ALESSANDRO SERENA - AS6316 - CID 01188591

Coursework 1: Image Filtering

In this coursework we will explore some basic image filters used in computer vision. The corresponding lectures are Lectures 3 and 4 on image filtering and edge detection.

This coursework includes both coding questions as well as written ones. Please upload the notebook, which contains your code, results and answers as a pdf file onto Cate.

Dependencies: If you work on a college computer in the Computing Lab, where Ubuntu 18.04 is installed by default, you can use the following virtual environment for your work, where relevant Python packages are already installed.

```
source /vol/bitbucket/wbai/virt/computer_vision_ubuntu18.04/bin/activate
```

Alternatively, you can use pip, pip3 or anaconda etc to install Python packages.

Note: please read the both the text and code comment in this notebook to get an idea what you are supposed to implement.

In [1]:

```
# Import libaries
import imageio
import numpy as np
import matplotlib.pyplot as plt
import noisy
import scipy
import scipy.signal
import math
import time
```

1. Moving average filter (20 points)

Task: Read a specific input image and add noise to the image. Design a moving average filter of kernel size 3x3, 5x5 and 9x9 respectively. Display the filtering results and comment on the results.

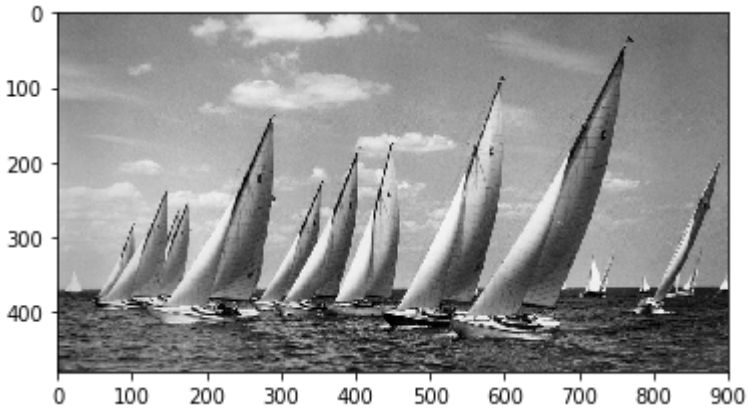
Please design the filter by yourself. Then, 2D image filtering can be performed using the function `scipy.signal.convolve2d()` .

In [2]:

```
# Read the image
image = imageio.imread('boat.png')
plt.imshow(image, cmap='gray')
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x10a9b050>
```

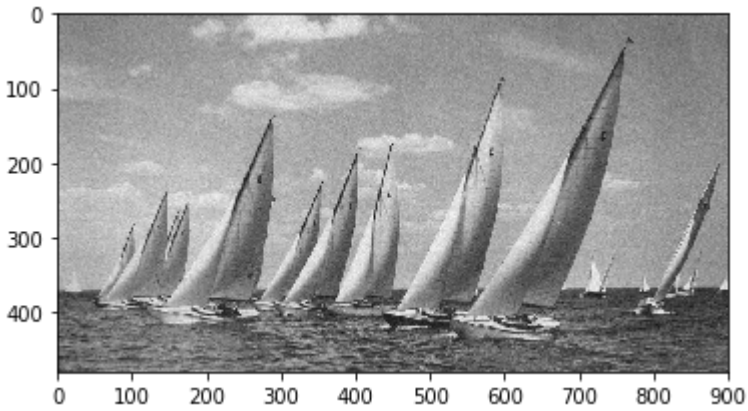


In [3]:

```
# Corrupt the image with Gaussian noise
image_noisy = noisy.noisy(image, 'gaussian')
plt.imshow(image_noisy, cmap='gray')
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x10c02ad0>
```



Note: from now on, please use the noisy image as the input for the filters.

1.1 Filter the noisy image with a 3x3 moving average filter (5 points)

In [4]:

```
# Design the filter h
i = 1.0/9
h = [[i , i , i],
     [i , i , i],
     [i , i , i]]

# Print the filter
print(h)

# Convolve the corrupted image with h using scipy.signal.convolve2d function
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same')
plt.imshow(image_filtered, cmap='gray')
```

```
[[0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.11111111
11111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.
1111111111111111, 0.1111111111111111]]
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x10bba590>
```

