ELE510 Image Processing with robot vision: LAB, Exercise 4, Spatial-domain filtering

Purpose: *To learn about Linear Filters and Local Image Features and its use in computer vision (image processing). Some basic experiments will be implemented using Pyhon, OpenCV and other packages.*

The theory for this exercise can be found in chapter 5 of the text book [1]. See also the following documentations for help:

- OpenCV
- numpy
- matplotlib
- scipy

IMPORTANT: Read the text carefully before starting the work. In many cases it is necessary to do some preparations before you start the work on the computer. Read necessary theory and answer the theoretical part first. The theoretical and experimental part should be solved individually. The notebook must be approved by the lecturer or his assistant.

Approval:

The current notebook should be submitted on CANVAS as a single pdf file.

To export the notebook in a pdf format, goes to File -> Download as -> PDF via LaTeX (.pdf).

Note regarding the notebook: The theoretical questions can be answered directly on the notebook using a *Markdown* cell and LaTex commands (if relevant). In alternative, you can attach a scan (or an image) of the answer directly in the cell.

Possible ways to insert an image in the markdown cell:

`![image name]("image_path")`

`<img src="image_path" alt="Alt text" title="Title text" />`

Under you will find parts of the solution that is already programmed.

You have to fill out code everywhere it is indicated with `...`

The code section under `######## a)` is answering subproblem a) etc.

Problem 1

In this problem we want to get a better understanding of linear filtering using convolution.

The computations should be done by hand on paper until you are confident that you know how to do it.

Thereafter you can use the notebook to complete and check the results.

Sobel and **Prewitt** masks are used to compute the two components of the gradient. They perform differentiation over a 3 pixel region in the horizontal (x) and vertical (y) direction respectively and smooth by a 3 pixel smoothing filter in the other direction. The masks represent separable 2D filters and can thereby be separated in a differentiation filter and a smoothing filter.

The **Sobel masks**:

$$\mathbf{h}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{h}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{1}$$

The **Prewitt masks**:

$$\mathbf{h}_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{h}_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}. \tag{2}$$

a) Find the 1D **differentiation filter** and the 1D **smoothing filter** for the Sobel and Prewitt masks. The result will be similar for the x- and y-direction. It is therefore sufficient to find the result for one of the directions, e.g. the x-direction.

Consider the following image:

$$\mathbf{I}m = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3}$$

We can immediately see the differentiation and smoothing filters by looking at the masks.

Sobel - differrentiation filter is $[1, 0, -1]$ and smoothing filter is $[1, 2, 1]$

$$\mathbf{h}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{h}_y = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{4}$$

Prewitt - differrentiation filter is $[1, 0, -1]$ and smoothing filter is $[1, 1, 1]$

$$\mathbf{h}_x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{h}_y = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{5}$$

b) Filter this image using the **Prewitt** masks. Find the two output images, representing the differential along the horizontal and vertical directions.

▶ **Click here for an optional hint**

Prewitt mask for x-direction

$$\mathbf{h}_{x_{flipped}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{6}$$

$$\mathbf{h}_{x_{flipped}} \circledast \mathbf{I}m = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \circledast \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dots \tag{7}$$

$$I_x = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 2 & 1 & -1 & -2 \\ 2 & 1 & -1 & -2 \\ 1 & 0 & 0 & -1 \end{bmatrix} \tag{8}$$

Prewitt mask for y-direction

$$\mathbf{h}_{y_{flipped}} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{9}$$

$$\mathbf{h}_{y_{flipped}} \circledast \mathbf{I}m = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \circledast \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dots \tag{10}$$

$$I_y = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ -1 & -2 & -2 & -1 \end{bmatrix} \tag{11}$$

c) Filter this image using the **Sobel** masks. Find the two output images, representing the differential along the horizontal and vertical directions.

▶ **Click here for an optional hint**

Sobel mask for x-direction

$$\mathbf{h}_{x_{flipped}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{12}$$

$$\mathbf{h}_{x_{flipped}} \circledast \mathbf{I}m = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circledast \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dots \tag{13}$$

$$I_x = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 3 & 2 & -4 & -3 \\ 1 & 0 & -2 & -3 \\ -1 & -3 & 1 & -1 \end{bmatrix} \tag{14}$$

Sobel mask for y-direction

$$\mathbf{h}_{y_{flipped}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{15}$$

$$\mathbf{h}_{y_{flipped}} \circledast \mathbf{I}m = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \circledast \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dots \tag{16}$$

$$I_y = \begin{bmatrix} 1 & 3 & 3 & 1 & \\ -1 & 2 & 2 & -1 & \\ 1 & -2 & -2 & 1 & \\ -1 & -3 & -3 & -1 & \end{bmatrix} \tag{17}$$

d) Compute the gradient, $|\nabla I| = \|\nabla I\| = \sqrt{I_x^2(m,n) + I_y^2(m,n)}$, images based on the **Prewitt** and **Sobel** masks.

