Image processing and drawing text line segments

1. Grayscale processing

The process of converting a color image into a grayscale image is the grayscale processing of the image. The color of each pixel in the color image is determined by the three components R, G and B, and the value of each component can be 0-255. Thus, a pixel can have a color change range of more than 16 million (256 * 256 * 256 = 1677256). The gray-scale image is a special color image with the same R, G and B components, and the variation range of one pixel is 256. Therefore, in digital image processing, images of various formats are generally converted into gray-scale images to reduce the calculation amount of subsequent images. Like the color image, the description of the gray image still reflects the distribution and characteristics of the overall and local chromaticity and highlight levels of the whole image.

2. Image graying processing

Grayscale processing is the process of converting a color image into a grayscale image. The color image is divided into R, G and B components, which respectively display various colors such as red, green and blue. Graying is the process of making the R, G and B components of color equal. The pixel with large gray value is brighter (the maximum value of the pixel is 255, which is white), while the pixel with large gray value is darker (the lowest value of the pixel is 0, which is black).

The core idea of image graying is r = g = B, which is also called grayscale value.

Algorithm of image graying

1) Maximum value method: make the value of R, G and B after transformation equal to the largest one of the three values before transformation, i.e. r = g = b = max (R, G, b). The gray-scale image converted by this method has high brightness.

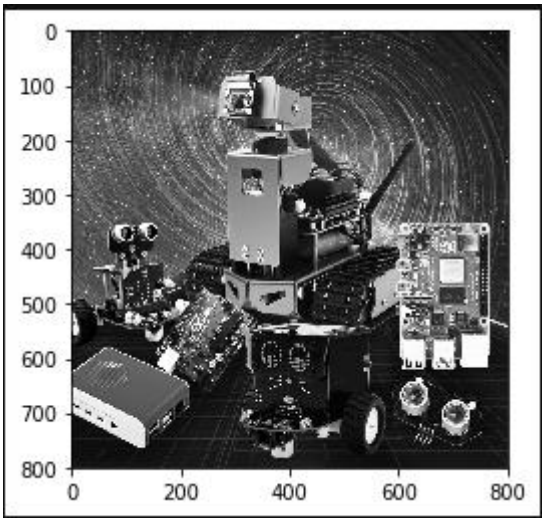
2) Average value method: the value of R, G and B after transformation is the average value of R, G and B before transformation. Namely: r = g = b = (R + G + b) / 3. The gray image produced by this method is relatively soft.

3) Weighted average method: weighted average the values of R, G and B according to a certain weight, that is, the weights of R, G and B are respectively taken to form different gray-scale images. Because the human eye is most sensitive to green, red is second, and the sensitivity to blue is the lowest, it will get a gray-scale image that is easy to recognize. Generally, the gray-scale image obtained is the best.

There are four methods to achieve grayscale in the following code:

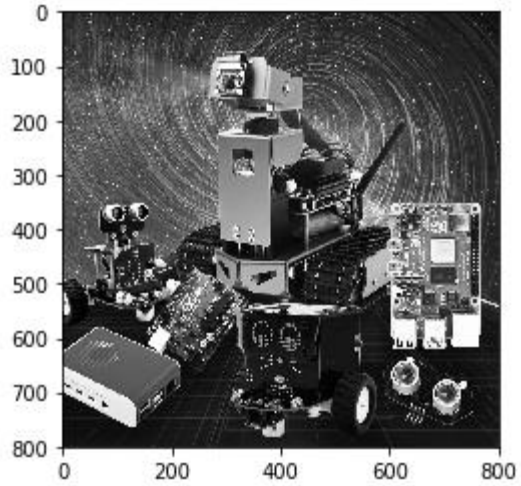
```
#way 1 imread

#Note: sometimes the first run image will not be displayed, and the second run image will be displayed



import cv2
```

```
import matplotlib.pyplot as plt



img0 = cv2.imread('yahboom.jpg',0)

img1 = cv2.imread('yahboom.jpg',1)

# print(img0.shape)

# print(img1.shape)

# cv2.imshow('src',img0)

# cv2.waitKey(0)



#original image

# img_bgr2rgb1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

# plt.imshow(img_bgr2rgb1)



#gray image

img_bgr2rgb0 = cv2.cvtColor(img0, cv2.COLOR_BGR2RGB)

    plt.imshow(img_bgr2rgb0)

    plt.show()
```