1.2 Filter the noisy image with a 5x5 moving average filter (5 points)

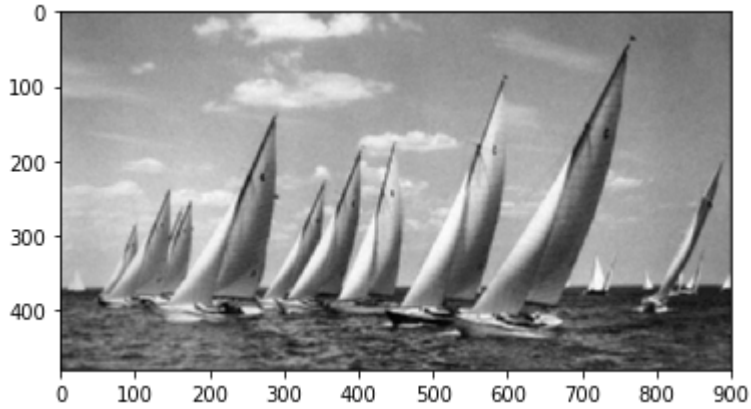
In [5]:

```
# Design the filter h
i = 1.0/25
h = [[i , i , i , i , i],
     [i , i , i , i , i],
     [i , i , i , i , i],
     [i , i , i , i , i],
     [i , i , i , i , i]]
# Print the filter
print(h)

# Convolve the corrupted image with h using scipy.signal.convolve2d function
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same')
plt.imshow(image_filtered, cmap='gray')
```

```
[[0.04, 0.04, 0.04, 0.04, 0.04], [0.04, 0.04, 0.04, 0.04, 0.04], [0.04, 0.
04, 0.04, 0.04, 0.04], [0.04, 0.04, 0.04, 0.04, 0.04], [0.04, 0.04, 0.04,
0.04, 0.04]]
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x10f45d50>
```



1.3 Filter the noisy image with a 9x9 moving average filter (5 points)

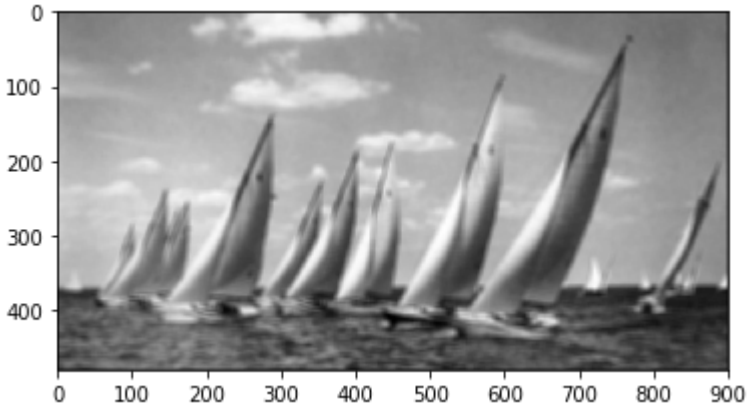
In [6]:

```
# Design the filter h
i = 1.0/81
h = [[i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i],
     [i , i , i , i , i , i , i , i , i]]


# Print the filter
print(h)

# Convolve the corrupted image with h using scipy.signal.convolve2d function
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same')
plt.imshow(image_filtered, cmap='gray')
```

```
[[0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.0123
45679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012
345678, 0.012345679012345678, 0.012345679012345678], [0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012
345679012345678, 0.012345679012345678, 0.012345679012345678, 0.01234567901
2345678, 0.012345679012345678], [0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012
345679012345678, 0.012345679012345678, 0.012345679012345678, 0.01234567901
2345678], [0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012
345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679
012345678, 0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012
345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679
012345678, 0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012
345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679
012345678, 0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.0
12345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679
012345678, 0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.0
12345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679
012345678, 0.012345679012345678, 0.012345679012345678, 0.0123456790123456
8, 0.012345679012345678]]
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x112fc1b0>
```



1.4 Comment on the filtering results when different window sizes are used (5 points)

Smoothing images by averaging neighbourhoods of pixels poses the issue of selecting an appropriate window size. As it is visible from the three examples provided above, different sizes generate different outcomes. A smaller window size (e.g. the 3x3 above) has better performance in preserving the quality of details whilst removing a decent amount of noise. Bigger windows (e.g. the 9x9 above) remove much more noise from the image but, at the same time, introduce the problem of blurring sharp edges. Therefore, as we can see, the bigger the window is the more blurred the details are going to be.

In conclusion, there is not a golden rule to establish which window size will produce a better outcome on any given image. In facts, when selecting the size of the kernel, we must take into consideration the size of the image features. In general, a smaller window will perform better (i.e. clear enough noise whilst not introducing too much blur) when the image contains small and sharp features. Vice versa, when objects are bigger, a larger window is preferable.