In [ ]:
```python
import numpy as np
import pprint

prewitt_I_x = np.array([[1, 0, 0, -1], [2, 1, -1, -2], [2, 1, -1, -2], [1, 0, 0, -1]])
prewitt_I_y = np.array([[1, 2, 2, 1], [0, 1, 1, 0], [0, -1, -1, 0], [-1, -2, -2, -1]])

gradient_prewitt = np.sqrt(prewitt_I_x**2 + prewitt_I_y**2)

sobel_I_x = np.array([[1, -1, 1, -1], [3, 2, -4, -3], [1, 0, -2, -3], [-1, -3, 1, -1]])
sobel_I_y = np.array([[1, 3, 3, 1], [-1, 2, 2, -1], [1, -2, -2, 1], [-1, -3, -3, -1]])

gradient_sobel = np.sqrt(sobel_I_x**2 + sobel_I_y**2)

print("Prewitt gradient matrix:")
pprint.pprint(gradient_prewitt)

print("Sobel gradient matrix:")
pprint.pprint(gradient_sobel)
```

```
Prewitt gradient matrix:
array([[1.41421356, 2.        , 2.        , 1.41421356],
       [2.        , 1.41421356, 1.41421356, 2.        ],
       [2.        , 1.41421356, 1.41421356, 2.        ],
       [1.41421356, 2.        , 2.        , 1.41421356]])
Sobel gradient matrix:
array([[1.41421356, 3.16227766, 3.16227766, 1.41421356],
       [3.16227766, 2.82842712, 4.47213595, 3.16227766],
       [1.41421356, 2.        , 2.82842712, 3.16227766],
       [1.41421356, 4.24264069, 3.16227766, 1.41421356]])
```

e) How will you interpret the results with respect to edges in the test image?

Problem 2

Given a test image with black background (gray level 0), and a white rectangle (gray level value 1), of size 6×8 pixels in the center. Use the notebook to create a matrix representing this image.

Let the test image be of size 10×12.

▶ **Click here for an optional hint**

Use the notebook to do the necessary computations in the following questions.

Use the Prewitt masks:

$$\mathbf{h}_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{h}_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}. \tag{18}$$

for the computation of the differentials, $\frac{\partial I}{\partial x} = I_x$ and $\frac{\partial I}{\partial y} = I_y$ respectively.

a) Compute and sketch the gradient of the test image using the 2-norm for the magnitude. Use $|\nabla I| = \|\nabla I\| = \sqrt{I_x^2(m,n) + I_y^2(m,n)}$. Show all relevant pixel values in the magnitude gradient image.

▶ **Click here for an optional hint**

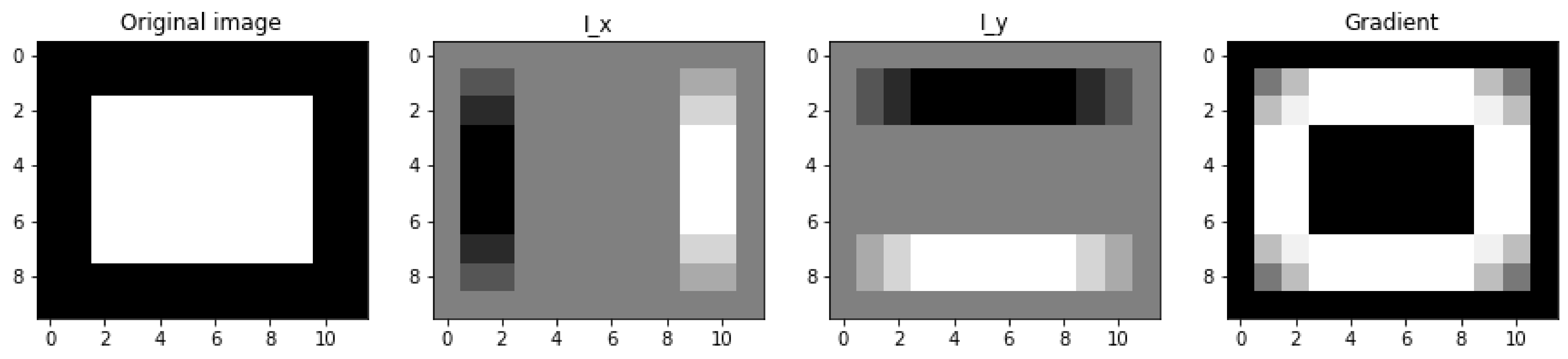
In [ ]:
```python
# Import useful packages
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [ ]:
```python
# Answer goes here
Rect, I = np.ones((6, 8)), np.zeros((10, 12))
I[2:8, 2:10] = Rect

# Define the kernel
kernelx, kernely = np.array([[1,0,-1],[1,0,-1],[1,0,-1]]), np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
I_x, I_y = cv2.filter2D(I, -1, kernelx), cv2.filter2D(I, -1, kernely)
gradient_i = np.sqrt(I_x**2 + I_y**2)

