In [ ]:
```
import numpy as np

from processImage import processImage
```

In [ ]:
```
img = processImage('../Bikesgray.jpg')
```

Sobel operator

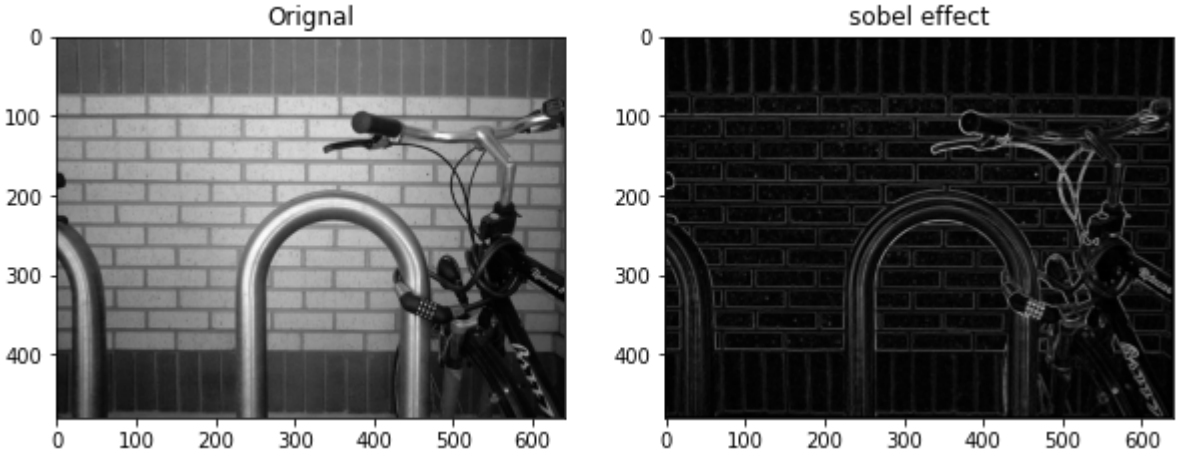
In [ ]:
```
# Edge Detection Kernel (sopel operator)
kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
```

In [ ]:
```
from convolve import convolve2D_multi_Kernal
# Convolve and Save Output
outputImage, theta = convolve2D_multi_Kernal(img, kernel_x, kernel_y, padding=1)
```

```
/home/shafei-aoc/Desktop/imageProcessing/task2/convolve.py:49: RuntimeWarning: divide by zero encountered in
double_scalars
  theta[i, j] = np.arctan(Gy / Gx)
/home/shafei-aoc/Desktop/imageProcessing/task2/convolve.py:49: RuntimeWarning: invalid value encountered in
double_scalars
  theta[i, j] = np.arctan(Gy / Gx)
```

In [ ]:
```
from plot import plot_image

plot_image(img, outputImage, title_2="sobel effect")
```



Prewitt operator

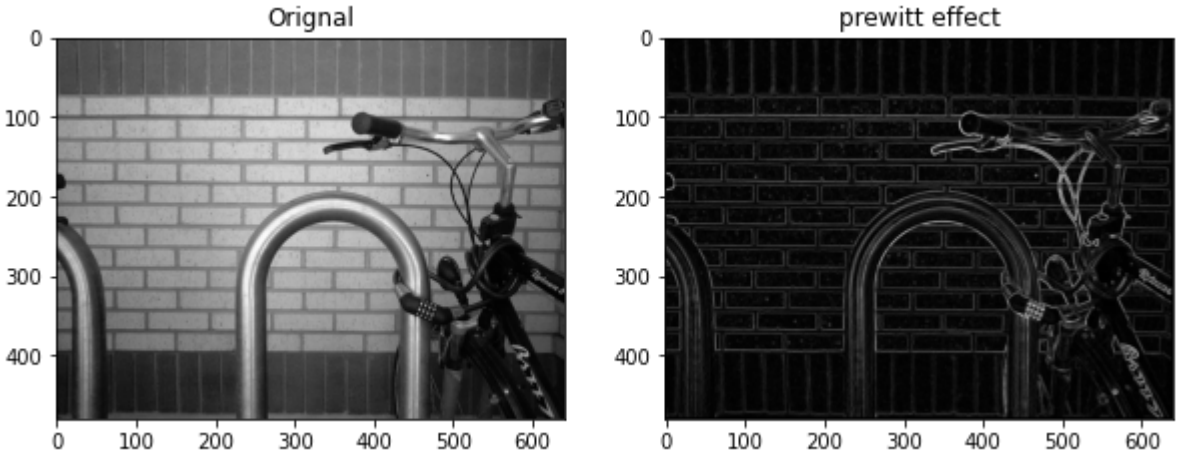
In [ ]:

```
# Edge Detection Kernel (prewitt operator)
kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

# Convolve and Save Output
outputImage, theta = convolve2D_multi_Kernal(img, kernel_x, kernel_y, padding=1)
```

In [ ]:

```
plot_image(img, outputImage, title_2="prewitt effect")
```



Laplacian of Gaussian (LoG)

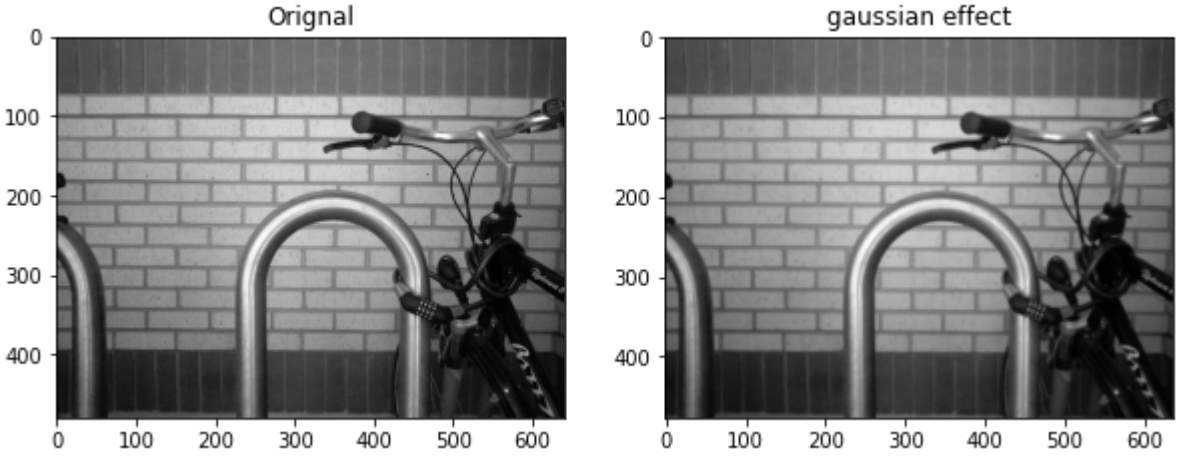
In [ ]:
```
import gaussian

gaussian_kernal = gaussian.generateGaussian(kernal_size=5, sigma = 1)
gaussian_kernal
```

Out[ ]:
```
array([[0.00291502, 0.01306423, 0.02153928, 0.01306423, 0.00291502],
       [0.01306423, 0.05854983, 0.09653235, 0.05854983, 0.01306423],
       [0.02153928, 0.09653235, 0.15915494, 0.09653235, 0.02153928],
       [0.01306423, 0.05854983, 0.09653235, 0.05854983, 0.01306423],
       [0.00291502, 0.01306423, 0.02153928, 0.01306423, 0.00291502]])
```

In [ ]:
```
from convolve import convolve2D_single_Kernal
image_qaussian_smoothed = convolve2D_single_Kernal(img, gaussian_kernal)
```

In [ ]:
```
plot_image(img, image_qaussian_smoothed, title_2="gaussian effect")
```



In [ ]:
```
laplacian_kernal = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])

laplacian_of_gaussian_kernal = convolve2D_single_Kernal(gaussian_kernal, laplacian_kernal)
laplacian_of_gaussian_kernal
```

Out[ ]:
```
array([[0.01500615, 0.08833553, 0.01500615],
       [0.08833553, 0.25049036, 0.08833553],
       [0.01500615, 0.08833553, 0.01500615]])
```

In [ ]:
```
image_laplacian_of_guassian = convolve2D_single_Kernal(img, laplacian_of_gaussian_kernal, padding=1)
```

In [ ]:
```
image_laplacian_of_guassian
# img.shape
```

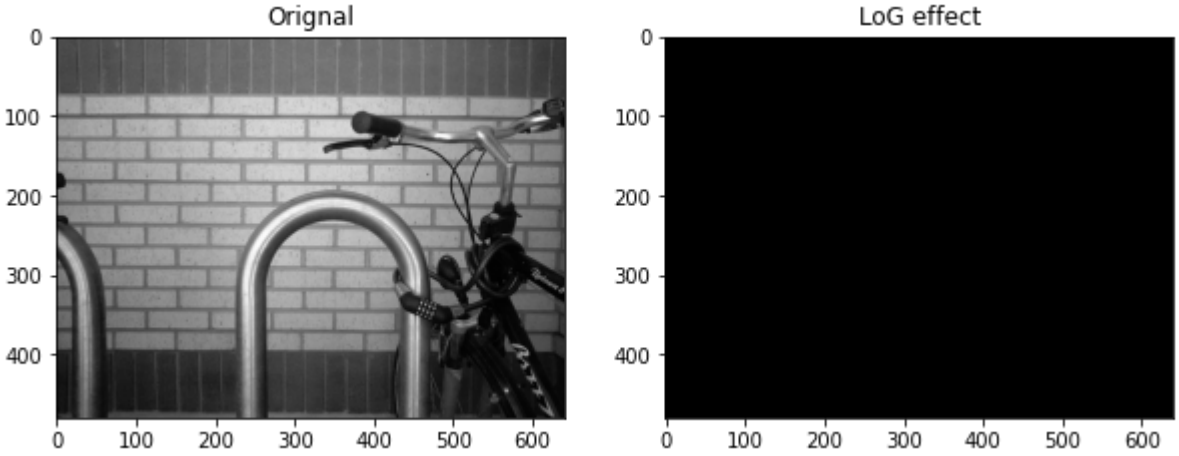
Applying zero crossing

In [ ]:
```
from zeroCrossing import zero_cross_detection
```

In [ ]:
```
image_after_z_c = zero_cross_detection(image_laplacian_of_guassian)
image_after_z_c
```

Out[ ]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [ ]:
```
plot_image(img, image_after_z_c , title_2="LoG effect")
```



Canny operator

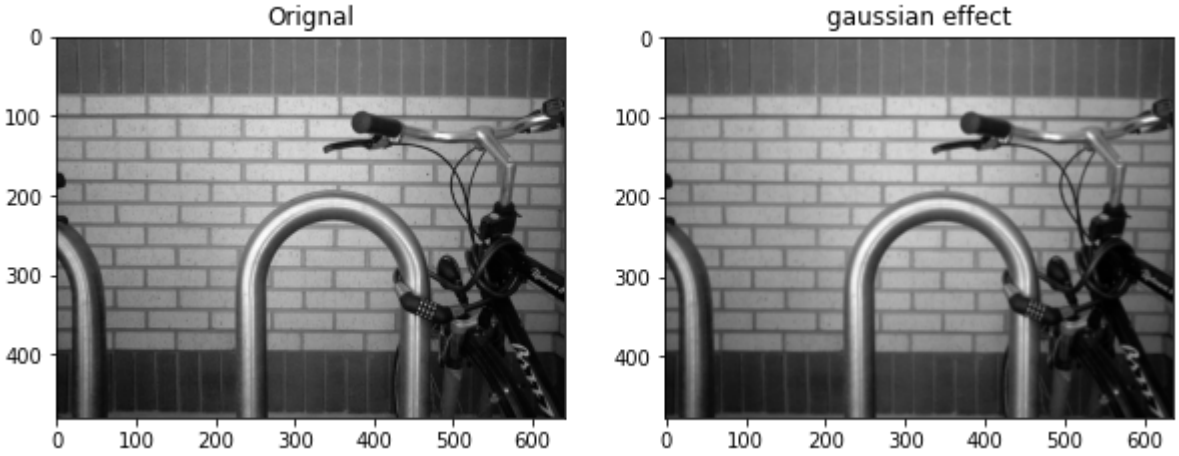
In [ ]:
```
import gaussian

gaussian_kernal = gaussian.generateGaussian(kernal_size=5, sigma = 1)
gaussian_kernal
```

Out[ ]:
```
array([[0.00291502, 0.01306423, 0.02153928, 0.01306423, 0.00291502],
       [0.01306423, 0.05854983, 0.09653235, 0.05854983, 0.01306423],
       [0.02153928, 0.09653235, 0.15915494, 0.09653235, 0.02153928],
       [0.01306423, 0.05854983, 0.09653235, 0.05854983, 0.01306423],
       [0.00291502, 0.01306423, 0.02153928, 0.01306423, 0.00291502]])
```

In [ ]:
```
gaussian_blured_image = convolve2D_single_Kernal(img, gaussian_kernal)
```

In [ ]:
```
plot_image(img, gaussian_blured_image, title_2="gaussian effect")
```



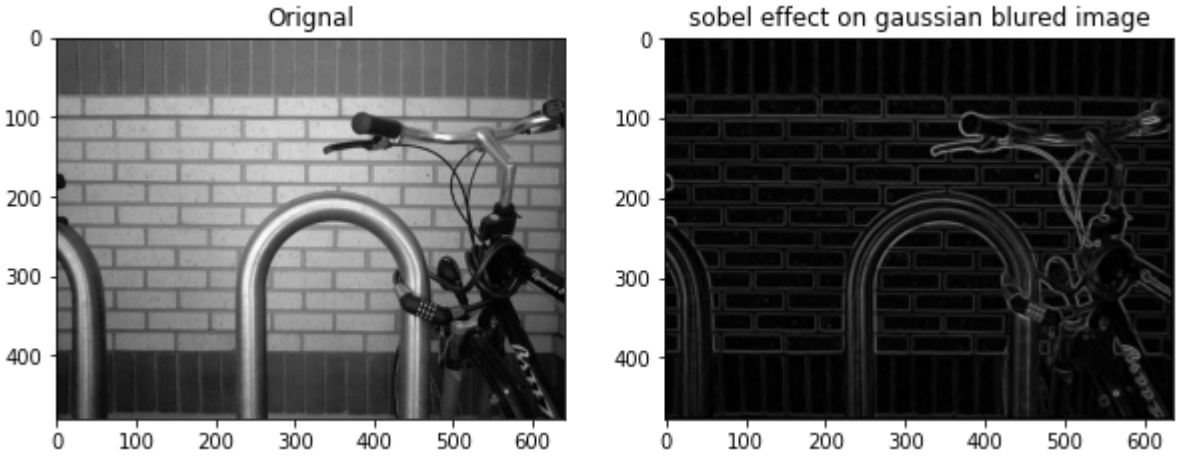
In [ ]:

```
# Edge Detection Kernel (sopel operator)
kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

# Convolve and Save Output
sobel_on_guassian_blured_image, theta = convolve2D_multi_Kernal(gaussian_blured_image, kernel_x, kernel_y, p
```

In [ ]:

```
plot_image(img, sobel_on_guassian_blured_image, title_2="sobel effect on gaussian blured image")
```



Calculating direction

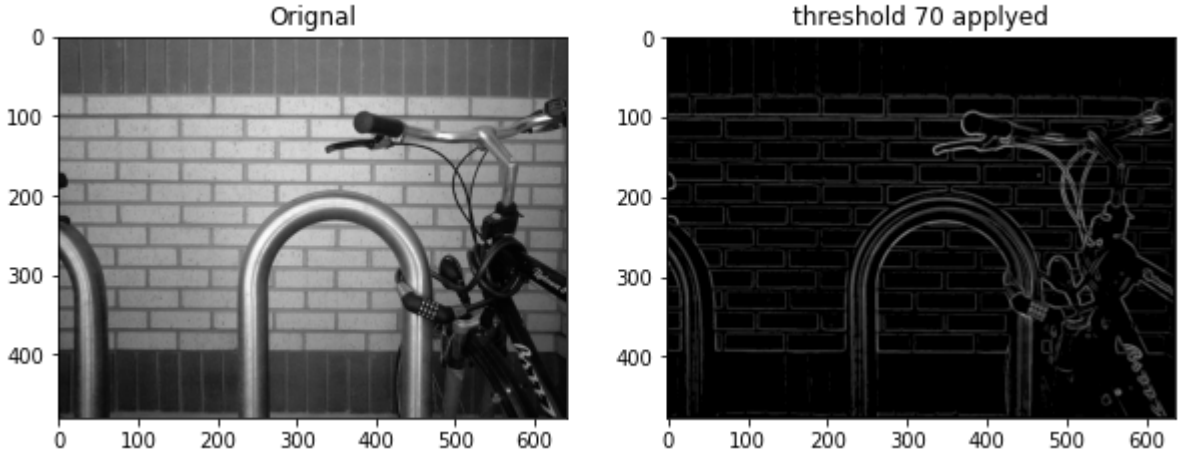
In [ ]:
```
angle = np.rad2deg(theta)
```

Applying threshold 70

In [ ]:
```
from applyThreshold import applyThreshold

threshold_applyed = applyThreshold(sobel_on_guassian_blured_image, threshold=70)
```

In [ ]:
```
plot_image(img, threshold_applyed, title_2="threshold 70 applyed")
```



Non max suppression

In [ ]:
```
# Find the neighbouring pixels (b,c) in the rounded gradient direction
# and then apply non-max suppression
M, N = threshold_applyed.shape
Non_max = np.zeros((M,N), dtype= np.uint8)

for i in range(1,M-1):
    for j in range(1,N-1):
       # Horizontal 0
        if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180) or (-22.5 <= angle[i,j] < 0) or (-180 <=
            b = sobel_on_guassian_blured_image[i, j+1]
```

```
            c = sobel_on_guassian_blured_image[i, j-1]
        # Diagonal 45
        elif (22.5 <= angle[i,j] < 67.5) or (-157.5 <= angle[i,j] < -112.5):
            b = sobel_on_guassian_blured_image[i+1, j+1]
            c = sobel_on_guassian_blured_image[i-1, j-1]
        # Vertical 90
        elif (67.5 <= angle[i,j] < 112.5) or (-112.5 <= angle[i,j] < -67.5):
            b = sobel_on_guassian_blured_image[i+1, j]
            c = sobel_on_guassian_blured_image[i-1, j]
        # Diagonal 135
        elif (112.5 <= angle[i,j] < 157.5) or (-67.5 <= angle[i,j] < -22.5):
            b = sobel_on_guassian_blured_image[i+1, j-1]
            c = sobel_on_guassian_blured_image[i-1, j+1]

        # Non-max Suppression
        if (sobel_on_guassian_blured_image[i,j] >= b) and (sobel_on_guassian_blured_image[i,j] >= c):
            Non_max[i,j] = sobel_on_guassian_blured_image[i,j]
        else:
            Non_max[i,j] = 0
```

In [ ]:

```
plot_image(img, Non_max, title_2="Non max supression effect")
```