Gaussian Filtering, Edge Detection using Sobel Filter and Non-maximum suppression

Importing libraries

In [41]:

```
from PIL import Image
import numpy as np
import math as m
import matplotlib.pyplot as plt
```

Padding function

In [42]:

```
def padding_the_image(image,size):
    img = np.array(image)
    prev_r, prev_c = img.shape
    img = np.pad(img,[(size//2, size//2), (size//2, size//2)]) ### here siz
e denotes the filter size
    r, c = img.shape

    ### sets the values for padded zeros in the image
    ### boundary values of the image are copied to the new padded cells (ro
w)
    for x in range(r):
        temp = size//2 - 1
        while temp >= 0:
            img[x][temp] = img[x][temp + 1]
            temp -= 1
        temp = size//2 + prev_c
        while temp < c:
            img[x][temp] = img[x][temp - 1]
            temp += 1

    ### sets the values for padded zeros in the image
    ### boundary values of the image are copied to the new padded cells (co
lumn)
    for y in range(c):
        temp = size//2 - 1
        while temp >= 0:
            img[temp][y] = img[temp + 1][y]
            temp -= 1
        temp = size//2 + prev_r
        while temp < r:
            img[temp][y] = img[temp - 1][y]
            temp += 1

    return img
```

Value calculator for 2D Gaussian filter

In [43]:

```
def val_for_gfilter(sigma, x, y):
    return (1/(2 * (m.pi) * ((sigma) ** 2))*(1/(m.exp(((x**2) + (y**2))/(2
* (sigma ** 2))))))
```

Creating the Gaussian filter

In [44]:

```
def gauss_filter(size, sigma):
    g_filter = np.zeros((size,size)) ### initially filling the filter with
 zeros
    res = 0 ### used to take the sum of all elements of the filter
    for x in range(size):
        for y in range(size):
            g_filter[x][y] = val_for_gfilter(sigma,x - (size//2),y - (size
//2))
            res += g_filter[x][y]

    g_filter *= 1.0/res ### we do this so that all values of the filter sum
to 1

    return g_filter
```

Function to apply Gaussian Filter on an image

In [45]:

```
def gfilter_the_img(image,gauss,size):
    image = np.array(image)
    image_row,image_col = image.shape
    res = np.zeros(image.shape)
    img = padding_the_image(image,size)

    ### calculating value for each cell of the smoothened image by matrix m
ultiplication
    for row in range(image_row):
        for col in range(image_col):
            res[row, col] = np.sum(gauss * img[row:row + size, col:col + si
ze])

    return res
```

Gradient Computation using Sobel Filters

In [46]:

```
def sobel_filtering(image):
    filter_x = [[-1,0,1],[-2,0,2],[-1,0,1]]
    filter_y = [[-1,-2,-1],[0,0,0],[1,2,1]]
    filter_x = np.array(filter_x)
    filter_y = np.array(filter_y)
    g_x = np.zeros(image.shape)
    g_y = np.zeros(image.shape)
    theta = np.zeros(image.shape)
    r, c = image.shape
    image = padding_the_image(image,3) ### padded with 3 because sobel filt
ers are 3 x 3 size

    ### calculating Gx
    for x in range(r):
        for y in range(c):
            g_x[x,y] = np.sum(filter_x * image[x:x+3,y:y+3])

    ### calculating Gy
    for x in range(r):
        for y in range(c):
            g_y[x,y] = np.sum(filter_y * image[x:x+3,y:y+3])

    ### calculating theta which is arctan(Gy/Gx)
    for x in range(r):
        for y in range(c):
            if(g_y[x][y] == 0 and g_x[x][y] == 0):
                theta[x][y] = 0
            elif(g_x[x][y] == 0):
                theta[x][y] = 90
            else:
                theta[x][y] = m.degrees(np.arctan(g_y[x][y]/g_x[x][y]))

    ### calculating magnitute
    g_x = g_x ** 2
    g_y = g_y ** 2
    g = np.add(g_x, g_y)
    g = g ** (0.5)
    ### selecting a threshold point
    tpoint = np.quantile(g, 0.85)
    for x in range(r):
        for y in range(c):
            if g[x][y] <= tpoint:
                g[x][y] = 0

    ### return image and the gradient matrix
    return g,theta
```

Non-maximum suppression

In [47]:

```
def non_max_suppress(image,theta):

    res = np.zeros(image.shape)
    r, c = image.shape
    img = padding_the_image(image,3)
    dev = 22.5

    for x in range(r):
        for y in range(c):
            if(0 - dev <= theta[x][y] <= 0 + dev or theta[x][y] >= 180 - de
v or theta[x][y] <= -180 + dev):
                if(img[x][y] > max(img[x-1][y],img[x+1][y])):
                    res[x][y] = img[x][y]
                else:
                    res[x][y] = 0
            elif(90 - dev <= theta[x][y] <= 90 + dev or -90 - dev <= theta[
x][y] <= -90 + dev):
                if(img[x][y] > max(img[x][y-1],img[x][y+1])):
                    res[x][y] = img[x][y]
                else:
                    res[x][y] = 0
            elif(45 - dev < theta[x][y] < 45 + dev or -135 - dev < theta[x]
[y] < -135 + dev):
                if(img[x][y] > max(img[x-1][y-1],img[x+1][y+1])):
                    res[x][y] = img[x][y]
                else:
                    res[x][y] = 0
            elif(135 - dev < theta[x][y] < 135 + dev or -45 - dev < theta[x
][y] < -45 + dev):
                if(img[x][y] > max(img[x-1][y+1],img[x+1][y-1])):
                    res[x][y] = img[x][y]
                else:
                    res[x][y] = 0

    ### used for wrapping
    res = res.astype(np.uint8)

    res = Image.fromarray(res)
    return res
```

Main function

In [48]:

```
def main(str = input("Enter image name with format: "), sigma = int(input(
"Enter the value of sigma: "))):
    ans = int(input("Press 1 for custom filter size or 0 for default size,
 i.e. 5: "))
    if ans == 1:
        size = int(input("Enter custom odd filter size: "))
    else:
        size = 5

    image = Image.open(str)
    image.show()
    g_img = gfilter_the_img(image,gauss_filter(size,sigma),size)
    temp_g = g_img.astype(np.uint8)
    temp_g = Image.fromarray(temp_g)
    temp_g = temp_g.save("gauss_1.jpg")
#     plt.imshow(g_img, cmap = 'gray')
#     plt.show()
    s_img_ar,direction = sobel_filtering(g_img)
    s_save = s_img_ar.astype(np.uint8)
    s_save = Image.fromarray(s_save)
    s_img = Image.fromarray(s_img_ar)
    s_save = s_save.save("sobel_1.jpg")
#     plt.imshow(s_img, cmap = 'gray')
#     plt.show()
    nms_img = non_max_suppress(s_img_ar,direction)
    nms_img = nms_img.save("nms_1.jpg")
#     plt.imshow(nms_img, cmap = 'gray')
#     plt.show()


main()
```

```
Enter image name with format: red.pgm
Enter the value of sigma: 1
Press 1 for custom filter size or 0 for default size, i.e. 5:
0
```