CS558 - Computer Vision Assignment 1 (Edge Detector)

Step 1 - Gaussian filtering of image

The below function is just an example to show that the user can specify the value of sigma and two examples on the value of sigma from 1 to 10. Here, I have chosen the values 2 and 7.

In [2]:
```
# Generate gaussian filter
def generate_gaussian_filter(k, sigma):
    o = k // 2
    a = np.linspace(-o, o, k)
    b = np.linspace(-o, o, k)

    y, x = np.meshgrid(a,b)

    pi_2sig = 2 * np.pi * sigma # 2*pi*sig
    x2_y2 = np.square(x) + np.square(y) # x^2 + y^2
    sig2 = np.square(sigma) # sig^2

    gaussian_filter = 1 / (pi_2sig)*np.exp(-(x2_y2)/(2*sig2))

    return gaussian_filter
```

In [3]:
```
gauss5x5_2 = generate_gaussian_filter(5, 2)
gauss5x5_7 = generate_gaussian_filter(5, 7)
```

In [4]:
```
print("5x5 Gaussian filter with sigma 2:\n", gauss5x5_2)
print("\n")
print("5x5 Gaussian filter with sigma 7:\n", gauss5x5_7)
```

```
5x5 Gaussian filter with sigma 2:
 [[0.02927492 0.04259475 0.04826618 0.04259475 0.02927492]
 [0.04259475 0.061975   0.07022687 0.061975   0.04259475]
 [0.04826618 0.07022687 0.07957747 0.07022687 0.04826618]
 [0.04259475 0.061975   0.07022687 0.061975   0.04259475]
 [0.02927492 0.04259475 0.04826618 0.04259475 0.02927492]]


5x5 Gaussian filter with sigma 7:
 [[0.02095412 0.02160549 0.02182709 0.02160549 0.02095412]
 [0.02160549 0.02227711 0.0225056  0.02227711 0.02160549]
 [0.02182709 0.0225056  0.02273642 0.0225056  0.02182709]
 [0.02160549 0.02227711 0.0225056  0.02227711 0.02160549]
 [0.02095412 0.02160549 0.02182709 0.02160549 0.02095412]]
```

Step 2 - Edge detection using Sobel filter

This step showcases us the functions that are used to detect edges. filtering() is a convolving function that is used to apply our desired filter on the image. Since we have two different filters - Gaussian and Sobel, we call the filtering() function to apply gaussian filter which we can specify in the main function.

We call the filtering() function two times in the apply_sobel_filter() function for Gx and Gy respectively.

In [5]:
```
def filtering(img, filter_ip):
    img_r, img_c = img.shape #get image dimensions
    filter_r, filter_c = filter_ip.shape #get filter dimensions

    out_img = np.zeros(img.shape)

    fill_h = int((filter_r - 1) / 2)
    fill_w = int((filter_c - 1) / 2)
```

```
    # Create a image with pad of ones
    img_pad = np.ones((img_r + (2 * fill_h), img_c + (2 * fill_w)))

    # Apply the pad on the original image
    img_pad[fill_h : img_pad.shape[0] - fill_h, fill_w : img_pad.shape[1] - fill_w] = img

    # Copy the values of the rows and columns to the padded image
    for i in range(img_pad.shape[0]):
        for j in range(img_pad.shape[1]):
            img_pad[0,j] = img_pad[2,j]
            img_pad[1,j] = img_pad[2,j]
            img_pad[-1,j] = img_pad[-3,j]
            img_pad[-2,j] = img_pad[-3,j]

            img_pad[i,0] = img_pad[i,2]
            img_pad[i,1] = img_pad[i,2]
            img_pad[i,-1] = img_pad[i,-3]
            img_pad[i,-2] = img_pad[i,-3]

    for r in range(img_r):
        for c in range(img_c):
            out_img[r,c] = np.sum(filter_ip * img_pad[r : r + filter_r, c : c + filter_c])

    return out_img
```

In [6]:
```
def apply_sobel_filter(img, kernel):
    img_x = filtering(img, kernel)
    img_y = filtering(img, np.flip(kernel.T, axis=0))

    x_2 = np.square(img_x) # x^2
    y_2 = np.square(img_y) # y^2

    # Get magnitude of the gradient
    del_mag = np.sqrt(x_2 + y_2) # sqrt(x^2 + y^2)
    del_mag *= 255.0 / del_mag.max()

    # Get direction of the gradient and convert the value
    # from radians to degrees
    # Add 180 to convert range from (-180,180) to (0,360)
    del_dir = np.arctan2(img_y, img_x)
    del_dir = np.rad2deg(del_dir)
    del_dir += 180

    # Show figure
    if True:
        plt.imshow(del_mag, cmap='gray')
        plt.title("After Sobel")
        plt.show()

    return del_mag, del_dir
```

Step 3 - Applying Non Maximum Suppression

The following function applies non maximum suppression on the image that we acquire from the edge detection using sobel filter. The function takes the magnitude and the direction of the gradient(del) as well as the values of the thresholding parameters.

