1.3.3 Edge detection

The purpose of edge detection is to significantly reduce the image data size while retaining the original image attributes.

Canny edge detection is to extract useful structural information from different visual objects, which can greatly reduce the amount of data need to be processed. It has been widely used in various computer vision systems.

Standards for edge detection:

Detect edges with a low error rate.
The detected edge should be accurately positioned at the center of the actual edge.
The given edges in the image should only be marked once, the noise of the image should not produce false edges.

Canny edge detection algorithm can be divided into the following 5 steps:
1) Use a Gaussian filter to smooth the image and filter out noise.
2) Calculate the gradient intensity and direction of each pixel in the image.
3) Apply Non-Maximum Suppression to eliminate the spurious response caused by edge detection.
4) Apply Double-Threshold detection to determine the true edges and potential edges.
5) Finalize edge detection by suppressing isolated weak edges.

Code path:

/home/pi/Yahboom_Project/Raspbot/1.OpenCV_course/03IP_Draw_text_line_segments/ 03_1edge_detection1.ipynb

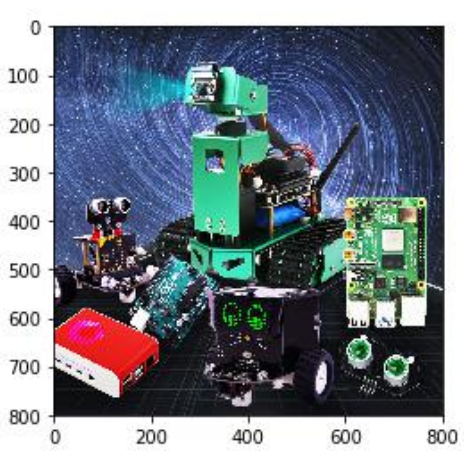
```
# Method 1
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt

img = cv2.imread('image0.jpg',1)
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
#cv2.imshow('src',img)
#canny 1 gray 2 Gaussian 3 canny
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgG = cv2.GaussianBlur(gray,(3,3),0)
dst = cv2.Canny(img,50,50) #Picture convolution--->th
# cv2.imshow('dst',dst)
```

```
    # cv2.waitKey(0)
```

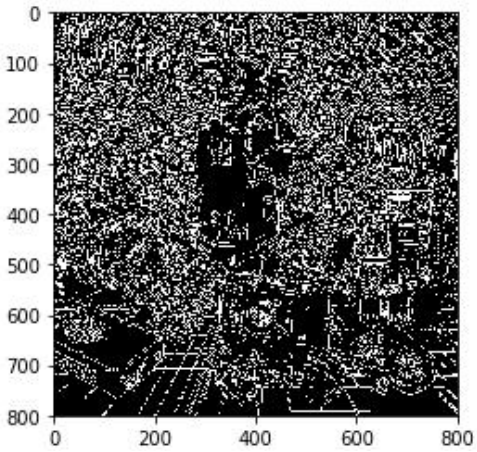
Two comparison picture

```
img_bgr2rgb1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb1)
    plt.show()
```

```
img_bgr2rgb1 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb1)
plt.show()
```



[Original picture]



[Edge detection]

Code path:

/home/pi/Yahboom_Project/1.OpenCV_course/03IP_Draw_text_line_segments

03_2edge_detection2.ipynb

```
# Method 2
import cv2
import numpy as np
import random
import math
img = cv2.imread('image0.jpg',1)
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
# cv2.imshow('src',img)
# sobel 1 Operator template 2 Picture convolution 3 Threshold decision
# [1 2 1                  [ 1 0 -1
#    0 0 0                    2 0 -2
# -1 -2 -1 ]             1 0 -1 ]

# [1 2 3 4] [a b c d] a*1+b*2+c*3+d*4 = dst
# sqrt(a*a+b*b) = f>th
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
dst = np.zeros((height,width,1),np.uint8)
for i in range(0,height-2):
```

```
        for j in range(0,width-2):
            gy = gray[i,j]*1+gray[i,j+1]*2+gray[i,j+2]*1-gray[i+2,j]*1-gray[i+2,j+1]*2-gray[i+2,j+2]*1
            gx = gray[i,j]+gray[i+1,j]*2+gray[i+2,j]-gray[i,j+2]-gray[i+1,j+2]*2-gray[i+2,j+2]
            grad = math.sqrt(gx*gx+gy*gy)
            if grad>50:
                dst[i,j] = 255
            else:
                dst[i,j] = 0
# cv2.imshow('dst',dst)
        # cv2.waitKey(0)
```

```
import matplotlib.pyplot as plt

img_bgr2rgb1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb1)
        plt.show()
```

```
img_bgr2rgb1 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb1)
plt.show()
```

After running the following program, original image and edge detection picture will be displayed in the jupyterLab control interface, as shown below.