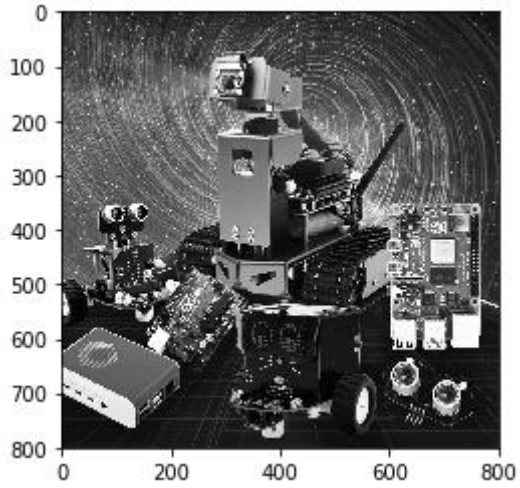

```
#way 2 cvtColor

#Note: sometimes the first run image will not be displayed, and the second run image will be displayed.



import cv2

import matplotlib.pyplot as plt



img = cv2.imread('image0.jpg',1)

dst = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)# Color space conversion 1 data 2 BGR gray

#cv2.imshow('dst',dst)

#cv2.waitKey(0)



# original image

# img_bgr2rgb1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
# plt.imshow(img_bgr2rgb1)



#gray image

img_bgr2rgb0 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

        plt.imshow(img_bgr2rgb0)

        plt.show()
```



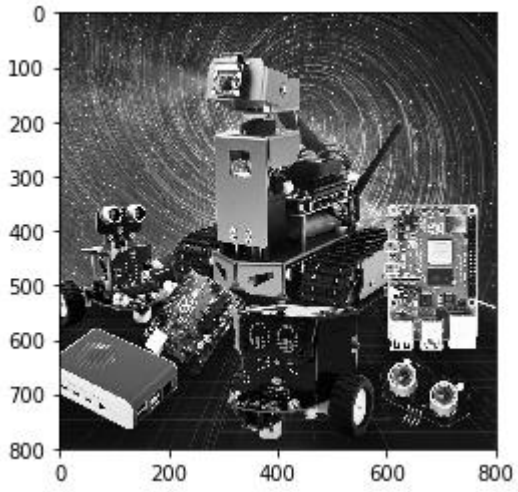

```
#way3 Average value method

import cv2

import numpy as np

import matplotlib.pyplot as plt




img = cv2.imread('yahboom.jpg',1)

imgInfo = img.shape

height = imgInfo[0]
```

```
width = imgInfo[1]

# RGB R=G=B = gray    (R+G+B)/3

dst = np.zeros((height,width,3),np.uint8)

for i in range(0,height):

        for j in range(0,width):

                (b,g,r) = img[i,j]

                gray = (int(b)+int(g)+int(r))/3

                dst[i,j] = np.uint8(gray)

#cv2.imshow('dst',dst)

#cv2.waitKey(0)



#original image

# img_bgr2rgb1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# plt.imshow(img_bgr2rgb1)



#gray image

img_bgr2rgb0 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.imshow(img_bgr2rgb0)

        plt.show()
```




```
#way 4 Weighted average method

# gray = r*0.299+g*0.587+b*0.114



import cv2

import numpy as np

img = cv2.imread('yahboom.jpg',1)

imgInfo = img.shape

height = imgInfo[0]

width = imgInfo[1]

dst = np.zeros((height,width,3),np.uint8)

for i in range(0,height):

    for j in range(0,width):

        (b,g,r) = img[i,j]

        b = int(b)

        g = int(g)
```

```
            r = int(r)
            gray = r*0.299+g*0.587+b*0.114
            dst[i,j] = np.uint8(gray)
#cv2.imshow('dst',dst)
#cv2.waitKey(0)


#original image
# img_bgr2rgb1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# plt.imshow(img_bgr2rgb1)


#gray image
img_bgr2rgb0 = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb0)
      plt.show()
```



3.image binaryzation

The core idea of binarization is to set the threshold value, and the value greater than the threshold

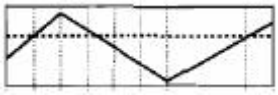
value is 0 (black) or 255 (white), so that the image is called a black-and-white image. The threshold can be fixed or adaptive.

The adaptive threshold is generally the comparison of the average value of the pixels in a point and the weighted sum of the Gaussian distribution of the pixels in the region where the point is in the middle order. A difference value may or may not be set.

Global threshold:

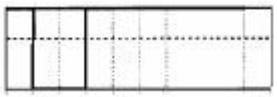
Python opencv provides a threshold function：

cv2.threshold（src, threshold, maxValue, method）

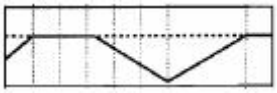


src Original: broken line is the value to be thresholded; Dashed line is the threshold

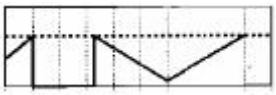
cv2.THRESH_BINARY： The grayscale value of the pixel point larger than the threshold value is set to maxvalue (for example, the maximum 8-bit grayscale value is 255), and the grayscale value of the pixel point smaller than the threshold value is set to 0.



cv2.THRESH_BINARY_INV ： The grayscale value of the pixel point larger than the threshold value is set to 0, and the grayscale value of the pixel point smaller than the threshold value is set to maxvalue.



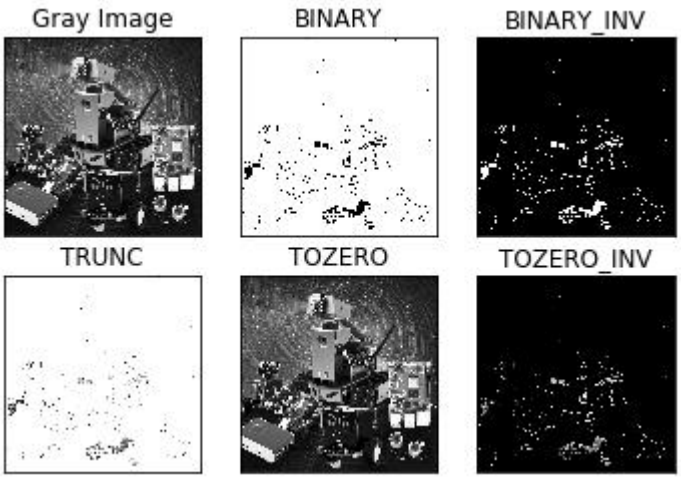
cv2.THRESH_TRUNC： The gray value of the pixel point is less than the threshold value and does not change, and the pixel point whose gray value is greater than the threshold value is set as the threshold value.



cv2.THRESH_TOZERO： If the gray value of a pixel is less than the threshold value, no change will be made, and if the gray value is greater than the threshold value, all the gray values will become 0.

cv2.THRESH_TOZERO_INV： If the gray value of a pixel point is greater than the threshold value, no change will be made. If the gray value of a pixel point is less than the threshold value, all the gray values will become 0.

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)

GrayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)     #Convert to grayscale image



ret,thresh1=cv2.threshold(GrayImage,10,255,cv2.THRESH_BINARY)

ret,thresh2=cv2.threshold(GrayImage,10,255,cv2.THRESH_BINARY_INV)

ret,thresh3=cv2.threshold(GrayImage,10,255,cv2.THRESH_TRUNC)

ret,thresh4=cv2.threshold(GrayImage,10,255,cv2.THRESH_TOZERO)

ret,thresh5=cv2.threshold(GrayImage,10,255,cv2.THRESH_TOZERO_INV)

titles = ['Gray Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']

images = [GrayImage, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

    plt.title(titles[i])

    plt.xticks([]),plt.yticks([])

plt.show()
```



4.edge detection

The purpose of edge detection is to significantly reduce the data size of the image while retaining the original image attributes. At present, there are many algorithms for edge detection. Although Canny algorithm has a long history, it can be said that it is a standard algorithm for edge detection and is still widely used in research.

Canny edge detection is a technology to extract useful structural information from different visual objects and greatly reduce the amount of data to be processed. It has been widely used in various computer vision systems. Canny found that the requirements for edge detection are similar in different vision systems. Therefore, an edge detection technology with wide application significance can be realized. General standards for edge detection include:

Detecting edges at a low error rate means that as many edges as possible in the captured image need to be captured as accurately as possible.

The detected edge shall be accurately positioned at the center of the real edge.

A given edge in an image should be marked only once, and where possible, the noise of the image should not produce false edges.

To meet these requirements, canny used the variational method. The optimal function in canny detector is described by the sum of four exponential terms, which can be approximated by the first derivative of Gaussian function.

Among the commonly used edge detection methods, Canny edge detection algorithm is one of the methods with strict definition and can provide good and reliable detection. Because it has the advantages of satisfying the three standards of edge detection and simple implementation process, it has become one of the most popular algorithms for edge detection.

Canny edge detection algorithm can be divided into the following 5 steps:

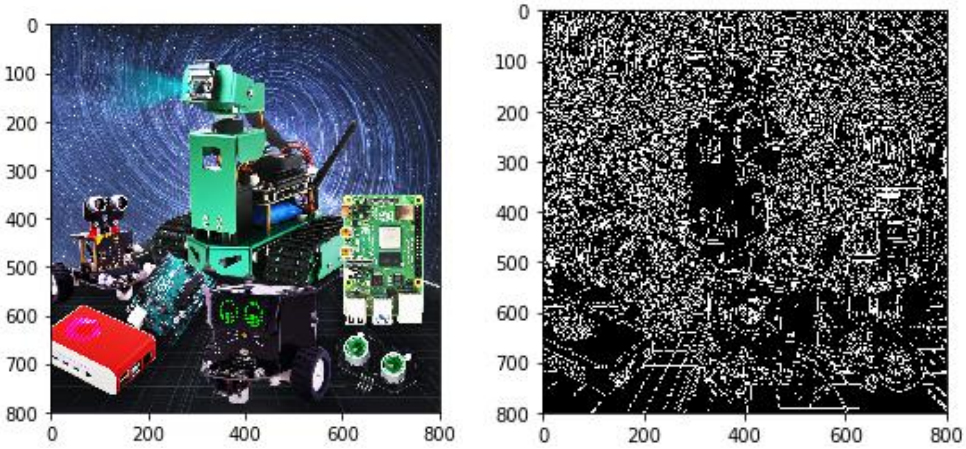
1. Gaussian filter is used to smooth the image and filter out noise.
2. The gradient intensity and direction of each pixel in the image are calculated.
3. Non maximum suppression is applied to eliminate spurious response caused by edge detection.
4. Double threshold detection is applied to determine real and potential edges.
5. The edge detection is finally completed by suppressing isolated weak edges.

```
#way 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)
```

Compare the two figures

```
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()
```



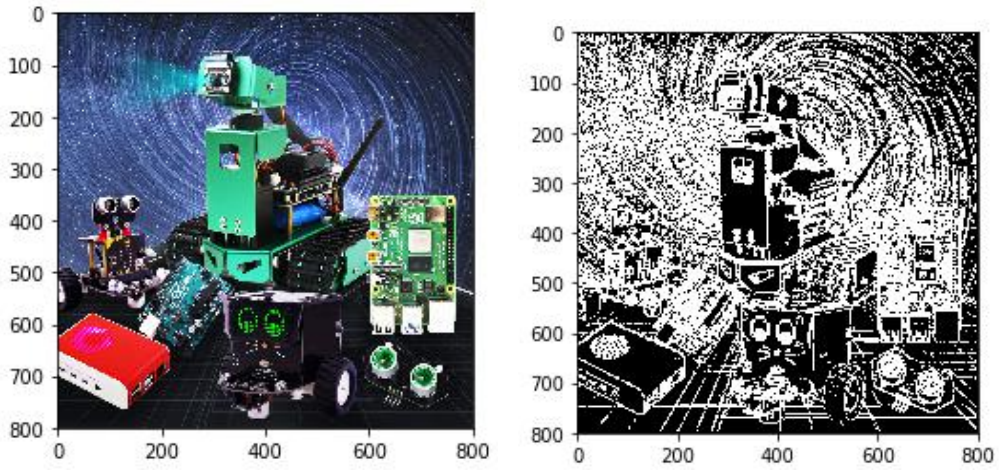

```
#way 2

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()
```



5.Line segments: Drawing

When using OpenCV to process images, we sometimes need to draw line segments and rectangles on the images. Opencv uses the line (DST, pT1, pT2, color, thickness = none, linetype = none, shift = none) function to draw line segments.

Parameter Description:

DST: output image.

PT1, pT2: required parameter. The coordinate point of the line segment, which represents the starting point and the ending point respectively

Color: required parameter. Used to set the color of line segments

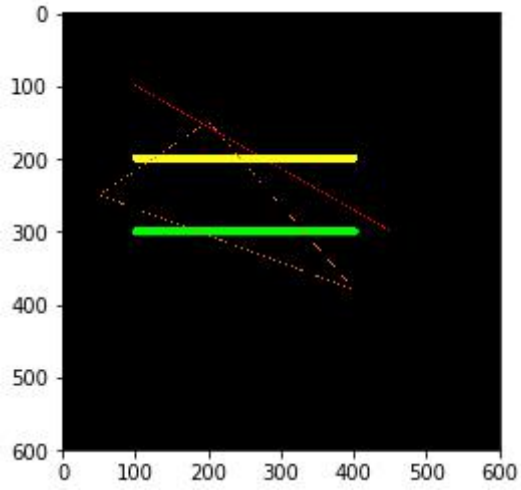
Thickness: optional parameter. Used to set the width of the line segment

Linetype: optional parameter. Used to set the type of line segment. 8 (8 adjacent connectors - default), 4 (4 adjacent connectors) and cv2.line are available_ AA is anti aliasing

```
import cv2

import numpy as np
```

```
import matplotlib.pyplot as plt



newImageInfo = (600, 600, 3)

dst = np.zeros(newImageInfo,np.uint8)



# line

# Draw line segments 1 dst 2 begin 3 end 4 color

cv2.line(dst, (100,100), (450,300), (0,0,255))

# 5 line w

cv2.line(dst, (100,200), (400,200), (0,255,255), 10)

# 6 line type

cv2.line(dst, (100,300), (400,300), (0,255,0), 10, cv2.LINE_AA)



cv2.line(dst, (200,150), (50,250), (25,100,255))

cv2.line(dst, (50,250), (400,380), (25,100,255))

cv2.line(dst, (400,380), (200,150), (25,100,255))