2. Edge detection (35 points)

Task: Perform edge detection using Sobel filters, as well as Gaussian + Sobel filters. Display the Sobel magnitude images and comment.

2.1 Implement 3x3 Sobel filters and convolve with the noisy image (5 points)

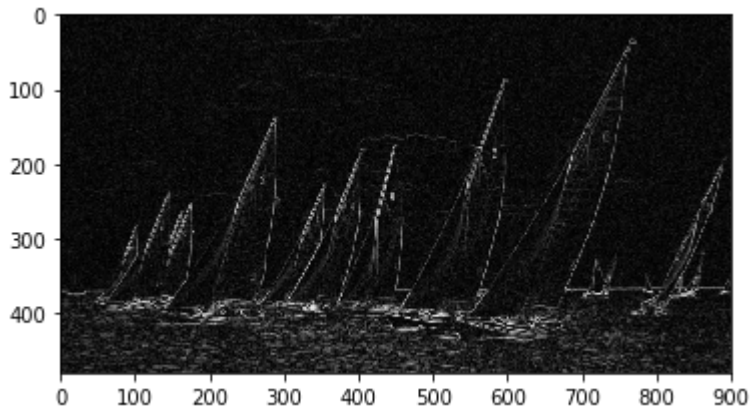
In [7]:

```
# Design the Sobel filters
h_sobel_x = [[1, 0, -1],
             [2, 0, -2],
             [1, 0, -1]]
h_sobel_y = [[ 1,  2,  1],
             [ 0,  0,  0],
             [-1, -2, -1],]

# Print the filters
print(h_sobel_x)
print(h_sobel_y)

# Sobel filtering
sobel_x = scipy.signal.convolve2d(image_noisy, h_sobel_x, mode='same')
sobel_y = scipy.signal.convolve2d(image_noisy, h_sobel_y, mode='same')

# Calculate the gradient magnitude
sobel_mag = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)

# Display the magnitude
plt.imshow(sobel_mag, cmap='gray')
```

```
[[1, 0, -1], [2, 0, -2], [1, 0, -1]]
[[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x11334a10>
```



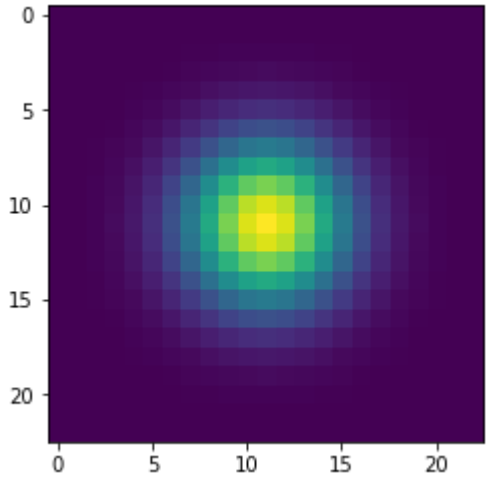
2.2 Design a 2D Gaussian filter (5 points)

In [8]:

```
#return value of Gaussian distribution given x,y and sigma
def gaussianDist2D(x, y, sigma):
    sigmaSquared = sigma**2
    return pow(np.e,(-((x**2)+(y**2))/(2*sigmaSquared))) / (2 * np.pi * sigmaSquared)

#return distance between point p and centre
def normToCentre(p, centre):
    return (math.sqrt((p-centre)**2))

# Design the Gaussian filter
def gaussian_filter_2d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 2D array for the Gaussian kernel

    # The filter radius is 3.5 times sigma
    rad = int(math.ceil(3.5 * sigma))
    sz = 2 * rad + 1
    # create empty kernel
    h = []
    kernelSum = 0
    # fill kernel with values
    for y in range(sz):
        row = []
        for x in range(sz):
            #normalize x and y with respect to distance to centre pixel of kernel
            normX = normToCentre(x, rad)
            normY = normToCentre(y, rad)
            temp = gaussianDist2D(normX, normY, sigma)
            kernelSum += temp
            row.append(temp)
        h.append(row)
    #normalize kernel dividing by total sum
    for i in range(sz):
        for j in range(sz):
            h[i][j] /= kernelSum
    return h

# Display the Gaussian filter when sigma = 3 pixel
sigma = 3
h = gaussian_filter_2d(sigma)
plt.imshow(h)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x114101d0>
```



