Source Code:

```
from PIL import Image
import numpy as np
import math as mt

def resize_image(ip_im, filter_size):
    r, c = ip_im.shape
    filter_n = int((filter_size-1)/2)
    op_r = r+2*(filter_n)
    op_c = c+2*(filter_n)
    op_im = np.zeros((op_r,op_c))
    for i in range(r):
        for j in range(c):
            op_im[i+filter_n][j+filter_n] = ip_im[i][j]
    for i in range(filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[filter_n][filter_n]
    for i in range(filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[filter_n][op_c-filter_n-1]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][filter_n]
    for i in range(op_r-filter_n, op_r):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[op_r-filter_n-1][op_c-filter_n-1]
    for i in range(filter_n):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[filter_n][j]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][j]
    for i in range(filter_n, op_r-filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[i][filter_n]
    for i in range(filter_n, op_r-filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[i][op_c-filter_n-1]
    return op_im

def convolution(ip,filter):
    filter_size = int(mt.sqrt(filter.size))
    filter_n = int((filter_size-1)/2)
    ip_r, ip_c = ip.shape
    r = ip_r - 2*filter_n
    c = ip_c - 2*filter_n
    op_im = np.zeros((r, c))
```

```
    for i in range(r):
        for j in range(c):
            for k in range(filter_size):
                for l in range(filter_size):
                    op_im[i][j] = op_im[i][j] + (filter[k][l] * ip[i+k][j+l])
    return op_im

def gauss_filter(im, size, sigma):
    size = int(size)
    sigma = float(sigma)
    og_im = np.array(im)
    filter = np.zeros((size,size))
    filter_n = int((size-1)/2)
    y, x = np.ogrid[float(-filter_n):float(filter_n+1),float(-filter_n):float(filter_n+1)]
    sum = 0
    for i in range(size):
        for j in range(size):
            e = mt.exp((-((x[0][j]**2)+(y[i][0]**2))/(2*(sigma**2))))
            filter[i][j] = e*(1/(2*mt.pi*(sigma**2)))
            sum = sum + filter[i][j]
    for i in range(size):
        for j in range(size):
            filter[i][j] = filter[i][j]/sum
    r, c = og_im.shape
    m_im = resize_image(og_im, size)
    m_r, m_c = m_im.shape
    op_im = convolution(m_im, filter)
    op_im = Image.fromarray(op_im)
    return op_im

def sobel_filter(im):
    filter_x = [[-1,0,+1], [-2,0,+2], [-1,0,+1]]
    filter_x = np.array(filter_x)
    filter_y = [[+1,+2,+1], [0,0,0], [-1,-2,-1]]
    filter_y = np.array(filter_y)
    og_im = np.array(im)
    r, c = og_im.shape
    m_im = resize_image(og_im, 3)
    gx_im = convolution(m_im,filter_x)
    gy_im = convolution(m_im,filter_y)
    orient = np.zeros((r, c))
    op_im = np.zeros((r, c))
    for i in range(r):
        for j in range(c):
            op_im[i][j] = mt.sqrt((gx_im[i][j]**2) + (gy_im[i][j]**2))
            if(op_im[i][j]<80):
                op_im[i][j]=0
            #else:
```

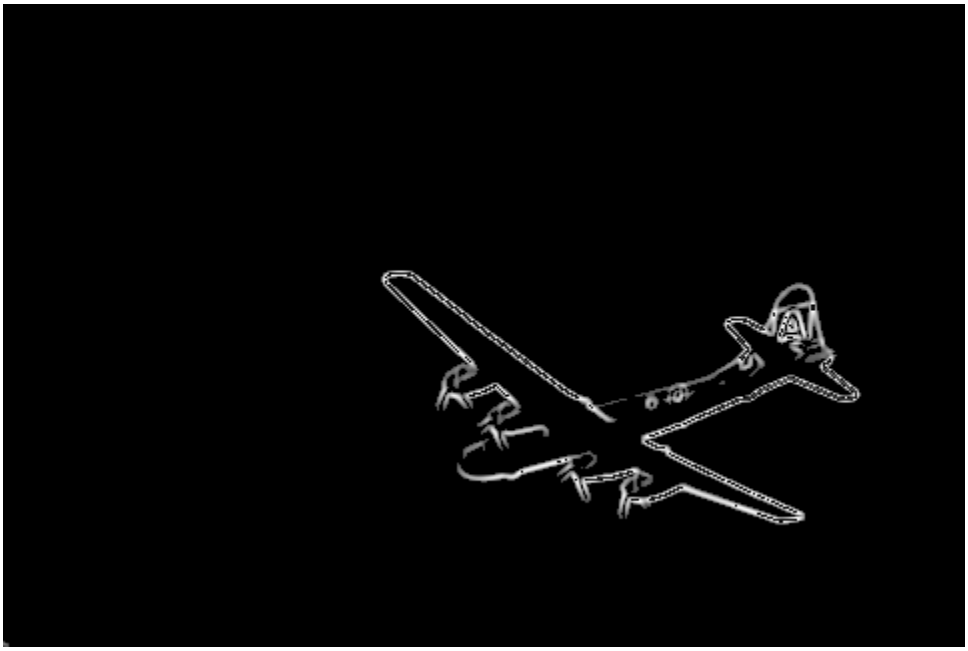
```
            #    op_im[i][j]=255
            orient[i][j] = mt.degrees(mt.atan(gy_im[i][j]/gx_im[i][j]))
    op_im = Image.fromarray(op_im)
    return op_im, orient

def non_max_supp(im, orient):
    og_im = np.array(im)
    r, c = og_im.shape
    op_im = np.zeros((r, c))
    m_im = resize_image(og_im, 3)
    for i in range(r):
        for j in range(c):
            if(float(-30)<=orient[i][j]<float(30) or float(150)<=orient[i][j]<float(-150)):
                if(m_im[i+1][j+1]==max(m_im[i+1][j],m_im[i+1][j+1],m_im[i+1][j+2])):
                    op_im[i][j] = m_im[i+1][j+1]
                else:
                    op_im[i][j] = 0
            if(float(30)<=orient[i][j]<float(60) or float(-150)<=orient[i][j]<float(-120)):
                if(m_im[i+1][j+1]==max(m_im[i+2][j],m_im[i+1][j+1],m_im[i][j+2])):
                    op_im[i][j] = m_im[i+1][j+1]
                else:
                    op_im[i][j] = 0
            if(float(60)<=orient[i][j]<float(120) or float(-120)<=orient[i][j]<float(-60)):
                if(m_im[i+1][j+1]==max(m_im[i][j+1],m_im[i+1][j+1],m_im[i+2][j+1])):
                    op_im[i][j] = m_im[i+1][j+1]
                else:
                    op_im[i][j] = 0
            if(float(120)<=orient[i][j]<float(150) or float(-60)<=orient[i][j]<float(-30)):
                if(m_im[i+1][j+1]==max(m_im[i][j],m_im[i+1][j+1],m_im[i+2][j+2])):
                    op_im[i][j] = m_im[i+1][j+1]
                else:
                    op_im[i][j] = 0
    op_im = op_im.astype(np.uint8)
    op_im = Image.fromarray(op_im)
    return op_im

def main(str = input('Please enter the image name with format: '), sigma = input('Please enter the value of
sigma: '), size = input('Please enter an odd filter size: ')):
    im = Image.open(str)
    im.show()
    im = gauss_filter(im, size, sigma)
    im, orient = sobel_filter(im)
    im = non_max_supp(im, orient)
    im.save('output_image.bmp')
    im.show()

main()
```