# cv2.imshow('dst',dst)

# cv2.waitKey(0)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.imshow(dst)
```

```
plt.show()
```



6.Rectangle circle drawing

1)Rectangle drawing

rectangle（img，pt1，pt2，color，thickness=None，lineType=None，shift=None）

Parameter description:

img：Canvas or carrier image。

pt1, pt2： Required parameter. The vertices of the rectangle represent the vertices and diagonal vertices respectively, that is, the upper left corner and the lower right corner of the rectangle (these two vertices can determine a unique rectangle)

color： Required parameter.Used to set the color of the rectangle

thickness： Optional parameters.Used to set the width of the rectangle side. When the value is negative, it means to fill the rectangle

lineType： Optional parameters.Used to set the type of line segment. 8 (8 adjacent connectors - default), 4 (4 adjacent connectors) and cv2.line are available_ AA is anti aliasing

2)Drawing of circles

cv2.circle(img, center, radius, color[,thickness[,lineType]])

Parameter Description:

img:Canvas or carrier image

center：Is the center coordinate, format: (50,50)

radius：radius

color：color

thickness： Line thickness. The default is 1. If - 1, it is filled solid.

lineType：Line type. The default is 8, and the connection type. The following table describes:

parameter	explain

cv2.FILLED	fill
cv2.LINE_4	4 connection type
cv2.LINE_8	8 connection type
cv2.LINE_AA	Antialiasing, this parameter will make the line smoother

3)Draw ellipse
cv2.ellipse(img, center, axes, angle, StartAngle, endAngle, color[,thickness[,lineType]])

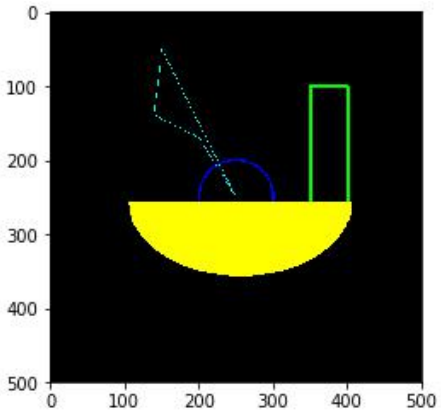
center：Center point of ellipse，（x，x）
axes：Refers to short radius and long radius，（x，x）
angle：Refers to the angle of counterclockwise rotation
StartAngle：Angle of arc starting angle
endAngle：Angle of arc end angle
img、color　Refer to the description of the circle.
#The fifth parameter refers to the angle at which the drawing starts counterclockwise, and the sixth parameter refers to the angle at which the drawing ends counterclockwise.
#If a symbol is added to the four, five and six parameters, it indicates the opposite direction, i.e. clockwise direction.

4)draw a polygon
cv2.polylines(img,[pts],isClosed, color[,thickness[,lineType]])
pts：Vertices of a polygon
isClosed：Whether it is closed.（True/False）
Other parameters: refer to the drawing parameters of the circle

```
import cv2

import numpy as np

newImageInfo = (500,500,3)

dst = np.zeros(newImageInfo,np.uint8)

#    1 2 top left corner 3 Lower right corner 4 5 fill -1 >0 line w
```

```
cv2.rectangle(dst,(350,100),(400,270),(0,255,0),3)

# 2 center 3 r

cv2.circle(dst,(250,250),(50),(255,0,0),2)

# 2 center 3 axis 4 angle 5 begin 6 end 7

cv2.ellipse(dst, (256,256), (150,100), 0, 0, 180, (0,255,255), -1)



points = np.array([[150,50], [140,140], [200,170], [250,250], [150,50]], np.int32)

#print(points.shape)

points = points.reshape((-1,1,2))

#print(points.shape)

cv2.polylines(dst,[points],True,(255,255,0))

# cv2.imshow('dst',dst)

        # cv2.waitKey(0)
```

```
import matplotlib.pyplot as plt



dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.imshow(dst)

plt.show()
```



7. Text picture drawing

function： cv2.putText(img, str, origin, font, size,color,thickness)

The parameters are: picture, added text, upper left corner coordinate (integer), font, font size, color, font thickness

```
import cv2

import numpy as np



img = cv2.imread('yahboom.jpg',1)

font = cv2.FONT_HERSHEY_SIMPLEX



cv2.rectangle(img,(200,100),(500,400),(0,255,0),3)

# 1 dst 2 Text content 3 coordinate 4 5 font size 6 color 7 thickness 8 line type

cv2.putText(img,'Yahboom',(250,50),font,1,(200,200,0),2,cv2.LINE_AA)

# cv2.imshow('src',img)

# cv2.waitKey(0)
```

```
import matplotlib.pyplot as plt
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)

plt.show()
```