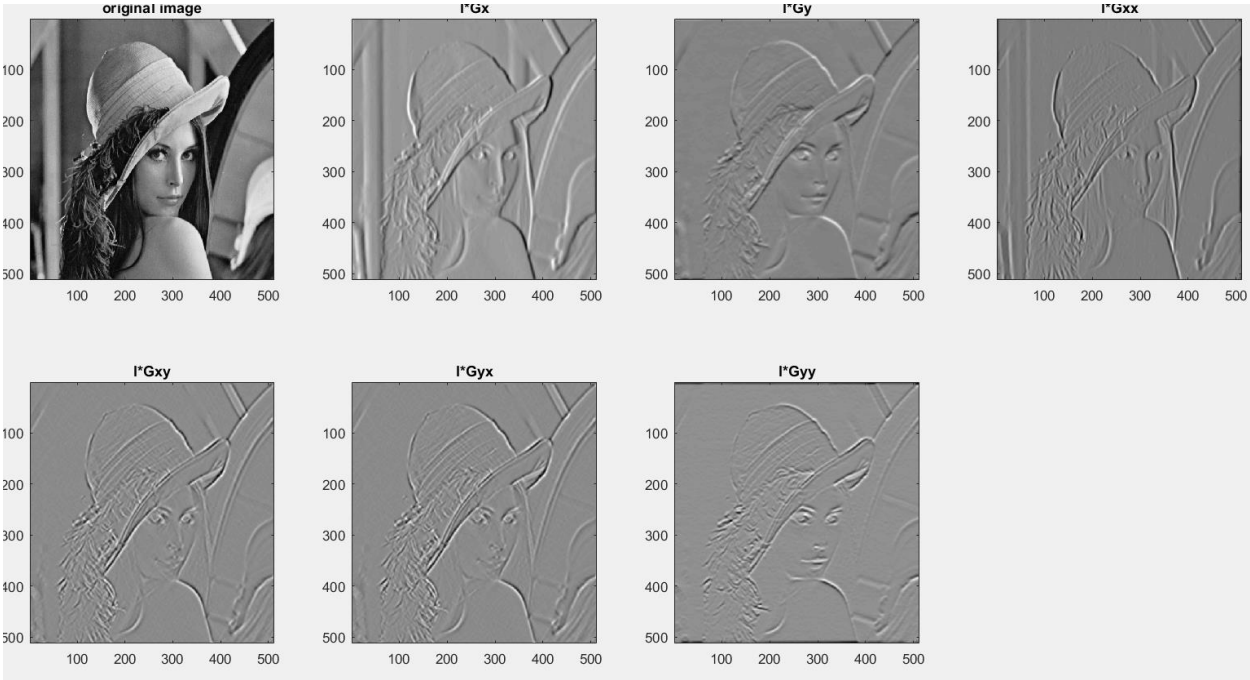
By running my code on the lena image with value of sigma 1 I have obtained the previous results:

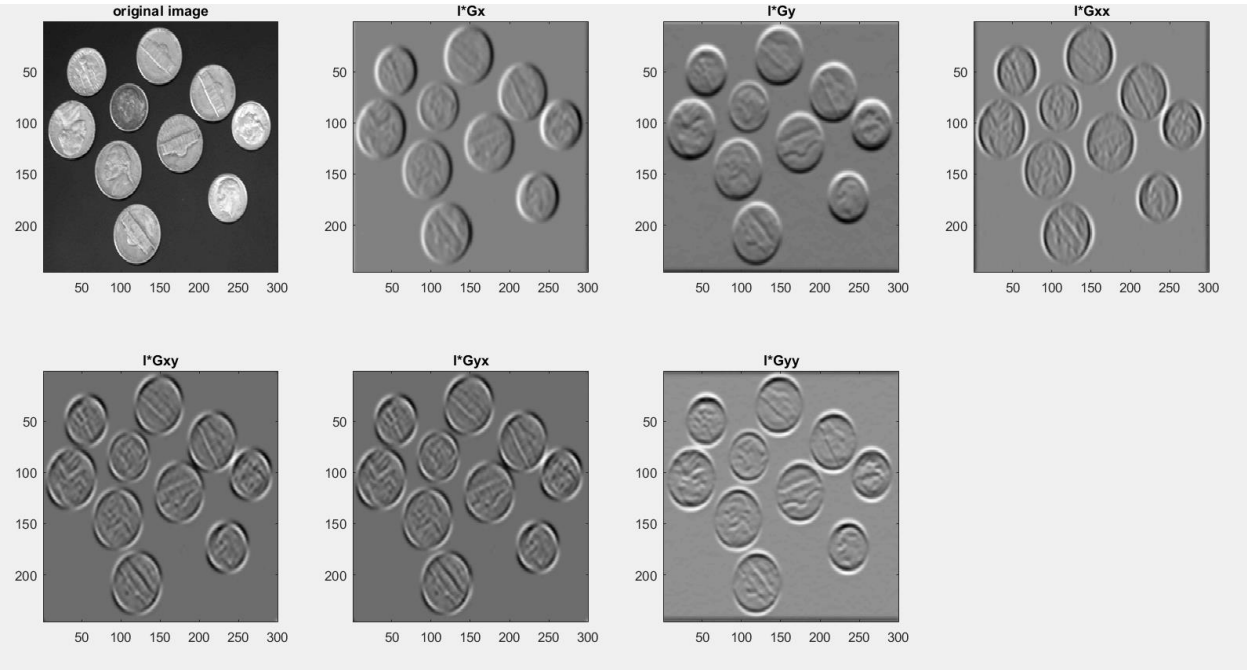


Since I found it is a very much used as example on the internet, I have decided to also use the coins image to detect the shape of the coins since these seem to have more practical use.