Output Images:

Gaussian Filter: Red Gaussian: Sigma = 1, Size = 5

Gaussian Filter: Red Gaussian: Sigma = 2, Size = 7

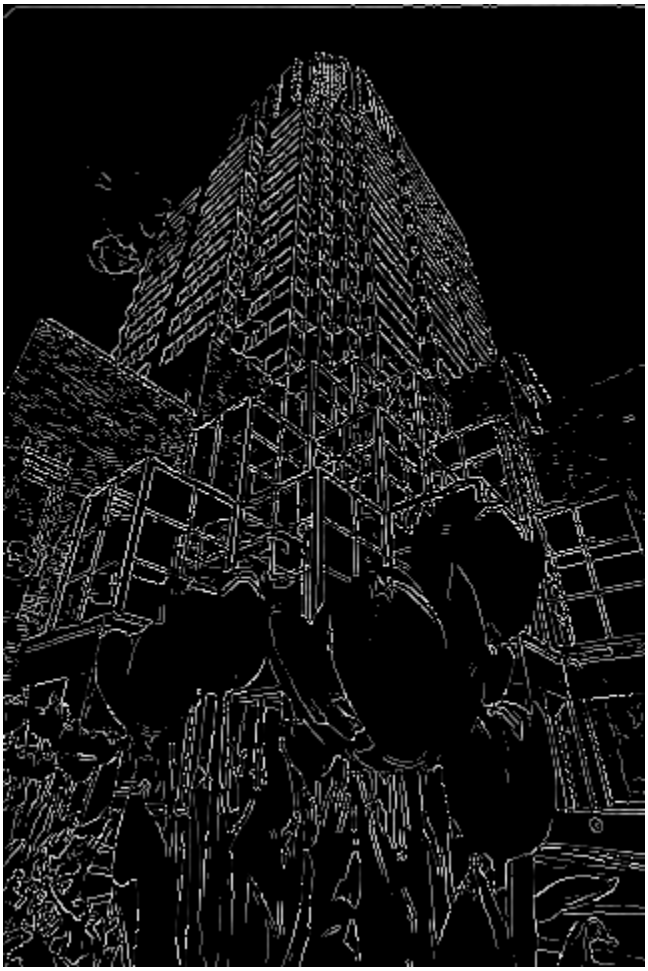
Sobel Filter: Red Gradient: Sigma = 1, Threshold = 75



Sobel Filter: Plane Gradient: Sigma = 1, Threshold = 75

Sobel Filter: Kangaroo Gradient: Sigma = 1, Threshold = 75

Non-Maximum Suppression: Red Final: Sigma = 1, Threshold = 80



Non-Maximum Suppression: Red Final: Sigma = 0.25, Threshold = 80

Non-Maximum Suppression: Plane Final: Sigma = 1, Threshold = 25

Non-Maximum Suppression: Plane Final: Sigma = 1.5, Threshold = 25

Non-Maximum Suppression: Kangaroo Final: Sigma = 1, Threshold = 50

Non-Maximum Suppression: Kangaroo Final: Sigma = 1.5, Threshold = 65

Additional Image: Lena Final: Sigma = 1, Threshold = 80

Additional Image: Lena Final: Sigma = 0.75, Threshold = 80

Notes:

Once the code is run, the actual image will be displayed. You might have to wait a while for the processing part, which, when complete, the final edge image will be displayed, after being saved by the name of ‘output_image’ as a BMP file.