fig, ax = plt.subplots(1, 4, figsize=(15, 10))
ax[0].imshow(I, cmap='gray')
ax[0].set_title('Original image')
ax[1].imshow(I_x, cmap='gray')
ax[1].set_title('I_x')
ax[2].imshow(I_y, cmap='gray')
ax[2].set_title('I_y')
ax[3].imshow(gradient_i, cmap='gray')
ax[3].set_title('Gradient')
```

Out[ ]:
```
Text(0.5, 1.0, 'Gradient')
```

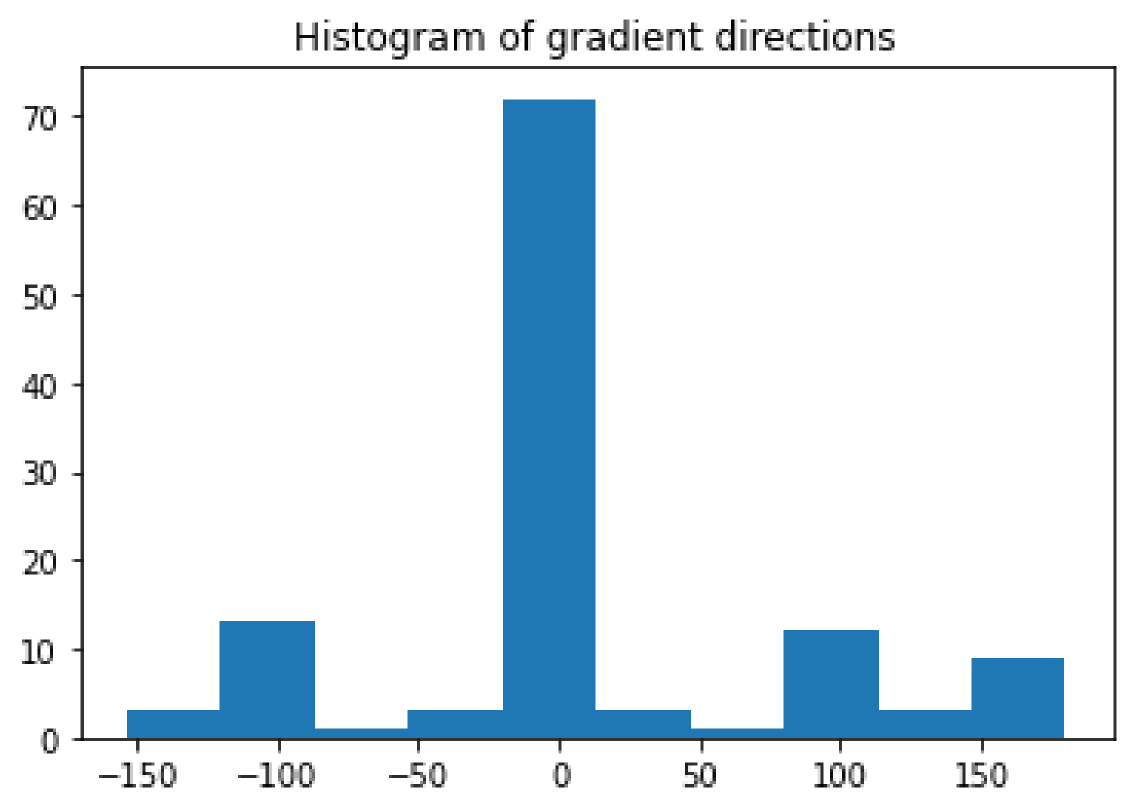


b) Sketch the histogram of gradient directions (**in degrees**).

▶ **Click here for an optional hint**

In [ ]:
```python
# Answer goes here
gradient_directions = np.degrees(np.arctan2(I_y, I_x))
plt.hist(gradient_directions.ravel())
plt.title('Histogram of gradient directions')
```

Out[ ]:
```
Text(0.5, 1.0, 'Histogram of gradient directions')
```



The Laplacian can be computed using the following mask:

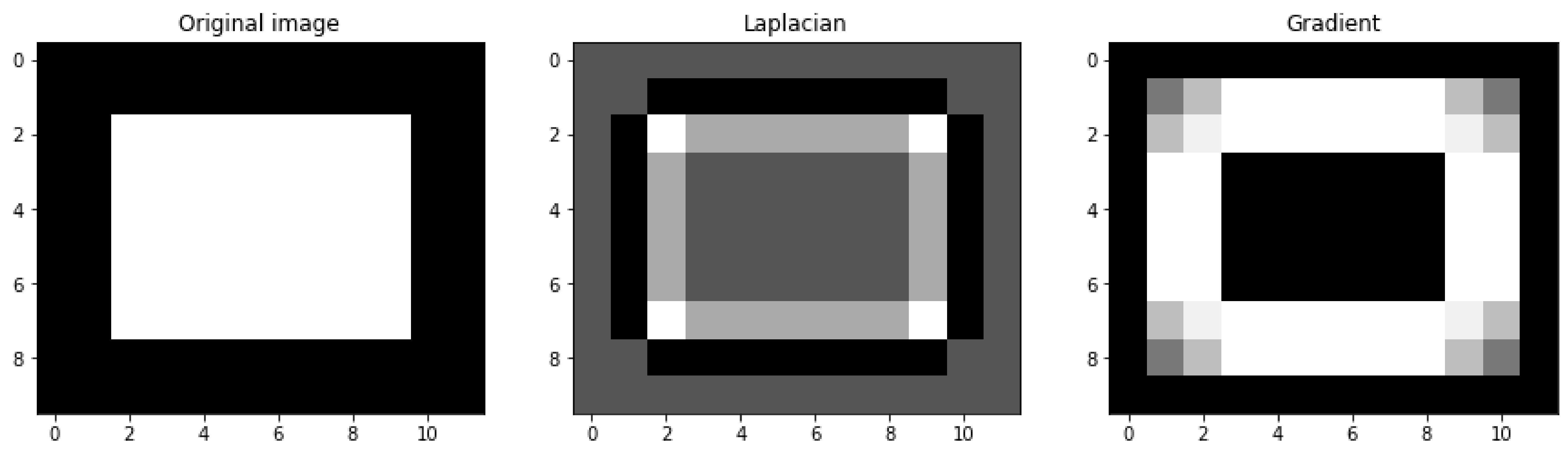
$$\mathbf{h}_L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad (19)$$

c) Sketch the Laplacian of the test image using the mask in previous equation. Show all relevant pixel values in the Laplacian image.