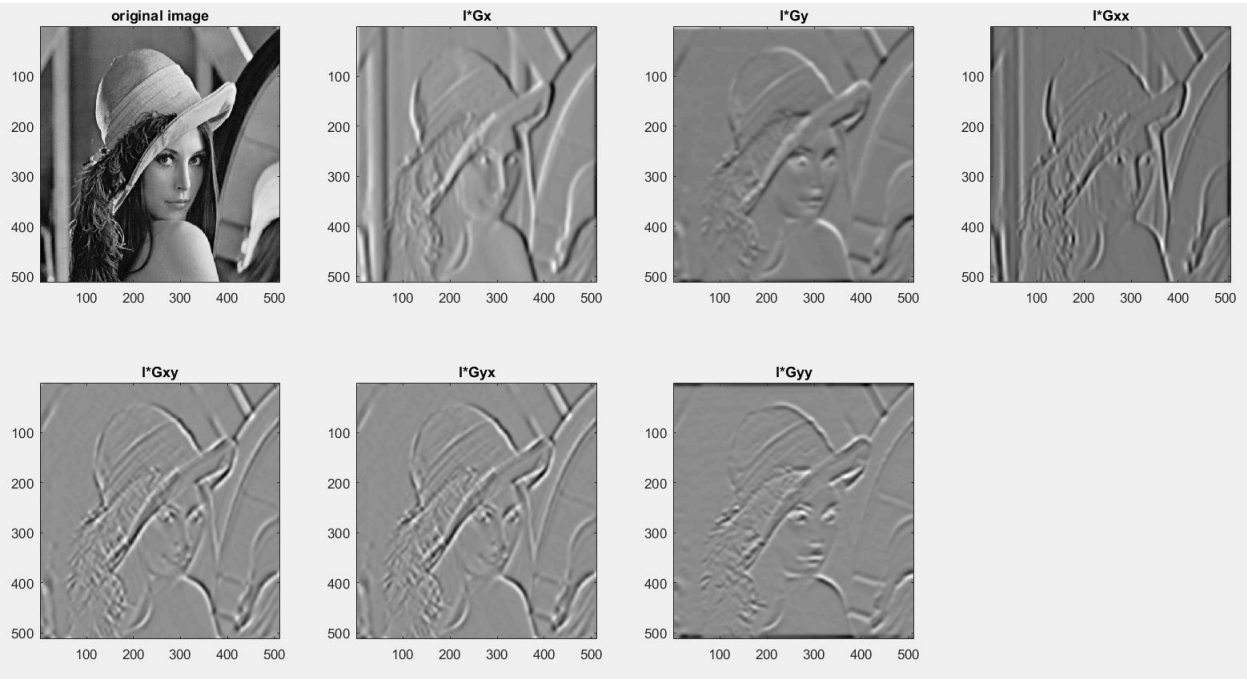


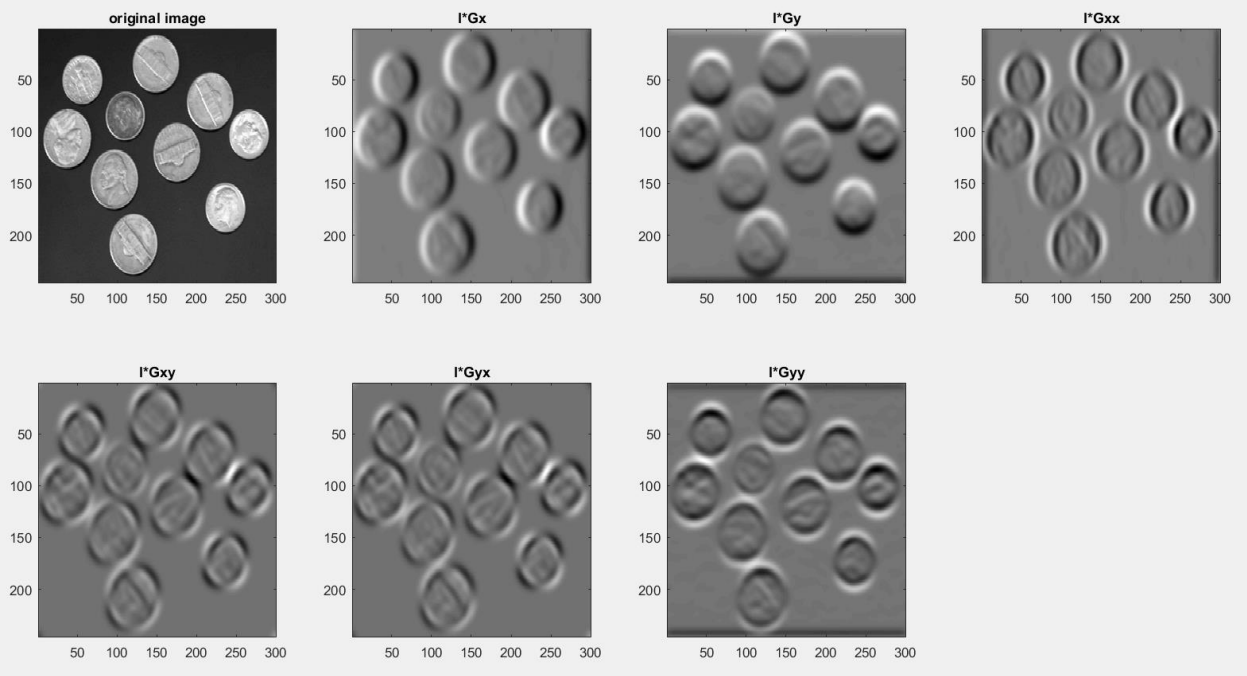
For the value of sigma 2 I have obtained:



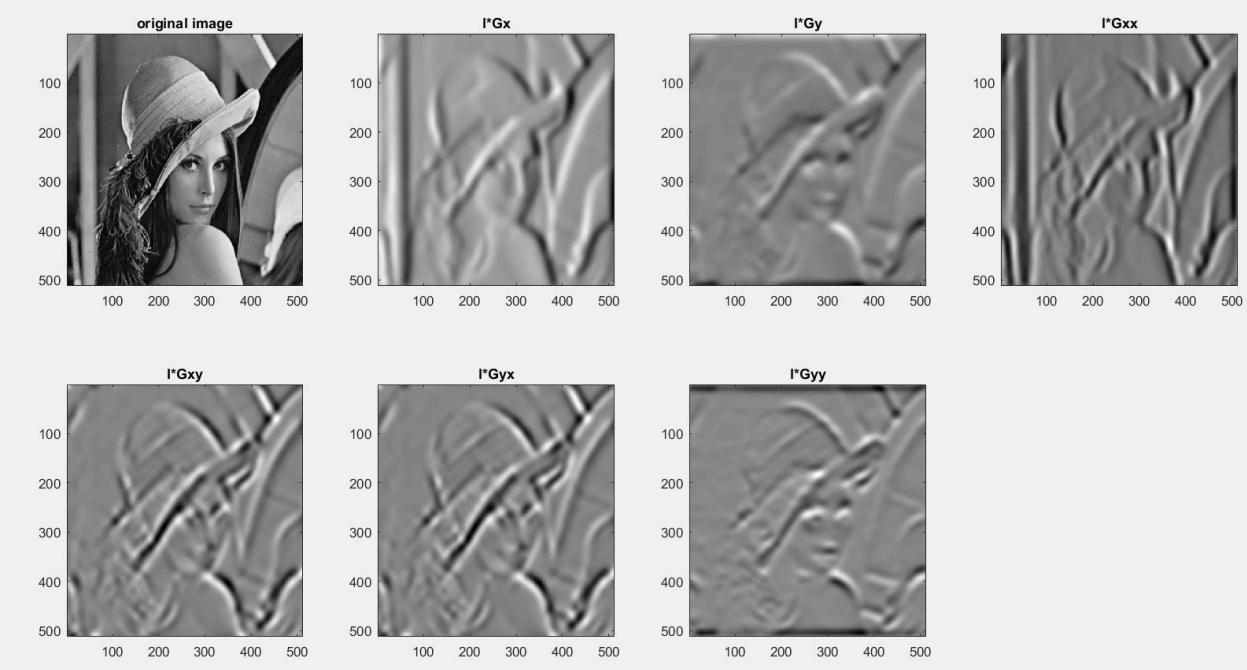

original image
I*Gx
I*Gy
I*Gxx
I*Gxy
I*Gyx
I*Gyy

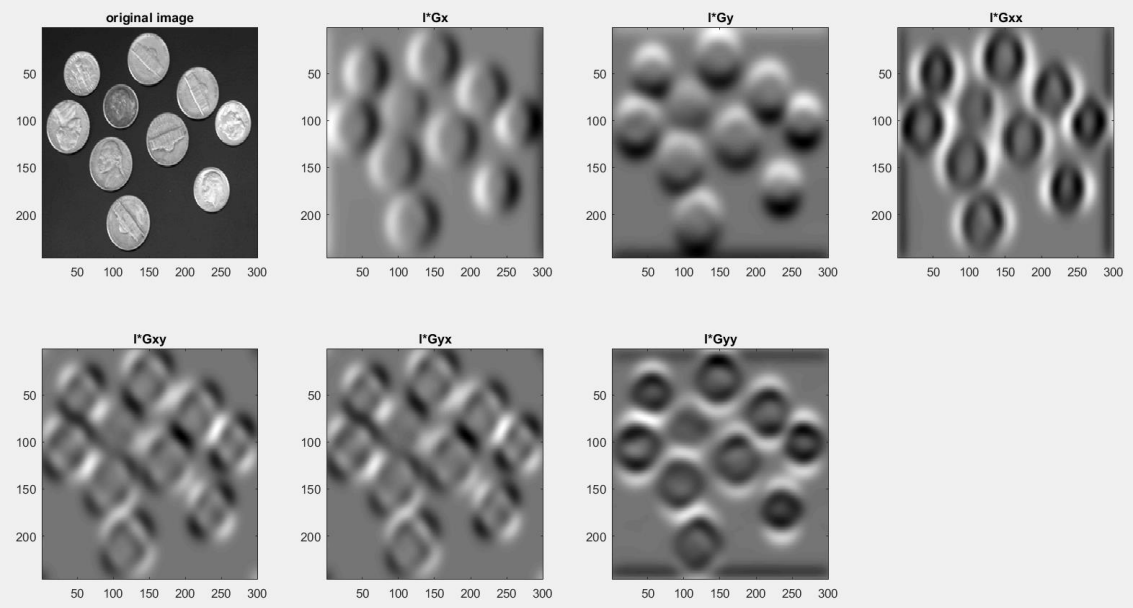
For sigma 4:


original image
I*Gx
I*Gy
I*Gxx
I*Gxy
I*Gyx
I*Gyy



And for sigma = 8:

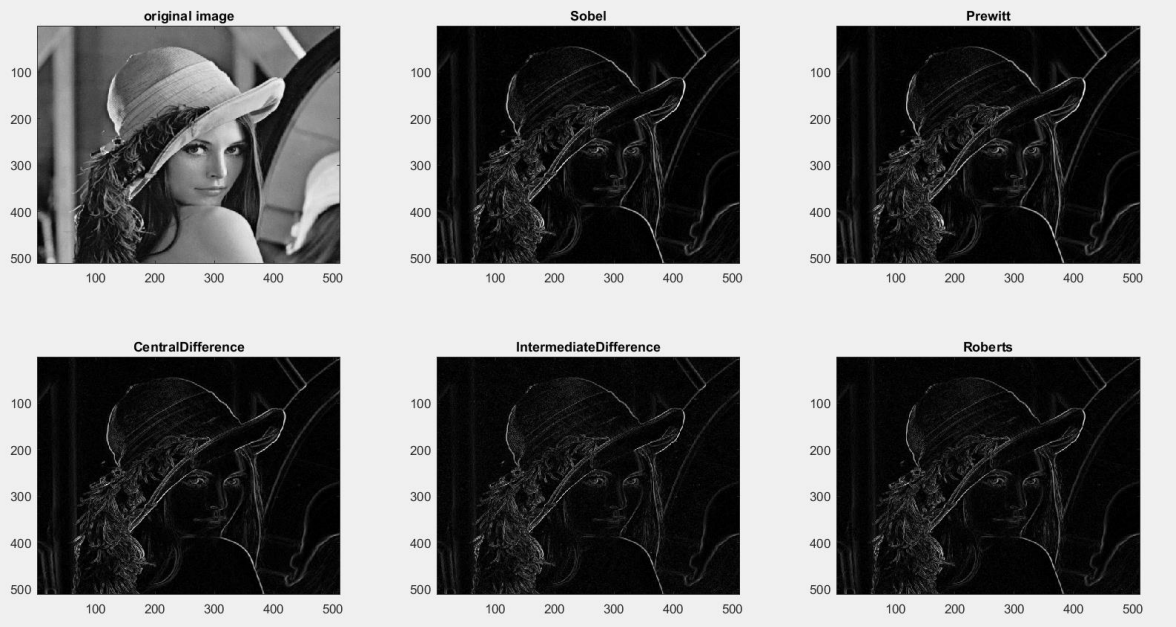




By viewing these results, it appears to be true what I have seen at previous exercise: for small value of deviation for Gaussian filter, more edges are visible in the gradient result while for the higher values of sigma more details disappear and only the shape of the main objects is still visible. For the coin with sigma 1 we can still see the drawing on the coins but for sigma 8 only the circular shape is visible. For the large value of sigma these edges are also thicker since they are smoothed by the Gaussian filter and I think this will obtain better results in future use of these edges for object recognition and more complex algorithms.

Also in the coins image is very visible by the shade of the coins edges the direction of gradient that the image is convolved with, for dx, the shade is on the left side while for dy the shade is on the lower side of the coins, For second order derivatives there are several shades of the coins on different sides according to the combination of directions that is computed.

Since they were mentioned in the lecture, I have also computed the gradients using different operators available for the Matlab *imgradient* function: Sobel, Prewitt, CentrailDifference, IntermediateDifference and Roberts. I have used Sobel operator previously but I not know much about the other oprators. However, I have obtained the following results for using these filters:



It does not seem to be a big difference between these methods of calculating gradient but they are certainly more different that for using the Gaussian derivatives. Maybe I should have used them with different parameters and images to get more information on how these work. These operators seem to detect the edges very well but there are many other details in the gradient like shades from the face and hat of Lena that may create problems for more complex applications that may need edge detection.

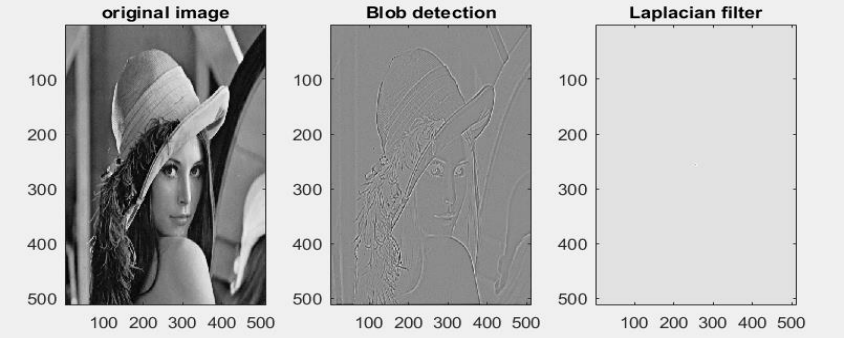
3. Mexican hat filtering

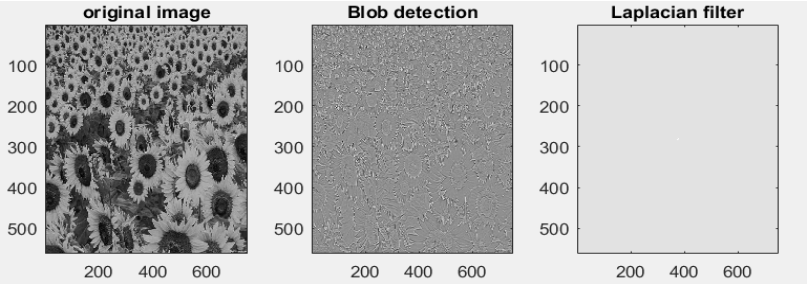
For the blob detection I have created the Laplacian on my own by using the final form of the equation that was shown at the lecture since I found it easy to understand and it is similar to what I could have also done for previous exercise to obtain the Gaussian derivatives.

I have first generated a mesh grid and shifted the coordinates so I could have (0,0) in the center of the image. Before doing this, I had a hard time to understand why I get weird results with the normal coordinates but when I plotted and saw the Laplacian in top-left corner, I understood why I had issues.

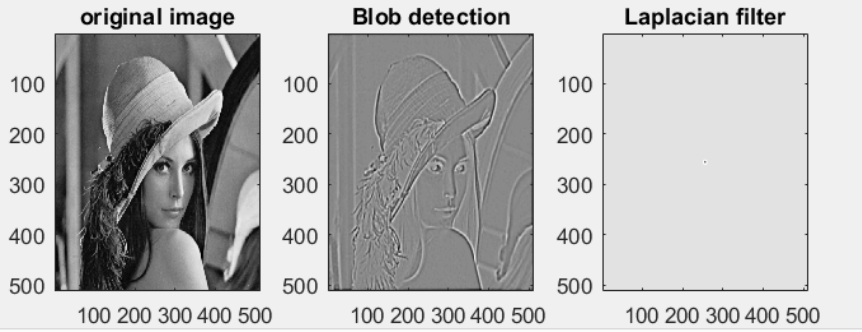
In the next step I have created the Laplacian with value of sigma that can be changed and I have convolved it with the image. I have tried lena image as suggested for the assignment but I also tried the sunflowers image that was used at the lecture since I thought that could obtain more meaningful results.

