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
from math import pi
```

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.

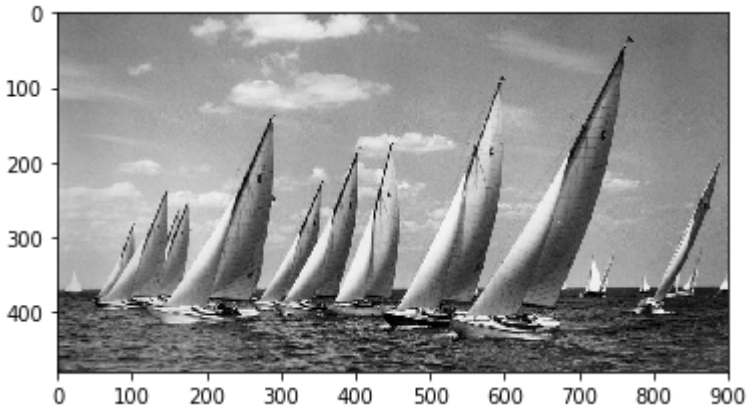
Please design the filter by yourself. Then, 2D image filtering can be performed using the function `scipy.signal.convolve2d()` .

In [2]:

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

Out[2]:

```
<matplotlib.image.AxesImage at 0x7fe6a38accf8>
```

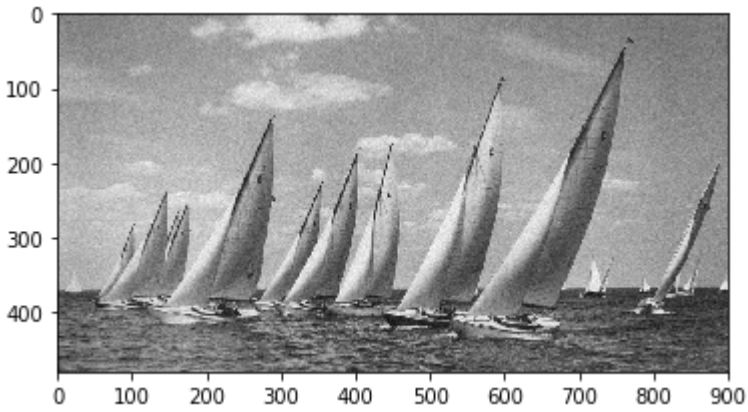


In [3]:

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

Out[3]:

```
<matplotlib.image.AxesImage at 0x7fe6a1fd0748>
```



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)

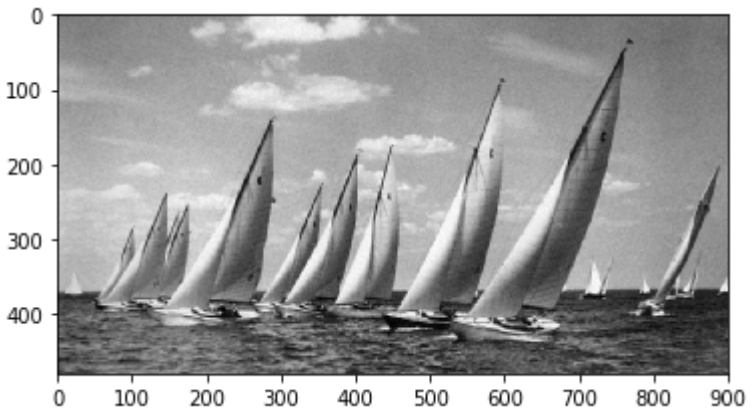
In [4]:

```
# Design the filter h
h = [[1/9,1/9,1/9], [1/9,1/9,1/9], [1/9,1/9,1/9]]

# 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], [0.1
11111111111111, 0.1111111111111111, 0.1111111111111111], [0.111111
1111111111, 0.1111111111111111, 0.1111111111111111]]
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7fe6a1fbcd30>
```



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

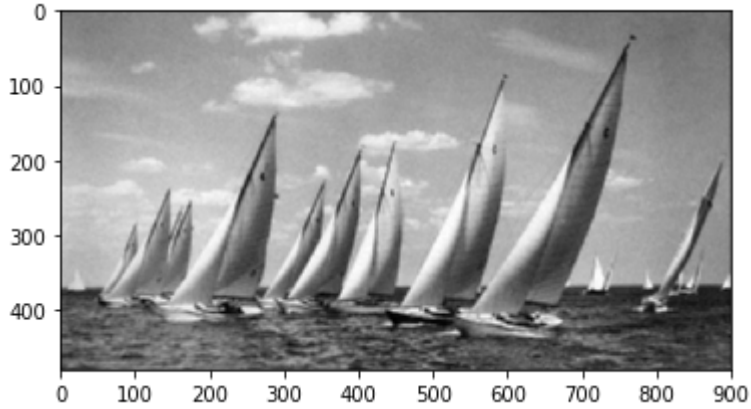
In [5]:

```
# Design the filter h
h = [[1/25,1/25,1/25,1/25,1/25],[1/25,1/25,1/25,1/25,1/25],[1/25,1/25,1/25,1/25,
1/25],[1/25,1/25,1/25,1/25,1/25],[1/25,1/25,1/25,1/25,1/25]]

# 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 0x7fe6a1f2b048>
```



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

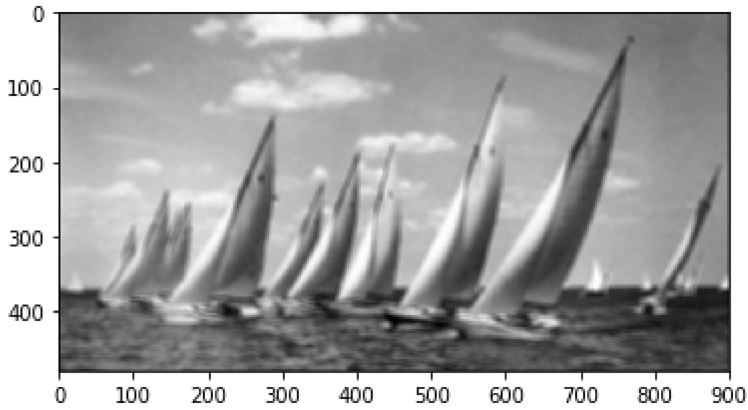
In [6]:

```
# Design the filter h
h = [[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81],[1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81,1/81]]

# 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, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678], [0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678, 0.012345679012345678]]
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7fe6a1e97cc0>
```



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

When the window size increases, the picture become more blurred and smooth. The moving average filter is simply a low pass filter, so it has removed high frequency component.

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)

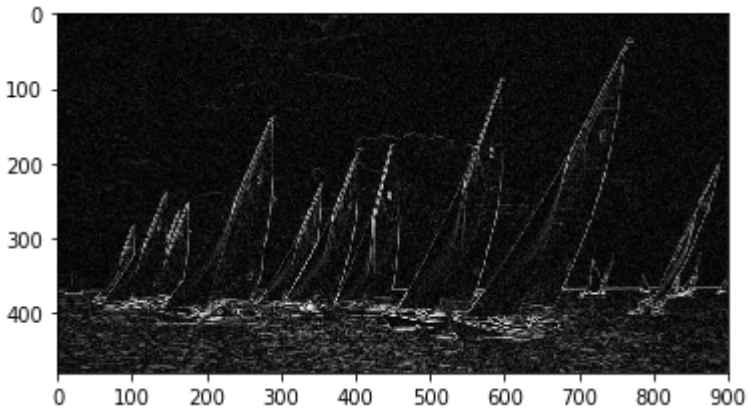
In [7]:

```
# 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 0x7fe6a1e6d8d0>
```



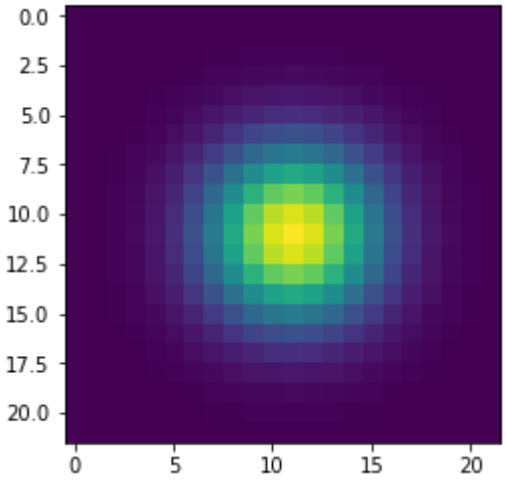
2.2 Design a 2D Gaussian filter (5 points)

In [8]:

```
# 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

    #initializes the matrix
    h = np.zeros(((sz-1),(sz-1))).astype(float)