```
In [ ]: kernel_l = np.array([[0, -1, 0],
                             [-1, 4, -1],
                             [0, -1, 0]])
        I_l = cv2.filter2D(I, -1, kernel_l)
        gradient_i_l = np.sqrt(I_x**2 + I_y**2)

        fig, ax = plt.subplots(1, 3, figsize=(15, 10))
        ax[0].imshow(I, cmap='gray')
        ax[0].set_title('Original image')
        ax[1].imshow(I_l, cmap='gray')
        ax[1].set_title('Laplacian')
        ax[2].imshow(gradient_i_l, cmap='gray')
        ax[2].set_title('Gradient')
```

```
Out[ ]: Text(0.5, 1.0, 'Gradient')
```



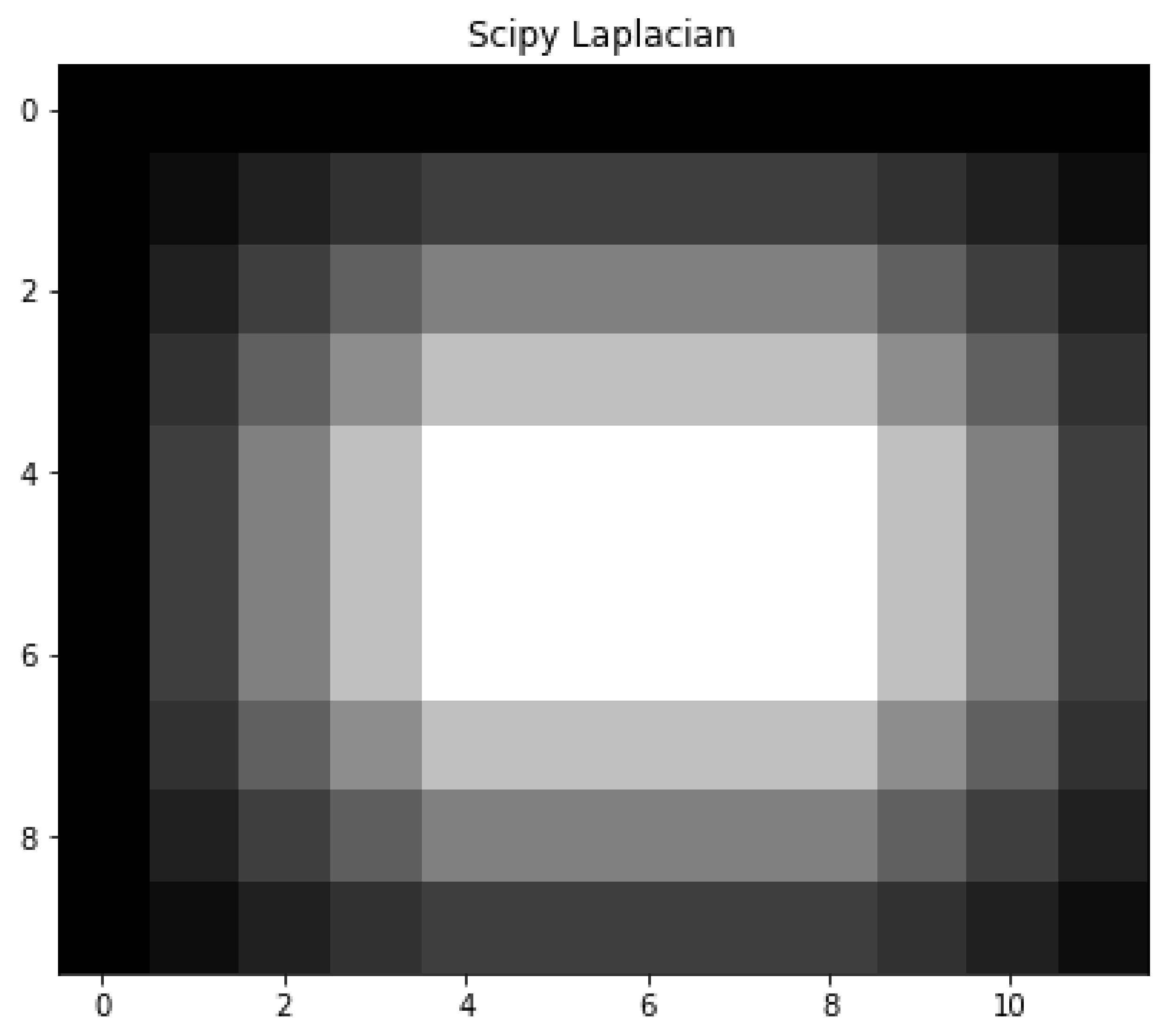
d) What is the resulting mask for computation of the Laplacian if the Prewitt masks are used for computation of the differentials?

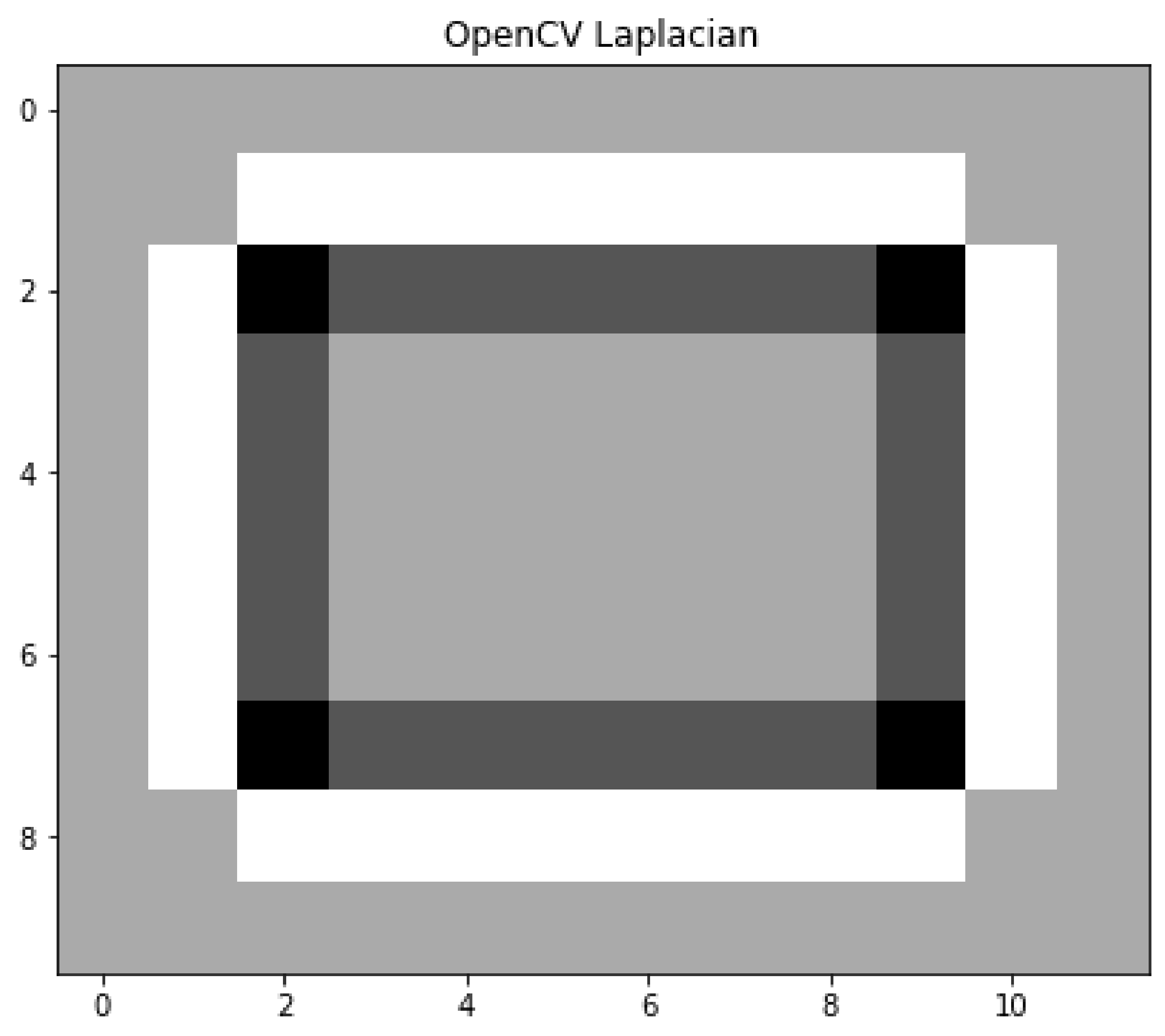
▶ **Click here for optional hints**

```
In [ ]: # Answer goes here
        import scipy
        I_lp = scipy.signal.convolve2d(I, gradient_prewitt, mode='same', boundary='symm')
        I_laplacian = cv2.Laplacian(I, -1)

        fig, ax = plt.subplots(1, 2, figsize=(15, 10))
        ax[0].imshow(I_lp, cmap='gray')
        ax[0].set_title('Scipy Laplacian')
        ax[1].imshow(I_laplacian, cmap='gray')
        ax[1].set_title('OpenCV Laplacian')
```

```
Out[ ]: Text(0.5, 1.0, 'OpenCV Laplacian')
```





Problem 3

One of the most common linear filters in computer vision applications is the Gaussian smoothing filter.

In this problem we want to study the use of Gaussian filters with different standard deviations, σ, and different sizes, $K \times K$, where K is odd ($K = 2k+1$, k is integer). The filter kernel (mask) is found by using the OpenCV function `cv2.getGaussianKernel()` (Documentation. Start by finding filter masks as follows

a) h1: $\sigma = 1$, $K = 9$

b) h15: $\sigma = 1.5$, $K = 11$

c) h2: $\sigma = 2$, $K = 15$

Use the `plt.stem()` function from Matplotlib and display each filter (sampled 1D Gaussian function).

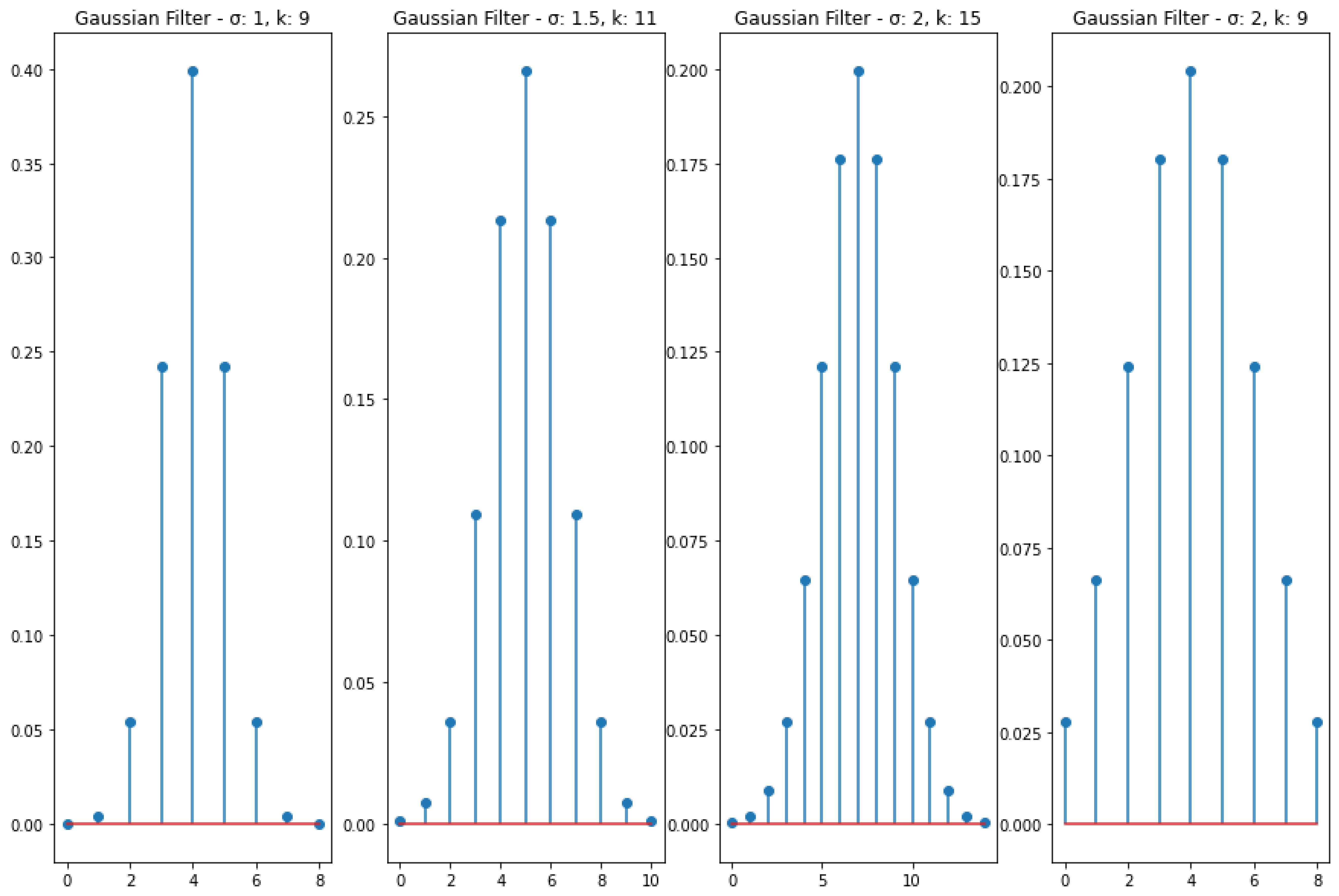
If the size K is too small we will get a truncated Gaussian with a step at the tails.

d) Show the result for c) above when $K = 9$.

If we want a proper Gaussian filter there is a connection between the value of σ and the size K. At three standard deviations, 3σ, the value of the Gaussian is 1% of its maximum value.

In [ ]:
```python
# Answer goes here
gauss = [[1, 9, -1], [1.5, 11, -1], [2, 15, -1], [2, 9, -1]]

fig, ax = plt.subplots(1, 4, figsize=(15, 10))
for i, gaus_filter in enumerate(gauss):
    gaus_filter[2] = cv2.getGaussianKernel(gaus_filter[1], gaus_filter[0])
    ax[i].stem(gaus_filter[2])
    ax[i].set_title(f'Gaussian Filter - σ: {gaus_filter[0]}, k: {gaus_filter[1]}')
```



Problem 4

In this exercise we want to study how two well-known filters perform on noise removal, namely the Gaussian and the median filter.

```
import cv2
from skimage.util import random_noise

Im = cv2.imread('./images/cameraman.jpg')
Im_gauss = random_noise(Im,  mode='gaussian', mean=0.05, var=0.01) # Gaussian white noise with variance of
0.01
Im_SP = random_noise(Im, 's&p', amount=0.05) # Salt and pepper noise on 5% of the pixels
```

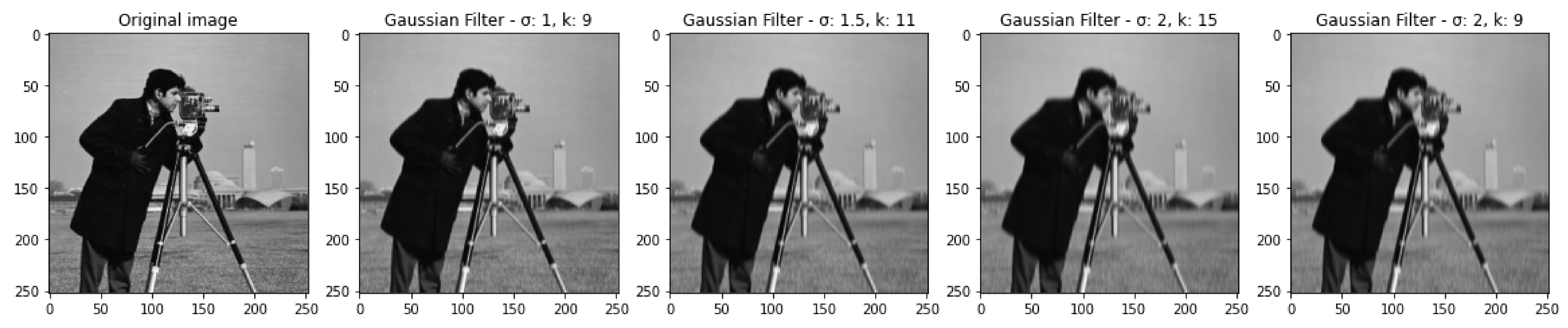
▶ **Click here for an optional hint**

a) Apply Gaussian smoothing to the original image, `Im`, using the defined filter kernels from problem 3. Explain the results.

```
In [ ]: # Answer goes here
import cv2
from skimage.util import random_noise

Im = cv2.imread('./images/cameraman.jpg')
Im_gauss = random_noise(Im,  mode='gaussian', mean=0.05, var=0.01) # Gaussian white noise with variance of 0.01
Im_SP = random_noise(Im, 's&p', amount=0.05) # Salt and pepper noise on 5% of the pixels