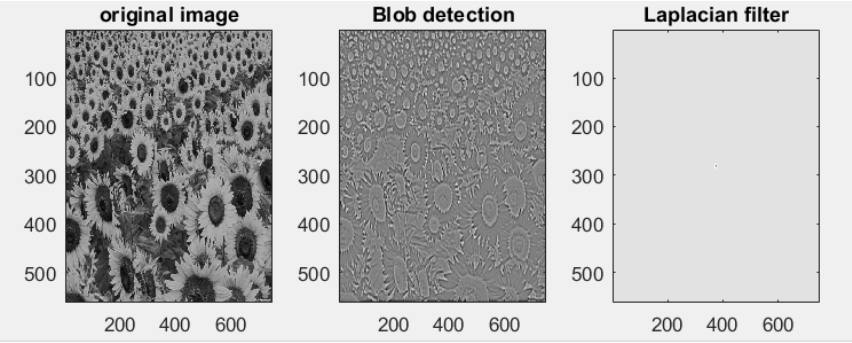
For sigma = 1, I have obtained:



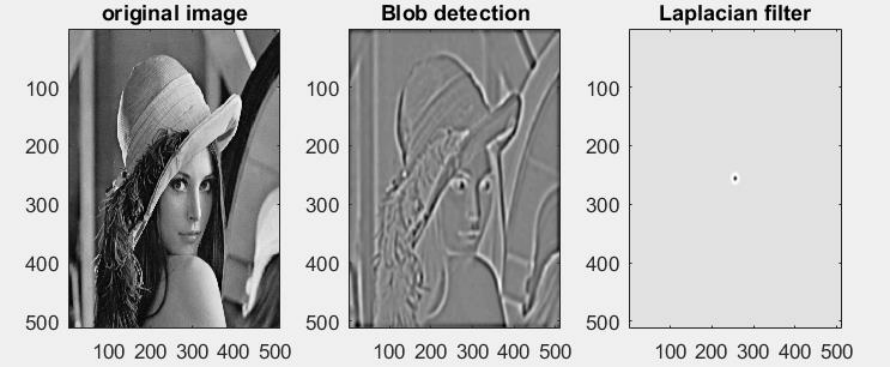


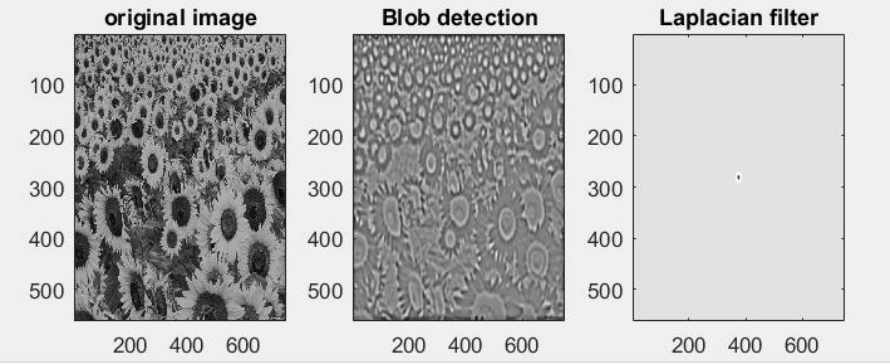
For sigma = 2:



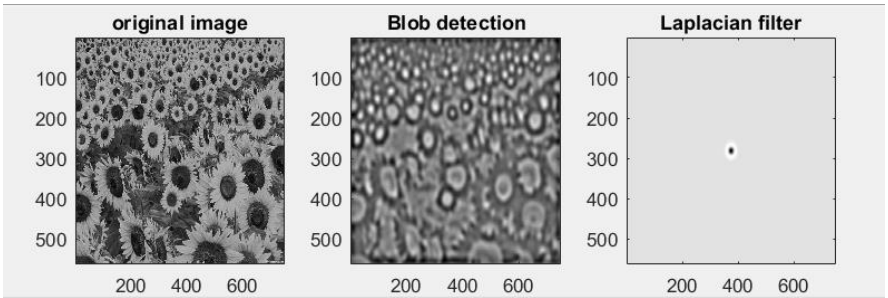


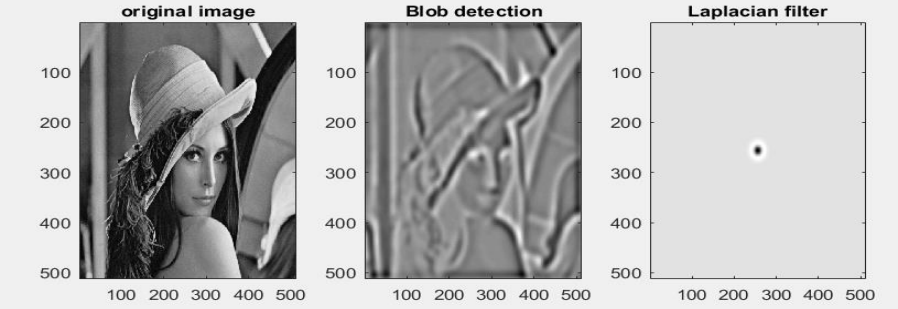
For sigma = 4


original image
Blob detection
Laplacian filter
100
200
300
400
500
100 200 300 400 500


original image
Blob detection
Laplacian filter
100
200
300
400
500
200 400 600

For sigma = 8:


original image
Blob detection
Laplacian filter
100
200
300
400
500
200 400 600


original image
Blob detection
Laplacian filter
100
200
300
400
500
100 200 300 400 500

It seems that even for Laplacian of a Gaussian filtering, we obtain noisy, thin and hard to distinguish edges for low values of sigma in both images. For large value of sigma, the edges important edges in the image are thick, easy to see and many unimportant details are removed.

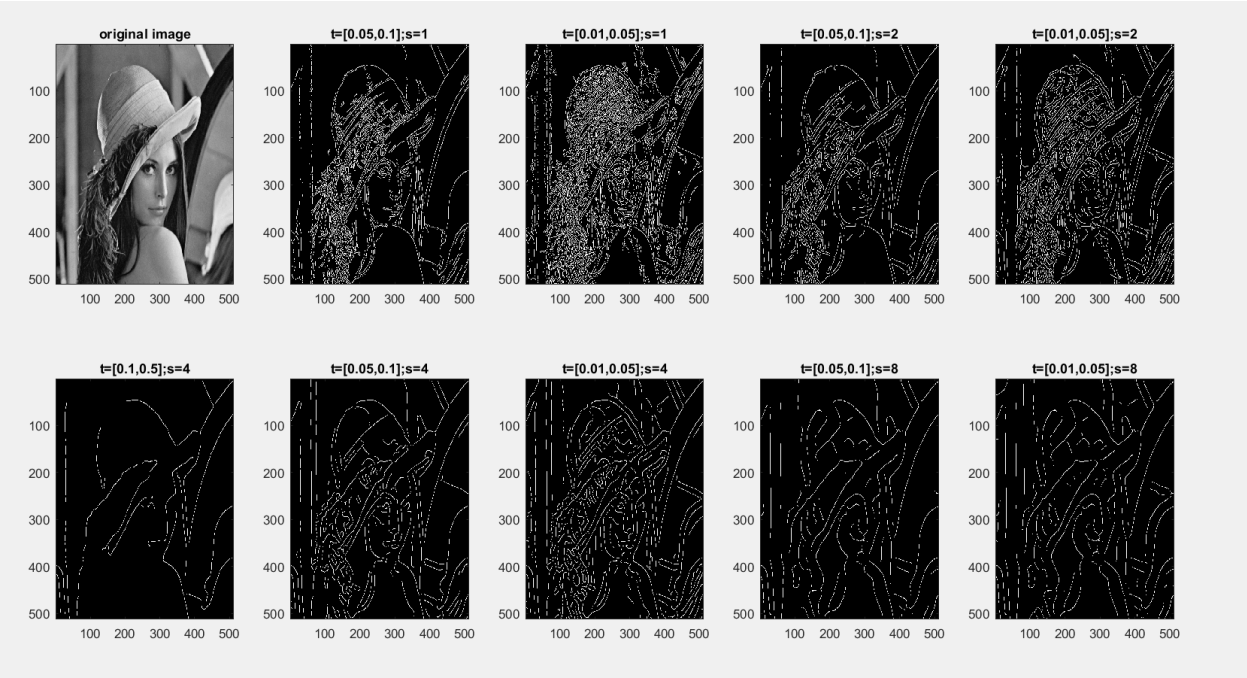
For the sunflowers image, we see most of the shapes from the original image in the result while for the sigma = 8 we only see the blobs, the shape of the sunflowers which can be used in more complex application like counting the sunflowers in the image. In case of lena we still see the important shapes as in case of the Gaussian derivative filtering and at a close comparison I noticed better results for using the Laplacian: many edges appear very clearly for using the Laplacian while those are quite interrupted for using the Gaussian derivatives.

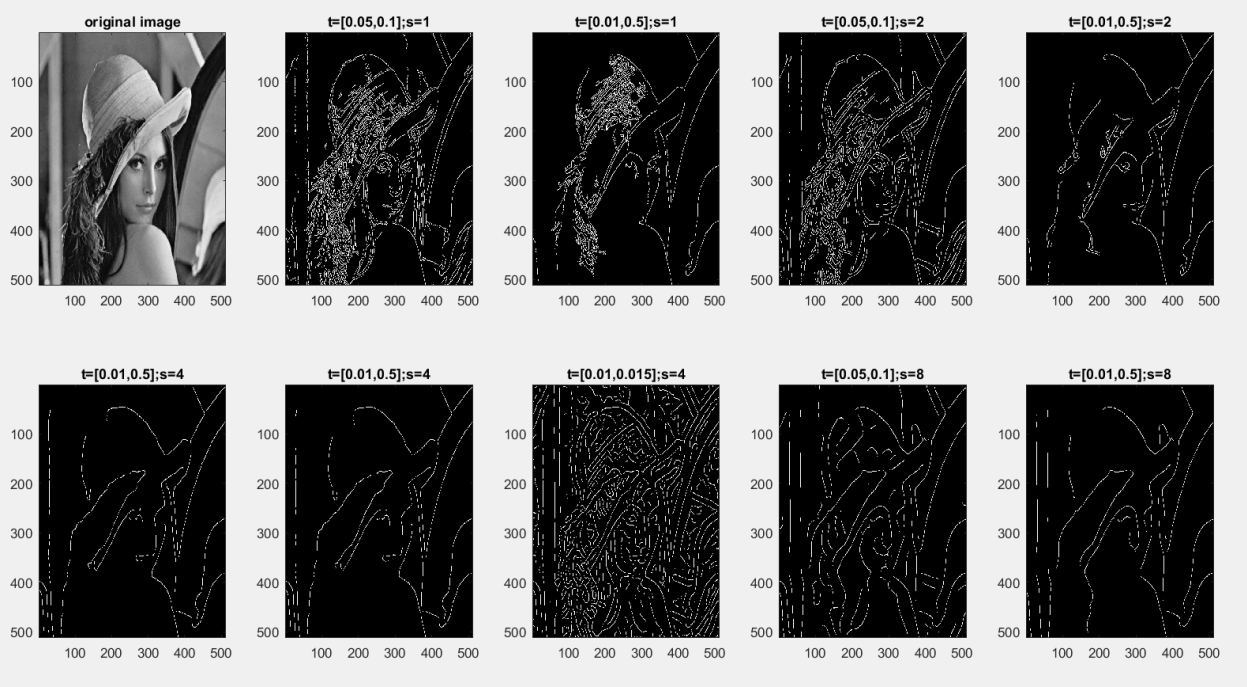
Because I had issues during the implementation and I have not used it before, I also shown the Laplacian for each parameter setting and for low value of sigma it is almost invisible but it appears as a dark circle for higher values of deviation.

4. Canny edge detection

For the canny edge detection, I first read on Wikipedia how it works and what are the main steps of the algorithm but unfortunately, I did not have enough time to also make an implementation so I used the Matlab function edge with the Canny operator and this function also allows me to set the low and high threshold and also the deviation of the Gaussian filter that is applied in the first step of the algorithm.

I have shown together the images for different parameter settings and I have shown in the title of each image first the values of thresholds and then the value of sigma. I have obtained the following results:





In comparison to the previous edge detectors, the first difference comes from the fact that the edge thickness is quite the same for all the edges that are found by the algorithm.

Just as before, for low values of deviation of the Gaussian filter, the edge detection is quite noisy and there are quite many useless details visible but the shape of Lena and many other details are quite easy to distinguish. However, I guess it will make issues is this result is needed for further processing of the image. For low value of sigma only the main shapes in the image are still visible.

However, in this case the threshold parameter has a big impact on the resulting edges, for low values of both thresholds more details are visible but the impact of this at higher values of sigma is diminished. There is also a big reduction in the number of edges that are detected for very high values of high threshold and low value of low threshold.

However, I find the most interesting the result for using the Canny edge detector with values 2 for sigma, thresholds 0.01 and 0.5 that just detects the shape of Lena and some shape of the mirror while other details that may not be useful are removed.

I can conclude that each edge detector that I have analyzed for this assignment may have its own applications: The Canny edge detectors obtains very thin edges without any thickness so it may not have enough information about the objects in the image which the Gaussian edge detectors obtain edges with different thicknesses that can be useful in applications but sometimes the edges are sometimes interrupted and contain quite much noise sometimes.