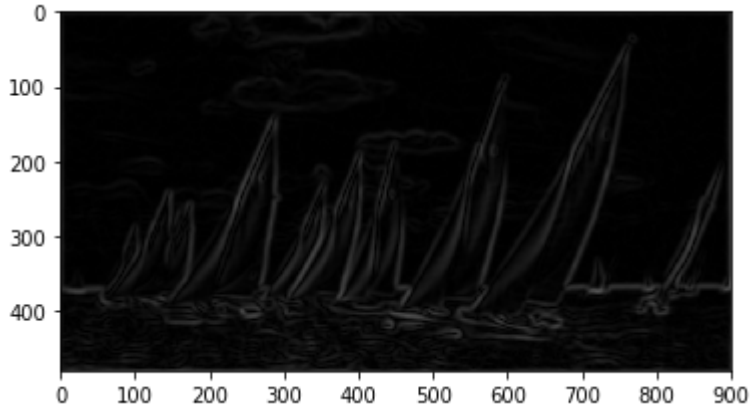
2.3 Perform Gaussian smoothing (σ = 3 pixels) before applying the Sobel filters (5 points)

In [9]:

```
# Perform Gaussian smoothing before Sobel filtering
sigma = 3
h = gaussian_filter_2d(sigma)
image_smoothed = scipy.signal.convolve2d(image_noisy, h, mode='same')

# Sobel filtering
sobel_x = scipy.signal.convolve2d(image_smoothed, h_sobel_x, mode='same')
sobel_y = scipy.signal.convolve2d(image_smoothed, h_sobel_y, mode='same')

# Calculate the gradient magnitude
sobel_mag = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)

# Display the magnitude
plt.imshow(sobel_mag, cmap='gray')
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x114418b0>
```



2.4 Perform Gaussian smoothing (σ = 7 pixels) before applying the Sobel filters. Evaluate the computational time for Gaussian smoothing. (5 points)

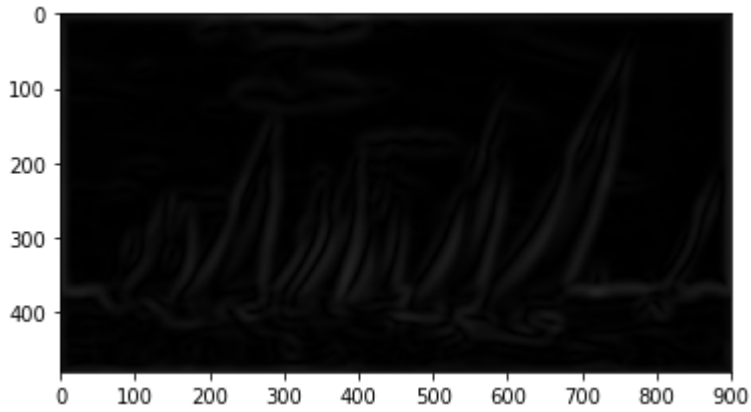
In [10]:

```
# Create the Gaussian filter
sigma = 7
h = gaussian_filter_2d(sigma)

# Perform Gaussian smoothing
start = time.time()
image_smoothed = scipy.signal.convolve2d(image_noisy, h, mode='same')
duration = time.time() - start
print('It takes {0:.6f} seconds for performing Gaussian smoothing.'.format(duration))

# Sobel filtering
sobel_x = scipy.signal.convolve2d(image_smoothed, h_sobel_x, mode='same')
sobel_y = scipy.signal.convolve2d(image_smoothed, h_sobel_y, mode='same')

# Calculate the gradient magnitude
sobel_mag = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)

# Display the magnitude
plt.imshow(sobel_mag, cmap='gray')
```

```
It takes 5.497000 seconds for performing Gaussian smoothing.
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x11032850>
```



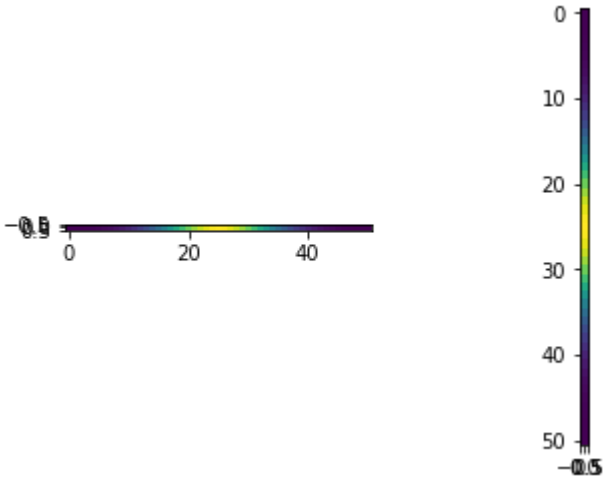
2.5 Design 1D Gaussian filters along x-axis and y-axis respectively. (5 points)