fig, ax = plt.subplots(1, len(gauss)+1, figsize=(20, 5))
ax[0].imshow(Im, cmap='gray')
ax[0].set_title('Original image')
for i, gaus_filter in enumerate(gauss):
    im_gauss_filtered = cv2.filter2D(Im, -1, gaus_filter[2])
    ax[i+1].imshow(im_gauss_filtered, cmap='gray')
    ax[i+1].set_title(f'Gaussian Filter - σ: {gaus_filter[0]}, k: {gaus_filter[1]}')
```


Original image
Gaussian Filter - σ: 1, k: 9
Gaussian Filter - σ: 1.5, k: 11
Gaussian Filter - σ: 2, k: 15
Gaussian Filter - σ: 2, k: 9

As seen in the results, the increase of the standard deviation increases the difumination of the pixels while smoothing the gray differences on it.

Gaussian noise:

b) Apply the three Gaussian filters, described in problem 3, to the image `Im_gauss`. Explain the results.

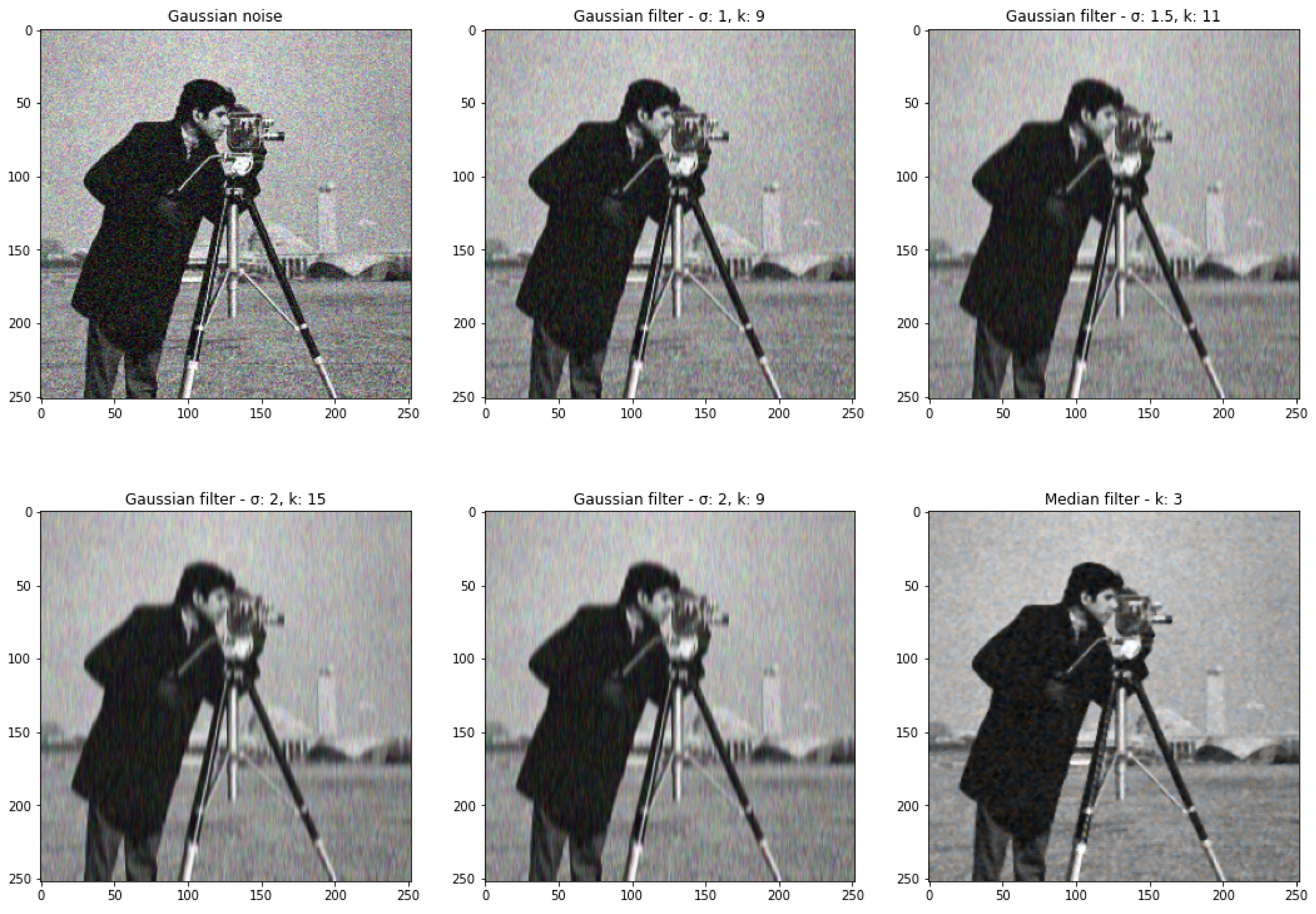
c) Apply a median filter on the image `Im_gauss` using the command `scipy.ndimage.median_filter` (Documentation). How does this filter perform compared to the Gaussian filters?

In [ ]:
```
# Answer goes here
from scipy.ndimage import median_filter
def compare_median_gauss_filter(img):
    fig, ax = plt.subplots(2,3,figsize=(18,13))
    ax[0,0].imshow(img, cmap='gray')
    ax[0,0].set_title('Gaussian noise')
    ax[0,1].imshow(cv2.filter2D(img,-1, gauss[0][2]))
    ax[0,1].set_title('Gaussian filter - σ: 1, k: 9')
    ax[0,2].imshow(cv2.filter2D(img,-1, gauss[1][2]))
    ax[0,2].set_title('Gaussian filter - σ: 1.5, k: 11')

    ax[1,0].imshow(cv2.filter2D(img,-1, gauss[2][2]))
    ax[1,0].set_title('Gaussian filter - σ: 2, k: 15')
    ax[1,1].imshow(cv2.filter2D(img,-1, gauss[3][2]))
    ax[1,1].set_title('Gaussian filter - σ: 2, k: 9')

    ax[1,2].imshow(median_filter(img, size=3))
    ax[1,2].set_title('Median filter - k: 3')
```

In [ ]:
```
compare_median_gauss_filter(Im_gauss)
```


Gaussian noise
Gaussian filter - σ: 1, k: 9
Gaussian filter - σ: 1.5, k: 11
Gaussian filter - σ: 2, k: 15
Gaussian filter - σ: 2, k: 9
Median filter - k: 3
0
50
100
150
200
250

The median filter looks like a better option for this kind of noise, since it preserves the edges of the image better than the Gaussian filters. However when looking at the computational side, the Gaussian filter is better, since it is much faster than the median filter.

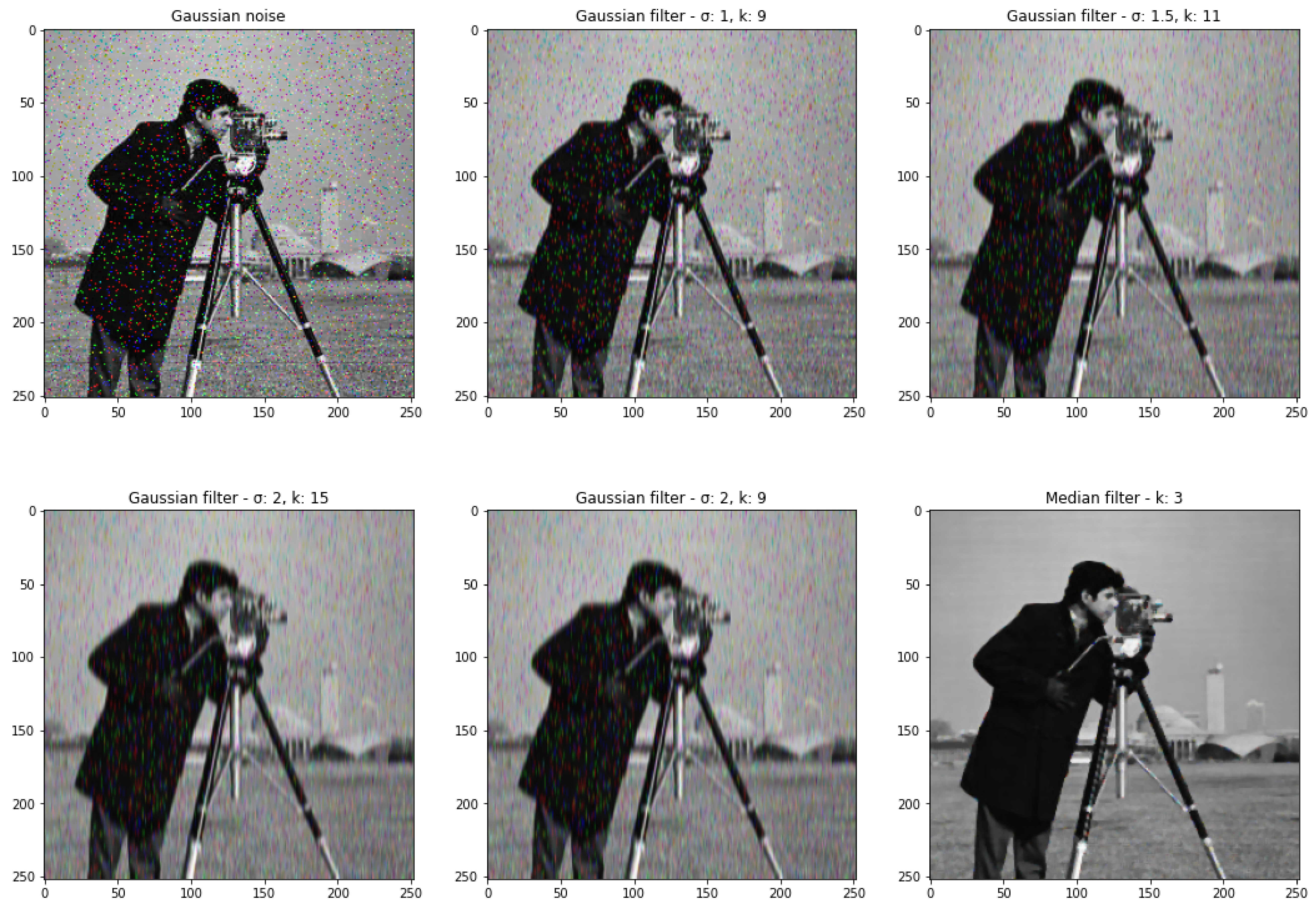
Salt & pepper noise:

d) Apply the three Gaussian filters, described in problem 3, to the image `Im_SP`. Explain the results.

e) Apply a median filter on the image `Im_SP` using the command `scipy.ndimage.median_filter` (Documentation). How does this filter perform compared to the Gaussian filters?

In [ ]:
```
# Answer goes here
```

```
compare_median_gauss_filter(Im_SP)
```



The results are even better. It seems like gauss is not able to distinguish between the noise from black and white pixels, while the median filter is able to do so. This is probably due to the fact that the median filter is able to take the median of the pixels around the current pixel, while the gaussian filter is not able to do so.

Delivery (dead line) on CANVAS: 30.09.2022 at 23.59

Contact

Course teacher

Professor Kjersti Engan, room E-431, E-mail: kjersti.engan@uis.no

Tomasetti Luca, room E-401 E-mail: luca.tomasetti@uis.no

Teaching assistant

Tomasetti Luca, room E-401 E-mail: luca.tomasetti@uis.no

Saul Fuster Navarro, room E-401 E-mail: saul.fusternavarro@uis.no

Jorge Garcia Torres Fernandez, room E-401 E-mail: jorge.garcia-torres@uis.no

References

[1] S. Birchfeld, Image Processing and Analysis. Cengage Learning, 2016.

[2] I. Austvoll, "Machine/robot vision part I," University of Stavanger, 2018. Compendium, CANVAS.