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.

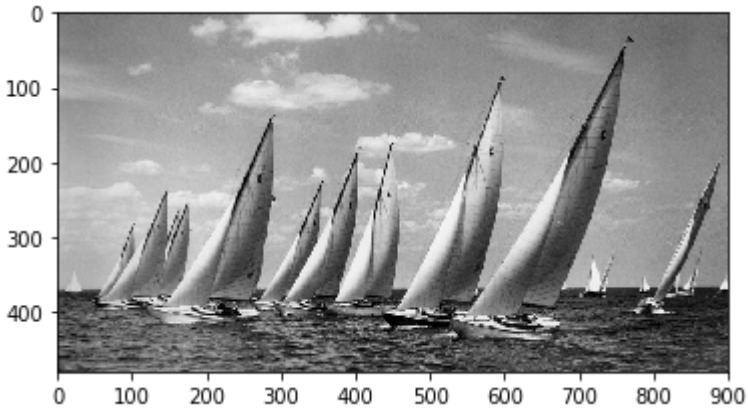
Please design the filter by yourself. Then, 2D image filtering can be performed using the function `scipy.signal.convolve2d()` .

In [2]:

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

Out[2]:

```
<matplotlib.image.AxesImage at 0x7f4b0f4f6cc0>
```

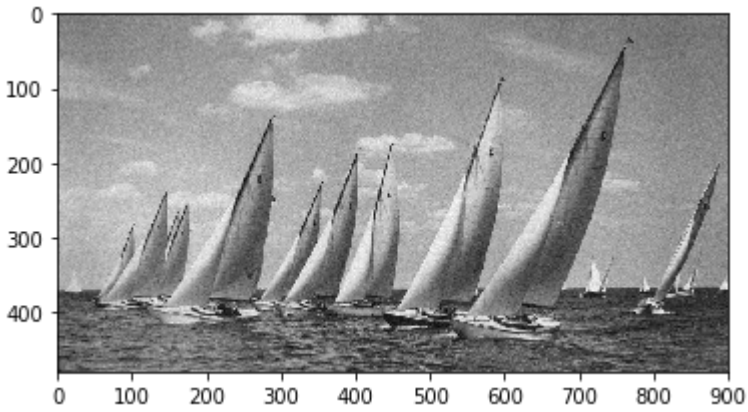


In [3]:

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

Out[3]:

```
<matplotlib.image.AxesImage at 0x7f4b0dc1b6d8>
```



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)

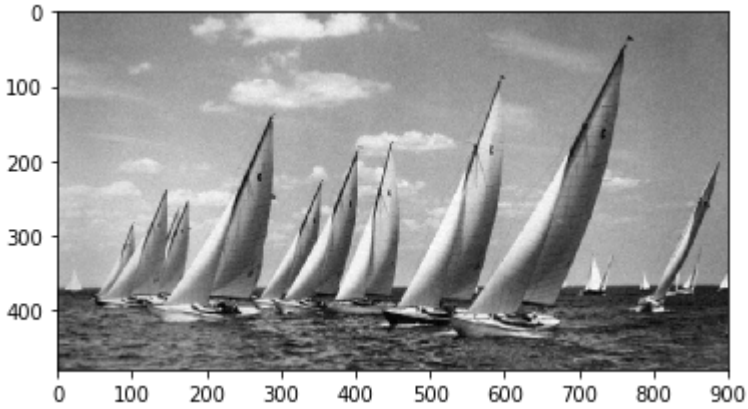
In [4]:

```
# Design the filter h
h = [[1/9,1/9,1/9],
     [1/9,1/9,1/9],
     [1/9,1/9,1/9]] #111 111 111 /9
#h = [[1,1,1],[1,1,1],[1,1,1]]/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
111111111111111, 0.1111111111111111, 0.1111111111111111], [0.111111
1111111111, 0.1111111111111111, 0.1111111111111111]]
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7f4b0dc05cc0>
```



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

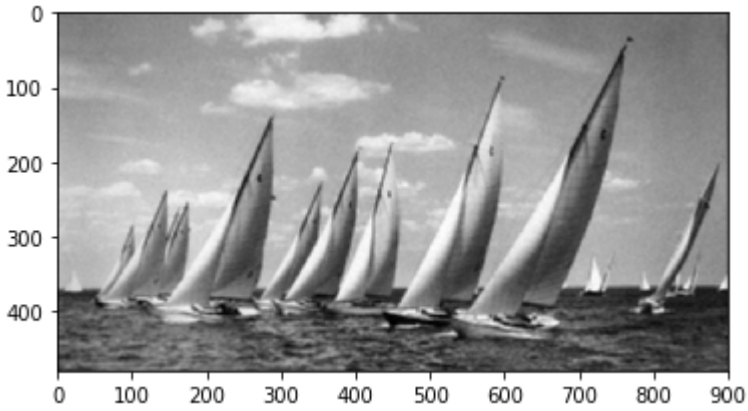
In [5]:

```
# 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 0x7f4b0db6ef98>
```