In [11]:

```
#return value of Gaussian distribution given x,y and sigma
def gaussianDist1D(x, sigma):
    sigmaSquared = sigma**2
    return pow(np.e,(-(x**2))/(2*sigmaSquared)) / math.sqrt(2 * np.pi * sigmaSquared)


# Design the Gaussian filter
def gaussian_filter_1d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 1D array for the Gaussian kernel

    # The filter radius is 3.5 times sigma
    rad = int(math.ceil(3.5 * sigma))
    sz = 2 * rad + 1
    h = []
    vectorSum = 0
    # Fill kernel with values
    for x in range(sz):
        #normalize x and y with respect to distance to centre pixel of vector
        normX =  normToCentre(x, rad)
        temp = gaussianDist1D(normX, sigma)
        vectorSum += temp
        h.append(temp)
    #normalize vector dividing by total sum
    for x in range(sz):
        h[x] /= vectorSum
    return h

# Display the Gaussian filters when sigma = 7 pixel
sigma = 7

# The Gaussian filter along x-axis. Its shape is (1, sz).
h_x = gaussian_filter_1d(sigma)
h_x = np.expand_dims(h_x, axis=0)

# The Gaussian filter along y-axis. Its shape is (sz, 1).
h_y = gaussian_filter_1d(sigma)
h_y = np.expand_dims(h_y, axis=-1)

# Display the filters
plt.subplot(1, 2, 1)
plt.imshow(h_x)
plt.subplot(1, 2, 2)
plt.imshow(h_y)
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x114ea290>
```



2.6 Perform Gaussian smoothing (sigma = 7 pixels) as two separable filters, then apply the Sobel filters. Evaluate the computational time for separable Gaussian filtering. (5 points)

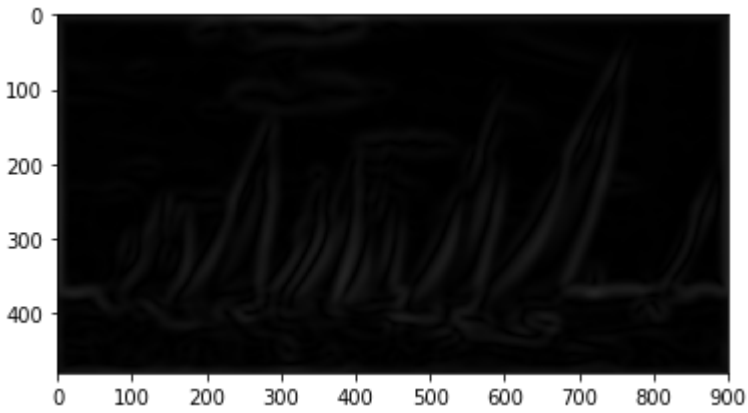
In [12]:

```
# Perform separable Gaussian smoothing before Sobel filtering
start = time.time()
image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same')
image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same')
duration = time.time() - start
print('It takes {0:.6f} seconds for performing Gaussian smoothing.'.format(duration))

# Sobel filtering
sobel_x = scipy.signal.convolve2d(image_smoothed, h_sobel_x, mode='same')
sobel_y = scipy.signal.convolve2d(image_smoothed, h_sobel_y, mode='same')

# Calculate the gradient magnitude
sobel_mag = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)

# Display the magnitude
plt.imshow(sobel_mag, cmap='gray')
```

```
It takes 0.460000 seconds for performing Gaussian smoothing.
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x1171b210>
```



2.7 Comment on the filtering results (5 points)

Gaussian blurring is a very effective method of noise attenuation. By controlling the σ (sigma) parameter it is possible to vary the blurring effect of the filter. Together with that, σ also determines the kernel size and therefore the complexity of the filtering operation. In the examples shown above we can see how increasing the parameter σ influences positively the reduction of noise. This facilitates the detection of edges performed by the Sobel operator. Furthermore, as in section 1, it is necessary to note that strong blurring (determined by high values of σ) might result in edges being excessively smoothed and therefore not easily identifiable after convolution with the Sobel kernel. We can infer that the blurring coefficient must be carefully chosen depending on the image features.