    #calculating standard difference
    #in python matrix, vertical comes first in index
    x = (sz-1)/2
    y = (sz-1)/2

    #looping through each coordinate to change values
    for j in range(0,sz-1):#vertical axis
        for i in range(0,sz-1): #horizontal axis
            h[j][i] = (1/(2*pi*sigma*sigma))*math.exp(-((j-y)*(j-y)+(i-x)*(i-x))
/(2*sigma*sigma))

      #divide the sum of the all kernels
    return h/h.sum()


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

Out[8]:

```
<matplotlib.image.AxesImage at 0x7fe6a1dcd978>
```



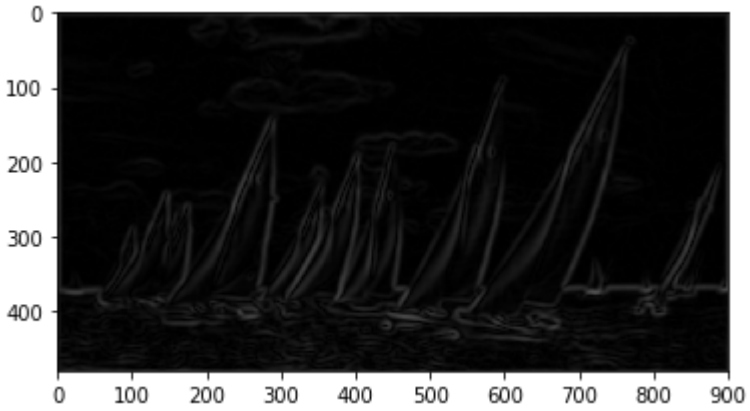
2.3 Perform Gaussian smoothing (σ = 3 pixels) before applying the Sobel filters (5 points)

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 0x7fe6a1daf128>
```



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

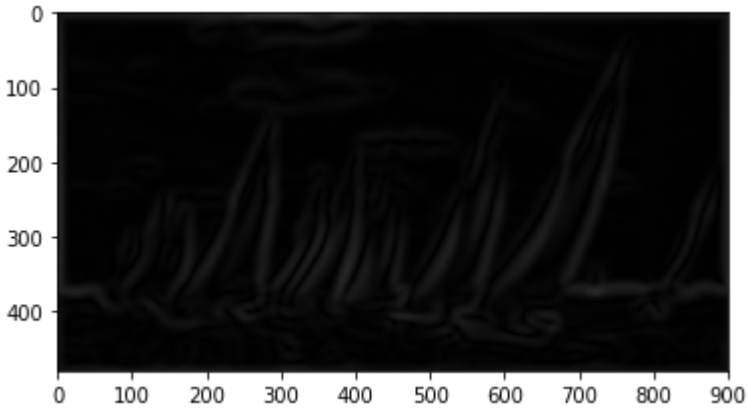
In [10]:

```
# 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(durat
ion))

# 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 2.032405 seconds for performing Gaussian smoothing.
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x7fe6a1d87c50>
```



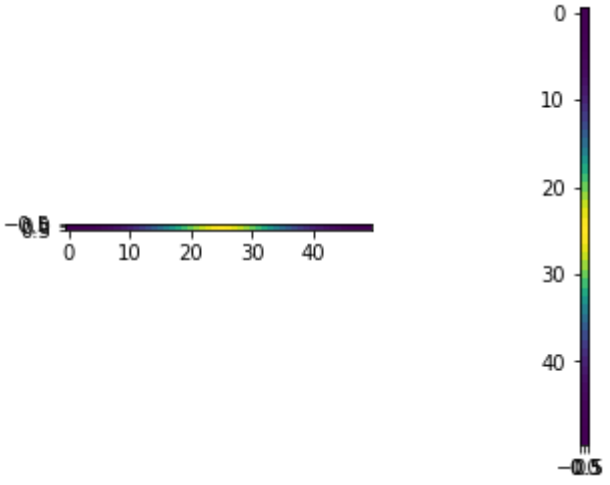
2.5 Design 1D Gaussian filters along x-axis and y-axis respectively. (5 points)

In [11]:

```
# 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

    #size (sz-1)
    x = (sz-1)/2
    h = np.zeros((sz-1))

    for i in range(0,sz-1):
        h[i] = (1/(math.sqrt(2*pi)*sigma))*math.exp((-(i-x)*(i-x))/(2*sigma*sigm
a))
    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 0x7fe6a1cf4400>
```



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)

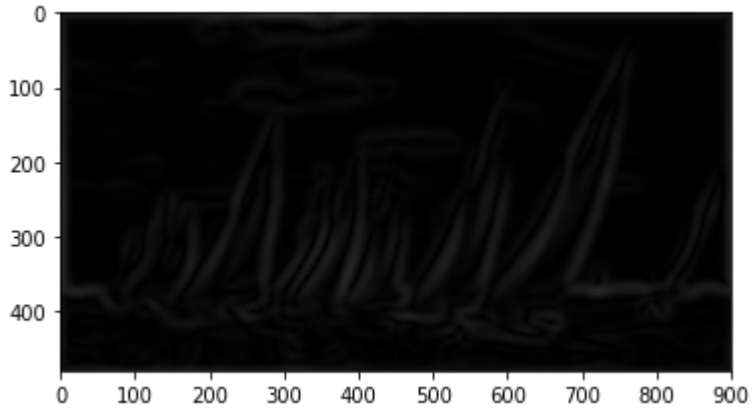
In [12]:

```
# 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(durat
ion))

# 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 0.157622 seconds for performing Gaussian smoothing.
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7fe6a192a048>
```



2.7 Comment on the filtering results (5 points)

The first picture compares to the second one(sigma of the gaussian increases) is more blurred in edge but remove more noise.By applying separatable filters, the running time reduces, and it is obvious when the kernel size increases(if we analysis the complexity of two algorithm, separate filter cost less runing time and complexity).

3. Laplacian filter (15 points)

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

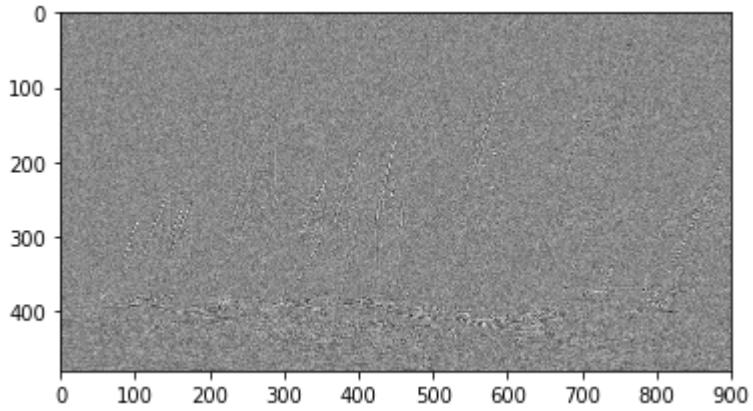
3.1 Implement a 3x3 Laplacian filter (5 points)

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 0x7fe6a1881c88>
```



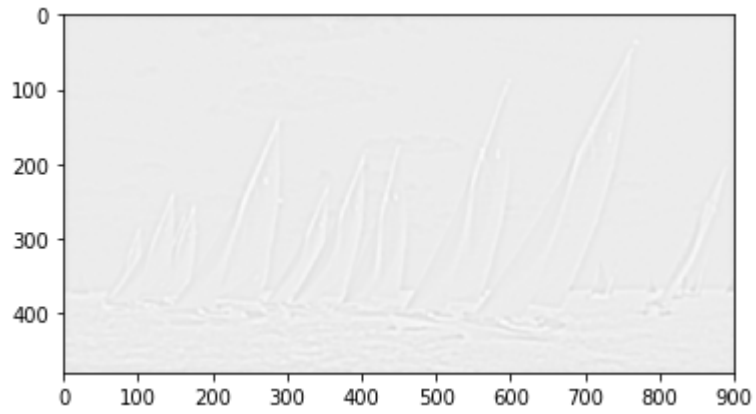
3.2 Implement the Laplacian of Gaussian filter (σ = 3 pixel) (5 points)

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 0x7fe6a1862f98>
```



3.3 Comments on the filtering results (5 points)

Laplacian filter highlights regions of rapid intensity change, so we can see in the first picture that it deemphasize regions with slowly varying gray levels The second one is more clear, because it reduces high frequency noise. Applying LOG make any edges in the original image much sharper and give them more contrast, and this is what it shows on the second graph.

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

Put your answer here.

In [ ]:

```
4 hours.
```