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

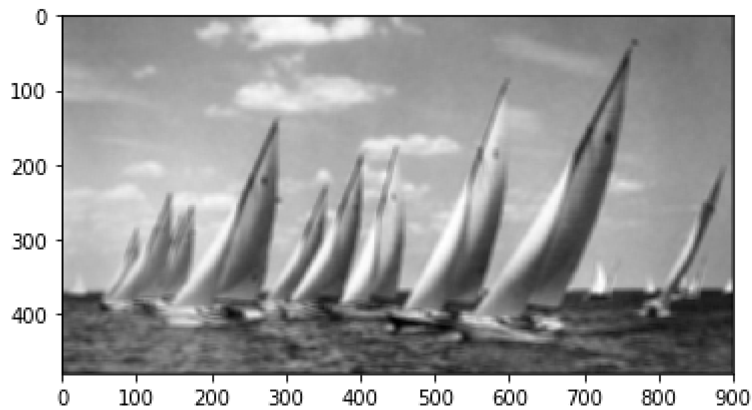
In [6]:

```
# 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 0x7f4b0dae2c88>
```



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

The moving average filter can be used to eliminate the noise of a picture, a window with bigger size can have a better result in removing the noise, but when the size is too big, the picture is blurred.

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)

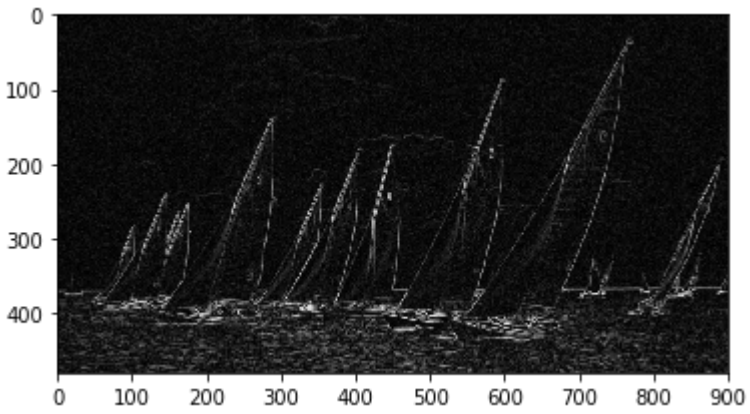
In [7]:

```
# 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 0x7f4b0dab5860>
```



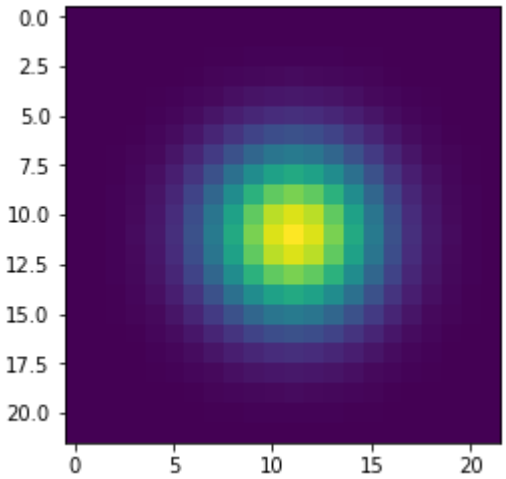
2.2 Design a 2D Gaussian filter (5 points)

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
    h = np.zeros((sz,sz))  #initiate a sz*sz matrix for h
    x, y = np.mgrid[-rad:rad, -rad:rad]
    h = (np.exp(-(x**2+y**2)/2/sigma**2))/(2*np.pi*sigma**2)
    return h