Sections 2.4 and 2.6 show how convolution can be simplified by separating a kernel into two vectors (a row and a column vector) and successively convolving them with an image. If we consider the multiplying and summing operations involved in convolution, convolving an image with a 7x7 kernel will determine a sum of 49 products for each pixel of the original image. On the other hand, convolving with a 7x1 kernel and then with a 1x7 kernel will need in total 2 sums of 7 products to process each pixel. It is then clear why separating the operations implies a lower computational complexity while producing exactly the same result. In this example (σ = 7) the improvement is noticeable in a reduction by a factor of ~10 in the execution time. In conclusion, we can note that the performance improvement is correlated with the dimensions of the matrix and its decomposed version. For example, running the same experiment with σ = 12 gives an improvement factor of ~20.

3. Laplacian filter (15 points)

Task: Perform Laplacian filtering and Laplacian of Gaussian filtering. Display the results and comment.

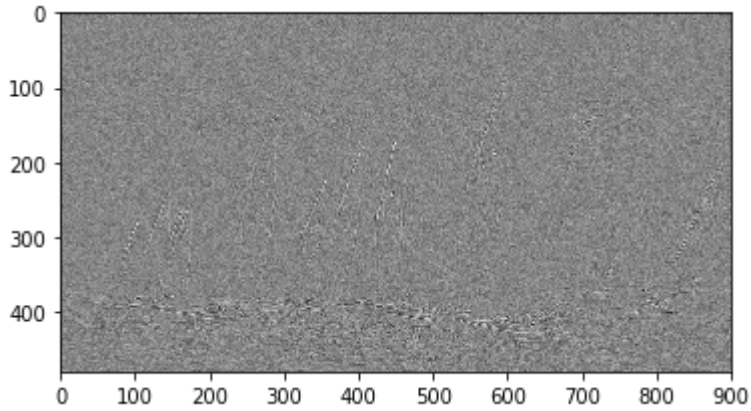
3.1 Implement a 3x3 Laplacian filter (5 points)

In [13]:

```
# Design the filter
h = [[0 , 1 , 0],
     [1 ,-4 , 1],
     [0 , 1 , 0]]

# Laplacian filtering
lap = scipy.signal.convolve2d(image_noisy, h, mode='same')

# Display the results
plt.imshow(lap, cmap='gray')
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x1635a930>
```



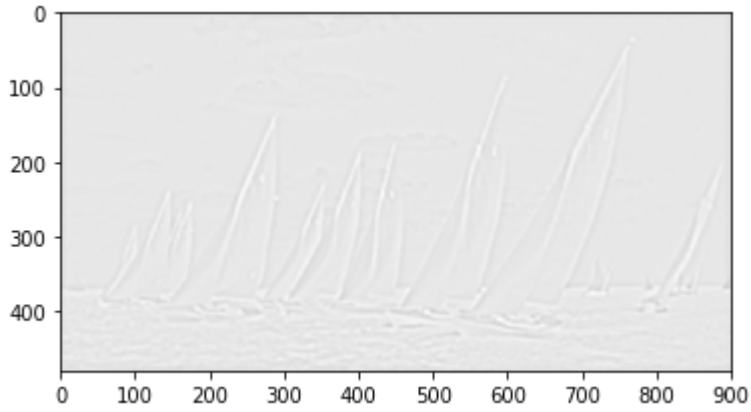
3.2 Implement the Laplacian of Gaussian filter (σ = 3 pixel) (5 points)

In [14]:

```
# Design the Gaussian filter
sigma = 3

# The Gaussian filter along x-axis
h_x = gaussian_filter_1d(sigma)
h_x = np.expand_dims(h_x, axis=0)

# The Gaussian filter along y-axis
h_y = gaussian_filter_1d(sigma)
h_y = np.expand_dims(h_y, axis=-1)

# Gaussian smoothing
image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same')
image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same')

# Design the Laplacian filter
h = [[0 , 1 , 0],
     [1 ,-4 , 1],
     [0 , 1 , 0]]

# Laplacian filtering
lap = scipy.signal.convolve2d(image_smoothed, h, mode='same')
plt.imshow(lap, cmap='gray')
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x11651c90>
```



3.3 Comments on the filtering results (5 points)

The Laplacian operator performs edge detection by calculating the second derivative of the image that it is convolved with. Since derivatives are very sensitive to noise, the first example (where the Laplacian kernel is applied directly on a noisy image) outputs a very noisy image where edges are really hard to identify.

When Gaussian blurring is used before applying the Laplacian operator (e.g. in 3.2), noise is reduced beforehand and the edge detection is much more effective.

4. Survey: How long does it take you to complete the coursework?

Put your answer here.

3.5 hours