1. If the pixel value is greater than a certain value(high), it is considered as a strong pixel and its value is changed to the value of strong.
2. If the pixel value is less than a certain value(low), it is converted to 0.
3. Otherwise, the pixel is considered to be a weak pixel and its value is not altered.

In [7]:
```
def NMS(del_mag, del_dir, low, high, weak, strong):
    img_r, img_c = del_mag.shape

    out_img = np.zeros(del_mag.shape)

    for i in range(1, img_r - 1):
```

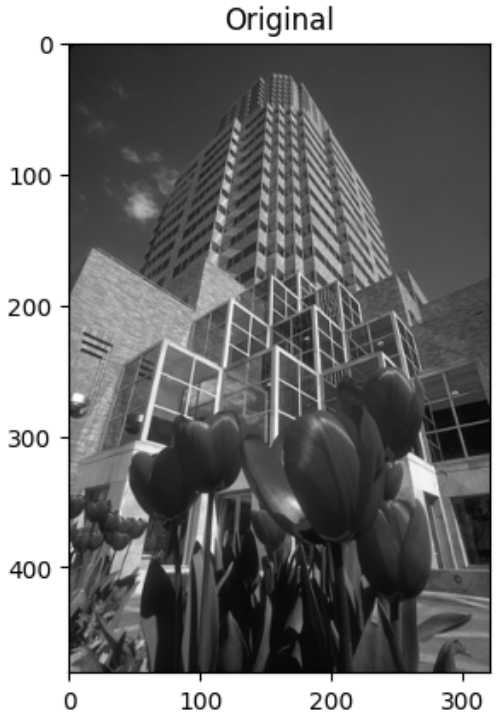
```
        for j in range(1, img_c - 1):
            orientation = del_dir[i, j]

            # Horizontal orientation
            if ((0 <= orientation < 22.5)
            or (337.5 <= orientation <= 360)
            or (157.5 <= orientation <= 202.5)):
                prev_px = del_mag[i, j - 1]
                next_px = del_mag[i, j + 1]
                if del_mag[i, j] >= prev_px and del_mag[i, j] >= next_px:
                    out_img[i, j] = del_mag[i, j]

            # Vertical orientation
            elif ((67.5 <= orientation < 112.5)
            or (247.5 <= orientation < 292.5)):
                prev_px = del_mag[i - 1, j]
                next_px = del_mag[i + 1, j]
                if del_mag[i, j] >= prev_px and del_mag[i, j] >= next_px:
                    out_img[i, j] = del_mag[i, j]

            # Orientation along the primary diagonal
            elif ((112.5 <= orientation < 157.5)
            or (292.5 <= orientation < 337.5)):
                prev_px = del_mag[i - 1, j - 1]
                next_px = del_mag[i + 1, j + 1]
                if del_mag[i, j] >= prev_px and del_mag[i, j] >= next_px:
                    out_img[i, j] = del_mag[i, j]

            # Orientation along the secondary diagonal
            elif ((22.5 <= orientation < 67.5)
            or (202.5 <= orientation < 247.5)):
                prev_px = del_mag[i + 1, j - 1]
                next_px = del_mag[i - 1, j + 1]
                if del_mag[i, j] >= prev_px and del_mag[i, j] >= next_px:
                    out_img[i, j] = del_mag[i, j]

            # Thresholding
            if out_img[i, j] >= high:
                out_img[i, j] = strong

            elif out_img[i, j] <= low:
                out_img[i, j] = 0

            else:
                out_img[i, j] = weak

    # Show figure
    if True:
        plt.imshow(out_img, cmap='gray')
        plt.title("After Non Maximum Suppression")
        plt.show()

    return out_img
```