# Display the Gaussian filter when sigma = 3 pixel
sigma = 3 # 23*23
h = gaussian_filter_2d(sigma)
plt.imshow(h)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x7f4b0da970b8>
```



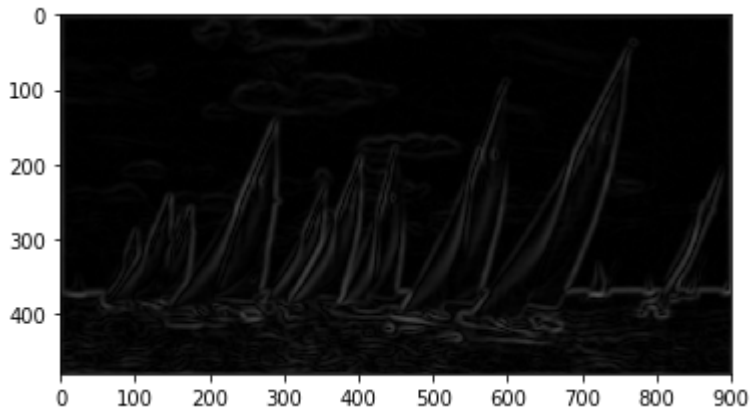
2.3 Perform Gaussian smoothing (σ = 3 pixels) before applying the Sobel filters (5 points)

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 0x7f4b0d9f60f0>
```



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

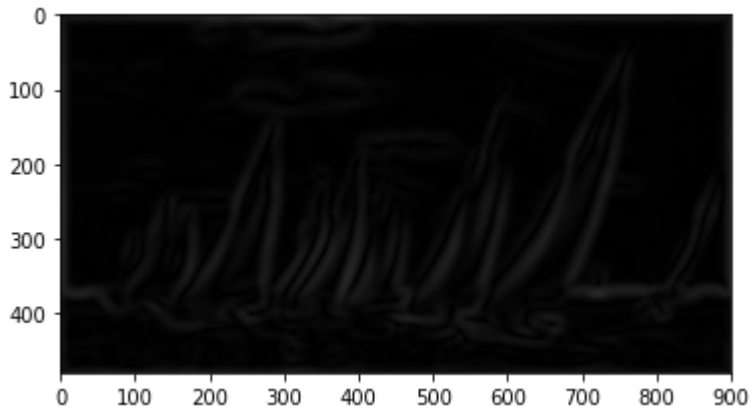
In [10]:

```
# 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')

# Evaluation of the computational time for Gaussian smoothing:
# For each pixel in the image, such a 2-dimension matrix of gaussian filter with
 size of sz*sz, which is 51*51 when sigma
# equals 7, will perform 51*51 mulplications and 51*51-1 summations, if it's a i
mage with the size of N*N pixels, the
# 2d-convolution performed in this section need N*N*(51*51)mulplcations and N*N*
(51*51-1) summations.And these steps will
# cost about 2 seconds.
# So the complexity is O(K^2*N^2), K^2 is the size of gaussian filter kernel.
```

```
It takes 2.069402 seconds for performing Gaussian smoothing.
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x7f4b0d9d0c18>
```



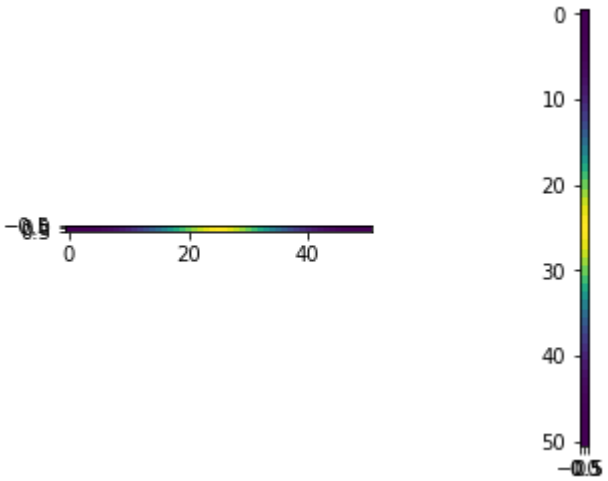
2.5 Design 1D Gaussian filters along x-axis and y-axis respectively. (5 points)

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
    t = np.mgrid[-rad:rad+1]  # t can be x or y
    h = np.zeros(sz) #initiate a 1*sz matrix for h
    h = 1/(np.sqrt(2*np.pi))/sigma*np.exp(-(t**2)/2/sigma**2)
    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 0x7f4b0d93d048>
```



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)

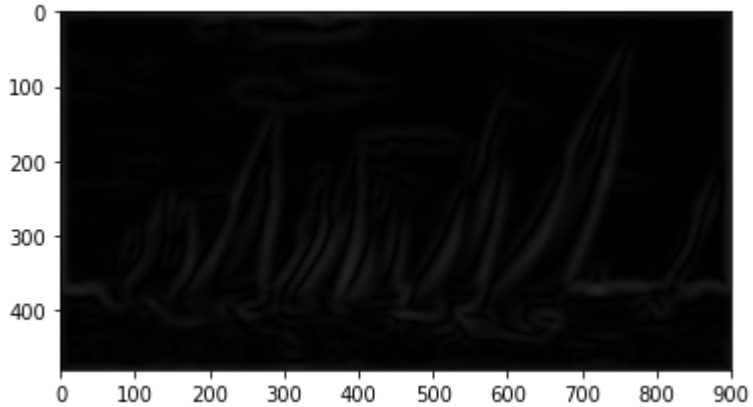
In [12]:

```
# 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')

# Evaluation of the computational time for separable Gaussian smoothing:
# When we use separable Gaussian smoothing kernel,for each pixel in the image, s
uch a 1-dimension matrix of gaussian
# filter with size of 1*sz or sz*1, which is 51*1 or 1*51 when sigma,equals 7, w
ill perform 51 mulplications and (51-1)
# summations, if it's a image with the size of N*N pixels, one 1d-convolution pe
rformed in this section need N*N*51
# mulplcations and N*N*50 summations.And there are two convolutions in this sect
ion, so these steps will perform twice,
# which will cost about 0.15 seconds.
# So the complexity is O(K*N^2), K is the size of gaussian filter kernels.
```

```
It takes 0.162590 seconds for performing Gaussian smoothing.
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7f4b0d56bb38>
```



2.7 Comment on the filtering results (5 points)

1.Funtion of Gaussian filter Because there are lots of noises in the image, before detecting the edges of the image, we usually first perform a Gausion filter to smooth the image, then perform the derivative or perform the convolution of the image and sobel filter, which will get the same result with direivative, to get the edges.

2.Influence of the value of sigma When sigma is bigger, the result for removing noise from images is better, but the image also become blurred.

3.Influence of the dimension of sigma The 2d Gaussian kernel can be represented by two 1d Gaussian kernels.While performing the convolution, the complexity of using 1d Gaussian kernel is O(K*N^2), which is O(K*^2N^2) for 2d Gaussian kernel, this will definitely save the time and space for the computing.

3. Laplacian filter (15 points)

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

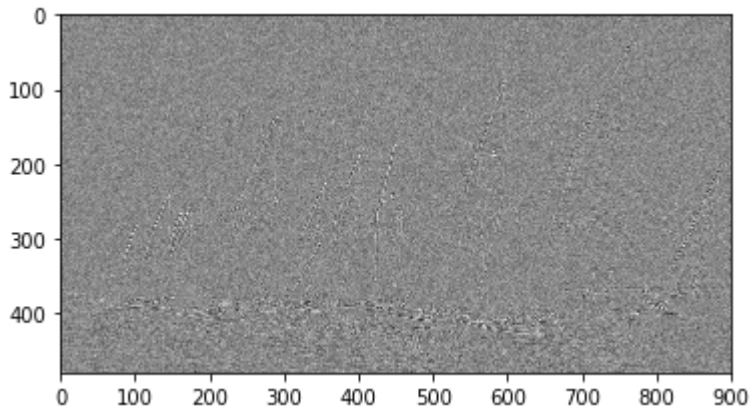
3.1 Implement a 3x3 Laplacian filter (5 points)

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 0x7f4b0d4c9780>
```



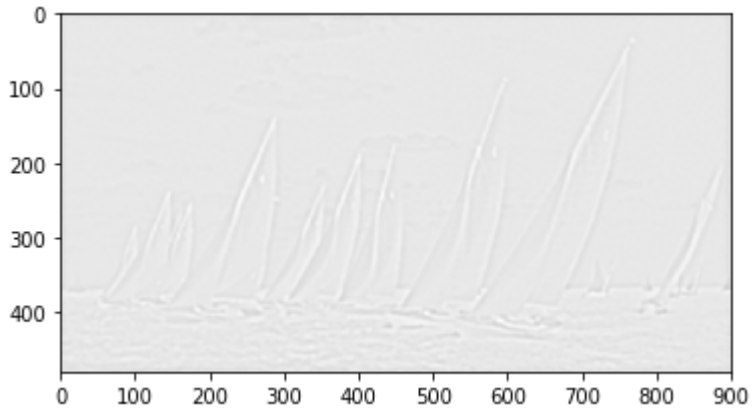
3.2 Implement the Laplacian of Gaussian filter (σ = 3 pixel) (5 points)

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 0x7f4b0d4ae9e8>
```



3.3 Comments on the filtering results (5 points)

We can also get the edges of a image by computing the second derivative of a image, which is equivalent to the convolution of image and Laplacian filter. But because the second derivative is more sensitive to the noise of a image, so we must use the Gaussian filter to smooth the picture before computing the second derivative, called LoG(Laplacian of Gaussian).

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

Put your answer here.

In [ ]:

```
# It takes me about 4-5 hours.
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```