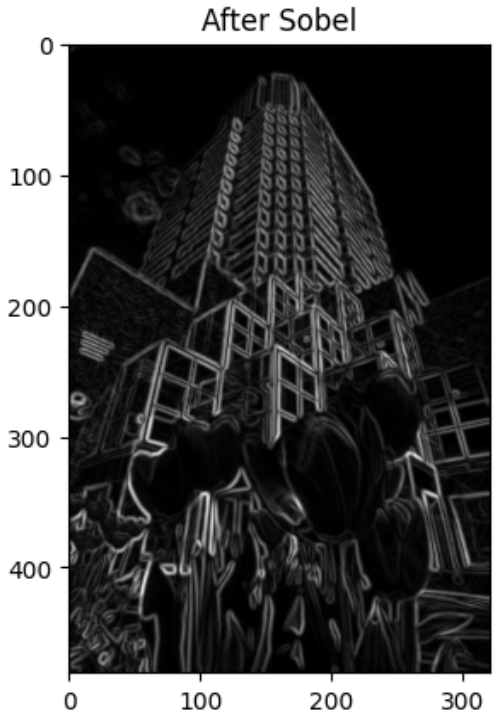
The main function has user-driven inputs to choose the value for k and sigma for the Gaussian filter as well as the choice to view any one of the three images given for the assignment. Input the corressponding number given with the images to view the original, after sobel filter, and after non-max suppression output.

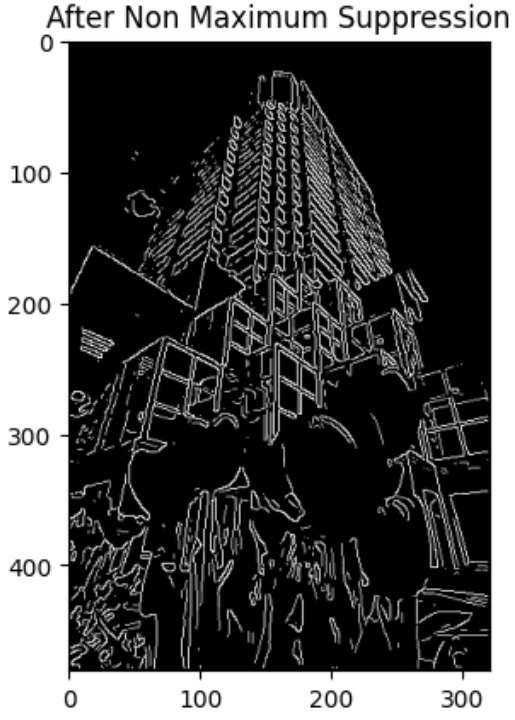
In [8]:
```
if __name__ == '__main__':
    # Assign values of k and sigma for the Gaussian filter
    k = int(input("Enter value of k for Gaussian kernel: "))
    sig = int(input("Enter value of sigma for Gaussian kernel: "))

    # Filter generation
    gauss = generate_gaussian_filter(k, sig)
    sobel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

    # Loop to choose between which image you want to see.
    # Don't restart the cell if no input given, you will have to
    # restart the kernel
    while True:
```

```
        choice = int(input("1: red.pgm, 2: plane.pgm, 3: kangaroo.pgm\n Input: "))

        if choice == 1:
            red = plt.imread('images/red.pgm')
            plt.imshow(red, cmap='gray')
            plt.title("Original")
            plt.show()

            red_gauss = filtering(red,gauss)
            red_sobel, red_dir = apply_sobel_filter(red_gauss, sobel)

            red_nms = NMS(red_sobel, red_dir, 40, 75, 200, 255)
            break

        elif choice == 2:
            plane = plt.imread('images/plane.pgm')
            plt.imshow(plane, cmap='gray')
            plt.title("Original")
            plt.show()

            plane_gauss = filtering(plane,gauss)
            plane_sobel, plane_dir = apply_sobel_filter(plane_gauss, sobel)

            plane_nms = NMS(plane_sobel, plane_dir, 40, 75, 128, 255)
            break

        elif choice == 3:
            kangaroo = plt.imread('images/kangaroo.pgm')
            plt.imshow(kangaroo, cmap='gray')
            plt.title("Original")
            plt.show()

            kangaroo_gauss = filtering(kangaroo,gauss)
            kangaroo_sobel, kangaroo_dir = apply_sobel_filter(kangaroo_gauss, sobel)

            kangaroo_nms = NMS(kangaroo_sobel, kangaroo_dir, 50, 85, 128, 255)
            break

        else:
            print("Invalid input, try again\n")
```

```
Enter value of k for Gaussian kernel: 5
Enter value of sigma for Gaussian kernel: 1
1: red.pgm, 2: plane.pgm, 3: kangaroo.pgm
 Input: 1
```




After Sobel
0
100
200
300
400
0
100
200
300


After Non Maximum Suppression
0
100
200
300
400
0
100
200
300

In [ ]: