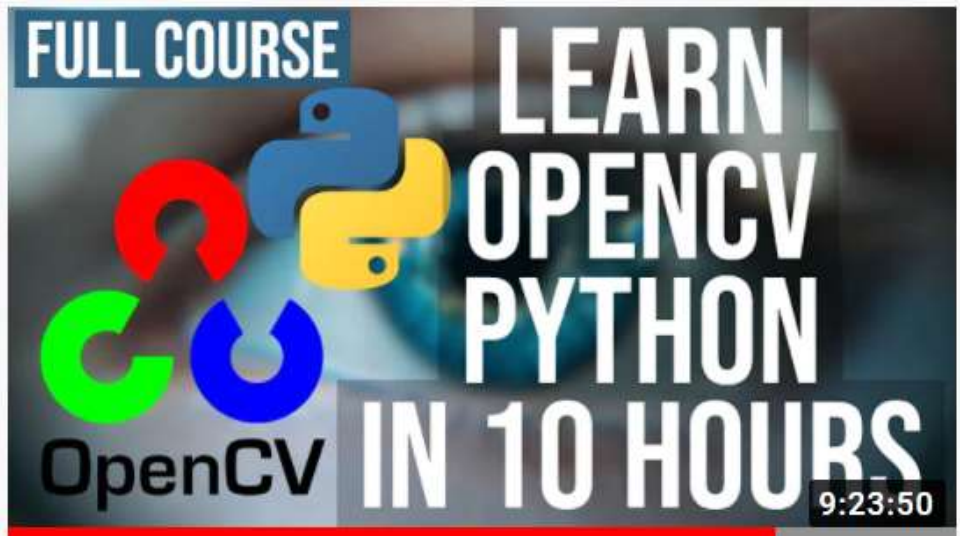
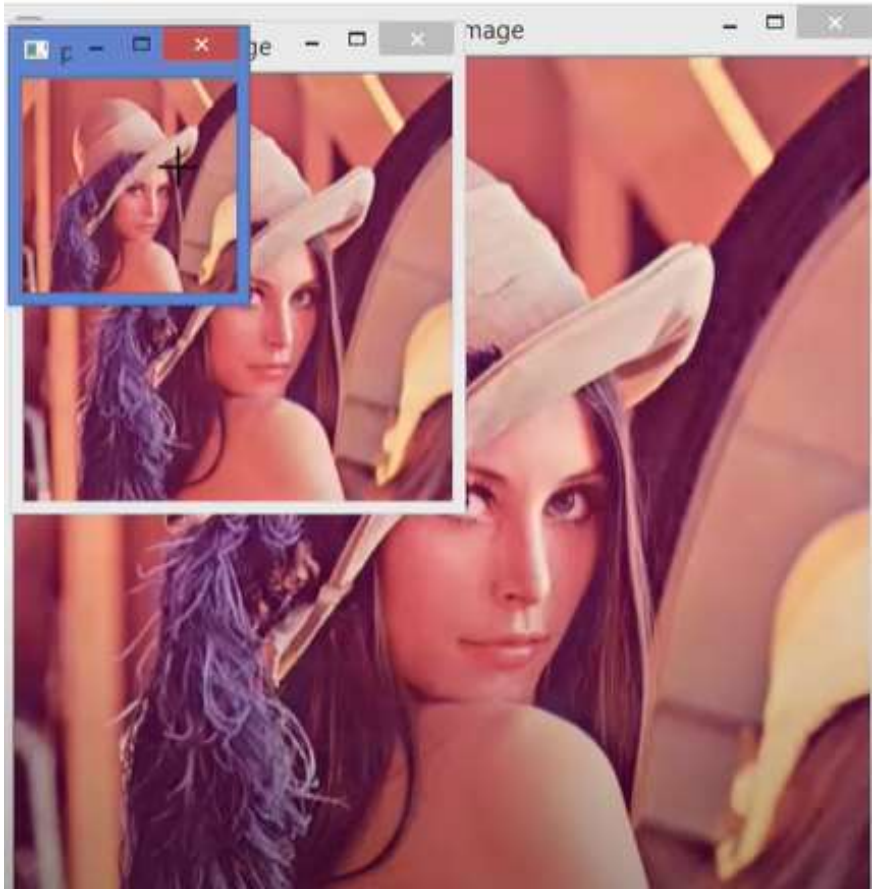
OpenCV Python Tutorial For Beginners (10 hrs)

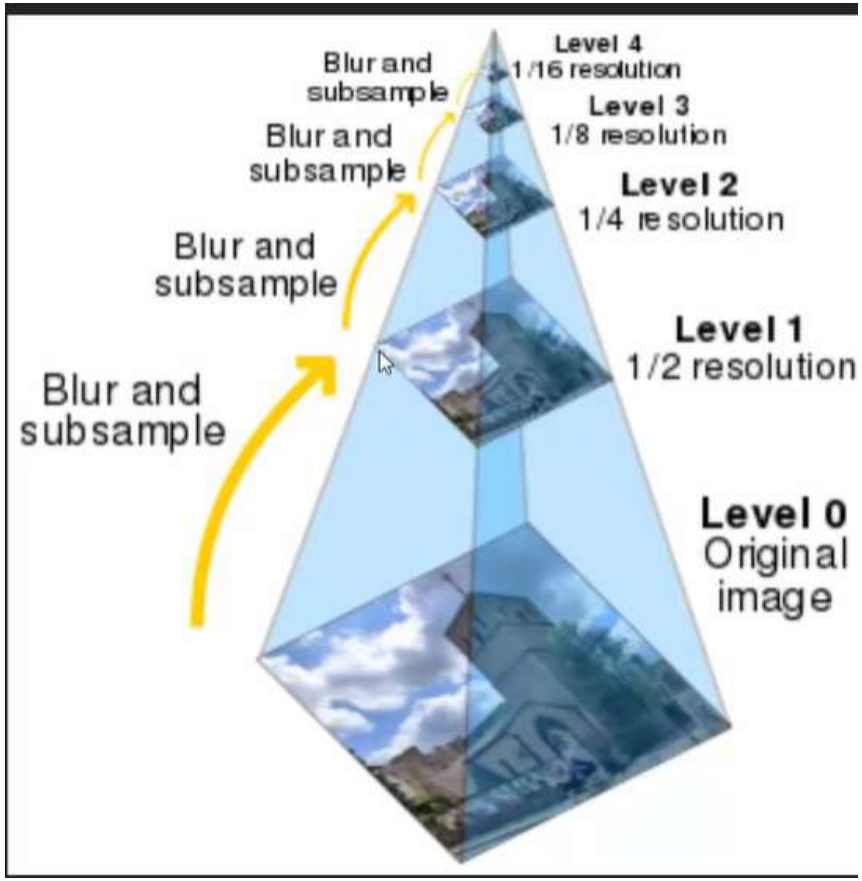
https://www.youtube.com/watch?v=N81PCpADwKQ&t=27645s

https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1Qu1Wo1RIa7D1O6skqDQ-JZ1GGHKK-K



21. Image Pyramids





¼ area

21. Image Pyramids

To convert an image to a size different than its original: (1) resize; (2) image pyramid

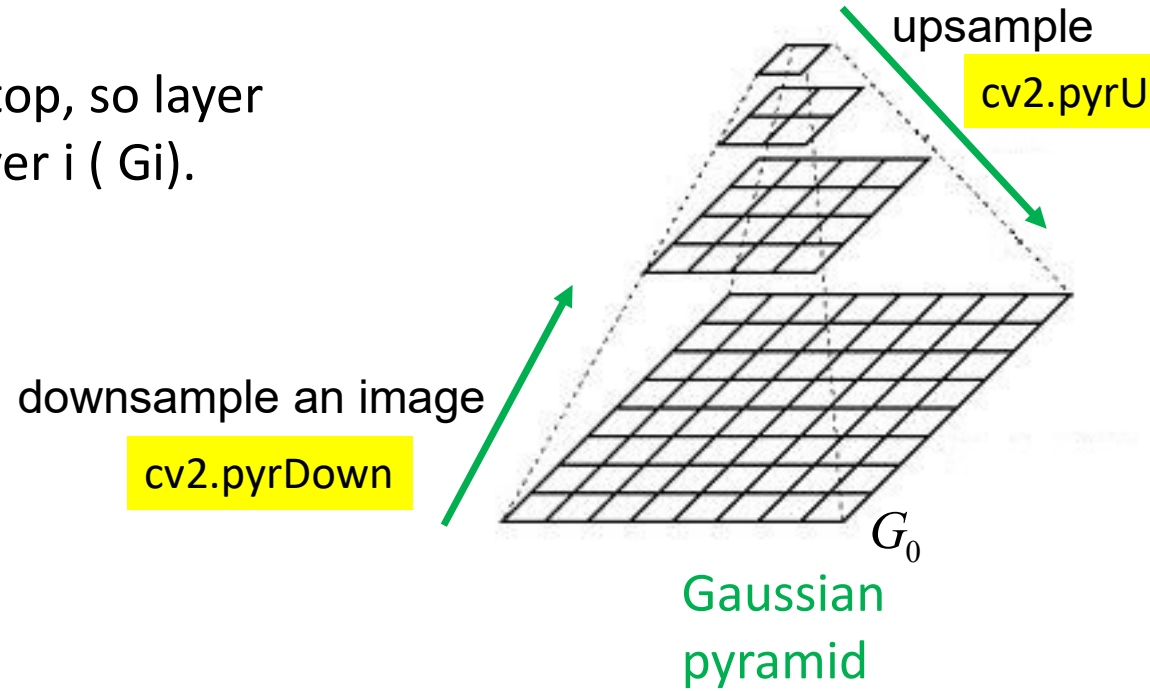
There are two kinds of Image Pyramid.
1) Gaussian Pyramid and 2) Laplacian Pyramid
Gaussian pyramids use cv.pyrDown() and cv.pyrUp() functions to downsample (zoom out) or upsample (zoom in) a given image.

Image Pyramid:
An image pyramid is a collection of images - all arising from a single original image - that are successively downsampled until some desired stopping point is reached.
Gaussian pyramid: Used to downsample images
Laplacian pyramid: Used to reconstruct an upsampled image from an image lower in the pyramid (with less resolution)

Every layer is numbered from bottom to top, so layer (i+1) (denoted as Gi+1) is smaller than layer i (Gi).

Convolve Gi with a Gaussian kernel:

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$



Remove every even-numbered row and column to obtain Gi+1.

The resulting image will be exactly one-quarter (1/4) the area of its predecessor.

Note: When we reduce the size of an image, we are actually losing information of the image.

cv2.pyrDown()函數接受3個參數：

src: 當前圖像，初始化為原圖像
dst: 目的圖像(顯示圖像，為輸入圖像的一半)
dstsize(src.cols/2, src.rows/2) :目的圖像大小， 既然我們是向下採樣

dst = cv2.pyrDown(src, dstsize =(cols//2, rows//2))

Python's map(function, iterable, ...)
map() is a built-in function that allows you to process and transform all the items in an iterable without using an explicit for loop, a technique commonly known as mapping. map() is useful when you need to apply a transformation function to each item in an iterable and transform them into a new iterable.

map(function, iterable, ...)

Apply function to every item of iterable and return a list of the results.

Ex1

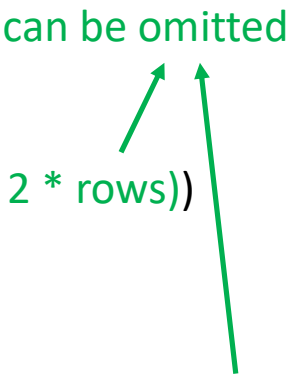
```
listA = ['1','2','3']
print(list(map(int,listA)))  # [1, 2, 3]
a,b,c = map(int, listA) # a=1, b=2, c=3
```

EX2

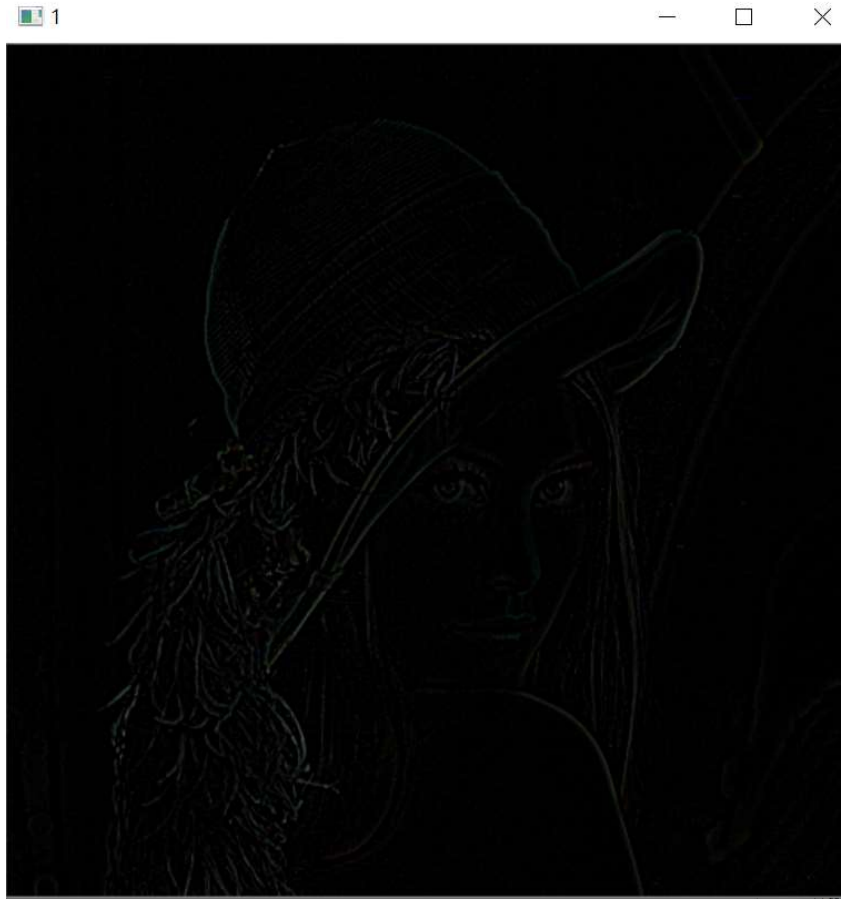
```
def multiple2(x):
  return x*2

list1 = [1,3,5,7,9]
print(list(map(multiple2,list1))) # [2, 6, 10, 14, 18]
```

ex26.py

```
import cv2
img = cv2.imread("lena.jpg")
print(img.shape) # (512, 512, 3)

print("""
      Zoom In-Out demo
      ----------------
      * [i] -> Zoom [i]n
      * [o] -> Zoom [o]ut
      * [ESC] -> Close program
      """)

while True:
   rows, cols, _channels = map(int, img.shape)
   cv2.imshow('Pyramids Demo', img)
   k = cv2.waitKey(0)
   if k == 27:
      break

   elif chr(k) == 'i':
      img = cv2.pyrUp(img, dstsize=(2 * cols, 2 * rows))
      print ('** Zoom In: Image x 2')

   elif chr(k) == 'o':
      img = cv2.pyrDown(img, dstsize=(cols // 2, rows // 2))
      print ('** Zoom Out: Image / 2')

cv2.destroyAllWindows()
```



Laplacian Pyramid:
looks like an edge detection



A level in Laplacian Pyramid is formed by the difference between that level in Gaussian Pyramid and expanded version of its upper level in Gaussian Pyramid.

ex27.py

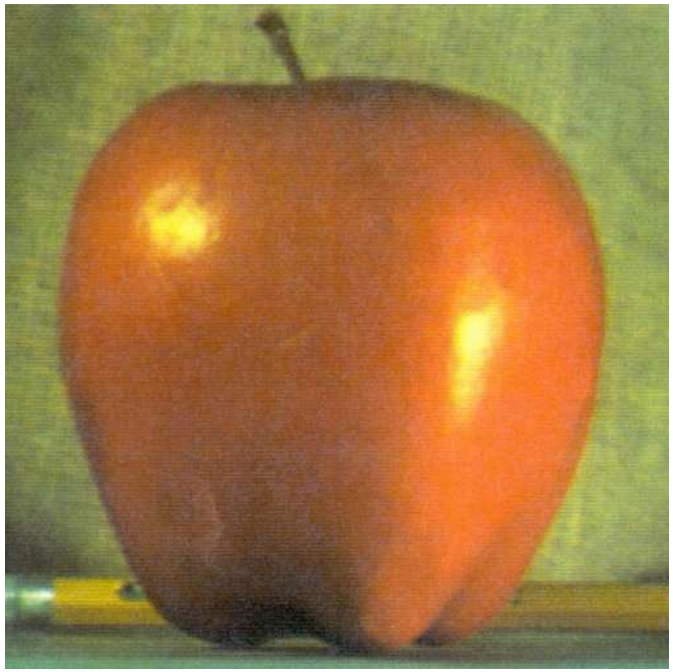
```
import cv2
img = cv2.imread("lena.jpg")
layer = img.copy()
gp = [layer]  # Gaussian Pyramid
for i in range(6):
    layer = cv2.pyrDown(layer)
    gp.append(layer)  # List
    #cv2.imshow(str(i), layer)

layer = gp[5]
cv2.imshow('upper level Gaussian Pyramid', layer)
lp = [layer] # Laplacian Pyramid

for i in range(5, 0, -1): # 5, 4, 3, 2, 1
    gaussian_extended = cv2.pyrUp(gp[i])
    laplacian = cv2.subtract(gp[i-1], gaussian_extended)
    lp.append(laplacian)
    cv2.imshow(str(i), laplacian)

cv2.imshow("Original image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

512, 256, 128, 64, 32, 16, 8
Gaussian pyramid

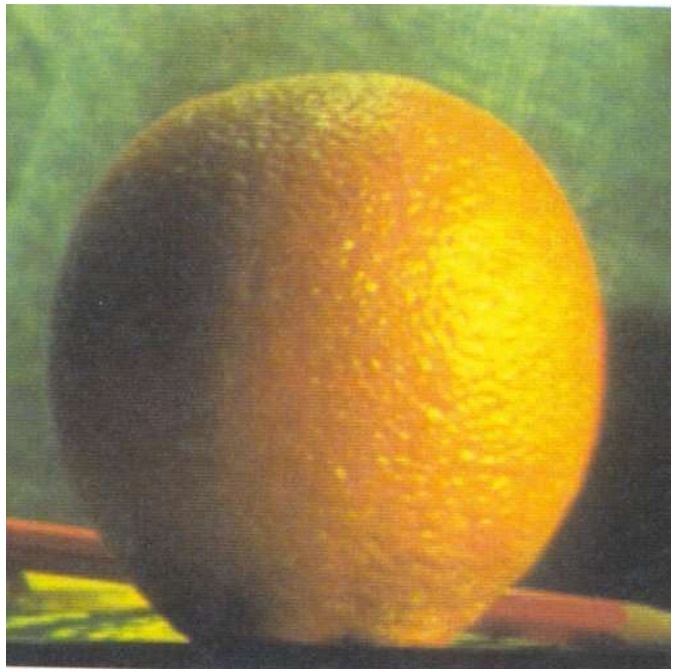
16, 32, 64, 128, 256, 512
Laplacian pyramid

22. Image Blending using Pyramids 圖像融合

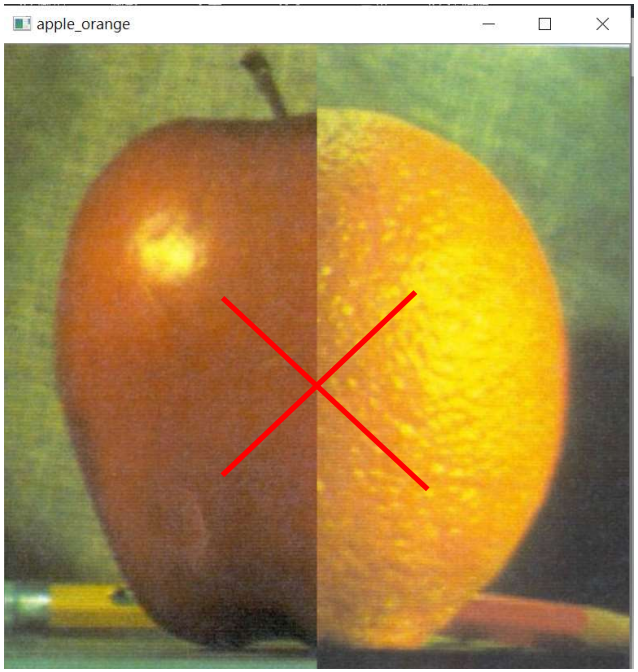
Joint the left half of apple with the right half of orange = ?



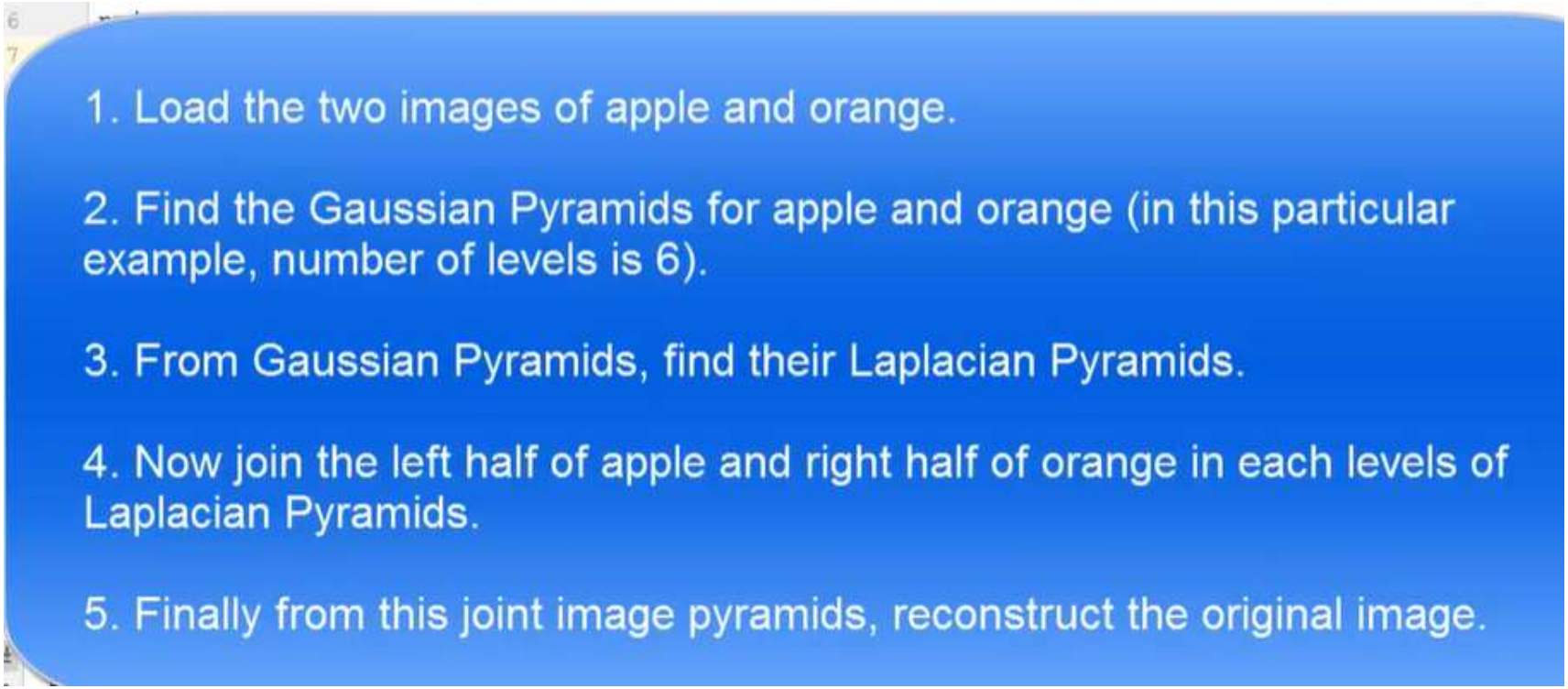
+



=



Five steps for image blending



ex28.py

```python
import cv2
import numpy as np
apple = cv2.imread('apple.jpg')   #(512, 512, 3)
orange = cv2.imread('orange.jpg') #(512, 512, 3)
print(apple.shape)
print(orange.shape)
apple_orange = np.hstack((apple[:, :256], orange[:, 256:]))

# generate Gaussian pyramid for apple (7 layers)
apple_copy = apple.copy()
gp_apple = [apple_copy]
for i in range(6):
    apple_copy = cv2.pyrDown(apple_copy)
    gp_apple.append(apple_copy)

# generate Gaussian pyramid for orange (7 layers)
orange_copy = orange.copy()
gp_orange = [orange_copy]
for i in range(6):
    orange_copy = cv2.pyrDown(orange_copy)
    gp_orange.append(orange_copy)
```

512, 256, 128, 64, 32, 16, 8
Gaussian pyramid

ex28.py

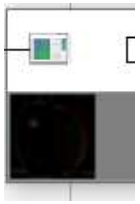
16, 32, 64, 128, 256, 512
Laplacian pyramid

```python
# generate Laplacian Pyramid for apple  (6 layers)
apple_copy = gp_apple[5]  #(16,16,3)
lp_apple = [apple_copy]
for i in range(5, 0, -1):  # 5,4,3,2,1
    gaussian_expanded = cv2.pyrUp(gp_apple[i])
    laplacian = cv2.subtract(gp_apple[i-1], gaussian_expanded)
    lp_apple.append(laplacian)

# generate Laplacian Pyramid for orange (6 layers)
orange_copy = gp_orange[5] #(16,16,3)
lp_orange = [orange_copy]
for i in range(5, 0, -1): # 5,4,3,2,1
    gaussian_expanded = cv2.pyrUp(gp_orange[i])
    laplacian = cv2.subtract(gp_orange[i-1], gaussian_expanded)
    lp_orange.append(laplacian)

# Now add left and right halves of images in each level
apple_orange_pyramid = [] # (6 layers) (16, 32, 64, 128, 256, 512)
n = 0
for apple_lap, orange_lap in zip(lp_apple, lp_orange):
    n += 1
    rows, cols, ch = apple_lap.shape
    laplacian = np.hstack((apple_lap[:, 0:int(cols/2)], orange_lap[:, int(cols/2):]))
    apple_orange_pyramid.append(laplacian)
```

apple_orange_pyramid[0]

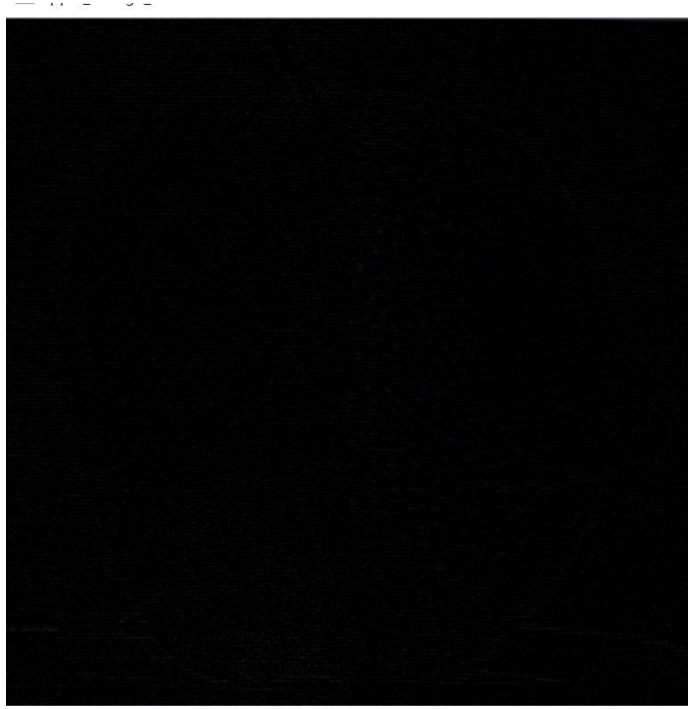
(16x16x3)

apple_orange_pyramid[1]

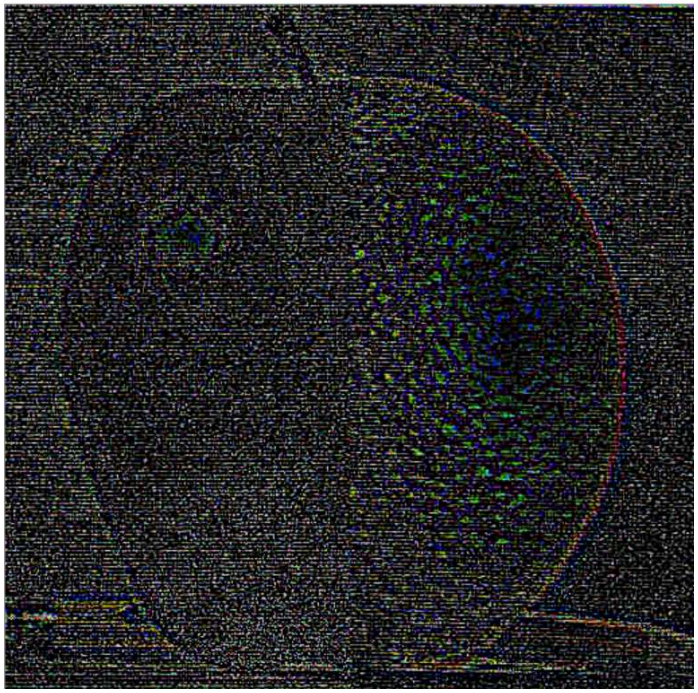


(32x32x3)

apple_orange_pyramid[5] (512x512x3)



*20=



16, 32, 64, 128, 256, 512

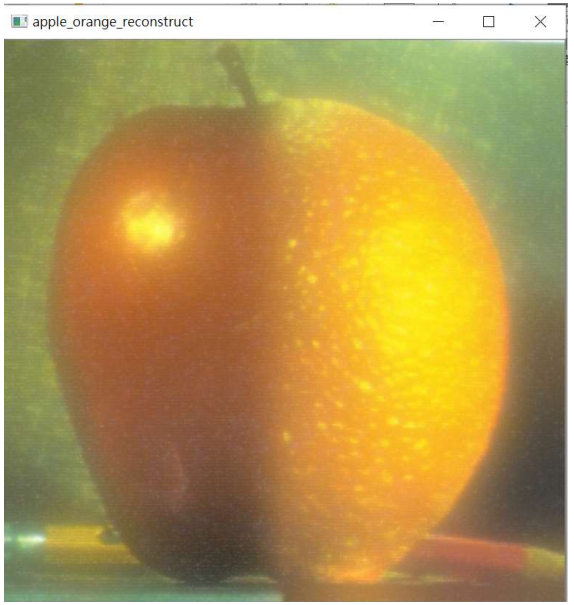
Laplacian pyramid

ex28.py

```
55 # now reconstruct
56 apple_orange_reconstruct = apple_orange_pyramid[0]
57 for i in range(1, 6):    # 1,2,3,4,5
58     apple_orange_reconstruct = cv2.pyrUp(apple_orange_reconstruct)
59     apple_orange_reconstruct = cv2.add(apple_orange_pyramid[i], apple_orange_reconstruct)
60
61 cv2.imshow("apple", apple)
62 cv2.imshow("orange", orange)
63 cv2.imshow("apple_orange", apple_orange)
64 cv2.imshow("apple_orange_reconstruct", apple_orange_reconstruct)
65 cv2.waitKey(0)
66 cv2.destroyAllWindows()
```

16, 32, 64, 128, 256, 512
Laplacian pyramid

apple_orange_pyramid[0]



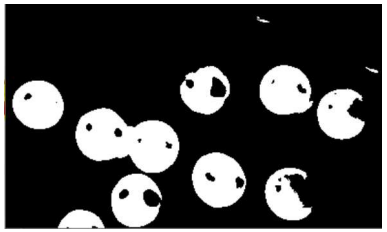
Lp(n,n) = Gp(n,n) - pryUP(Gp(n/2,n/2))

23. Find and Draw Contours 輪廓線 cv2.findContours; cv2.drawContours

Contours: simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
three arguments:
first one is source image: For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
second is contour retrieval mode
third is contour approximation method.

In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.



23. Find and Draw Contours

```python
import numpy as np
import cv2

img = cv2.imread('opencv-logo.png')
imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(imgray, 127, 255, 0)
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

cv2.imshow('Image', img)
cv2.imshow('Image GRAY', imgray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.THRESH_BINARY

contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

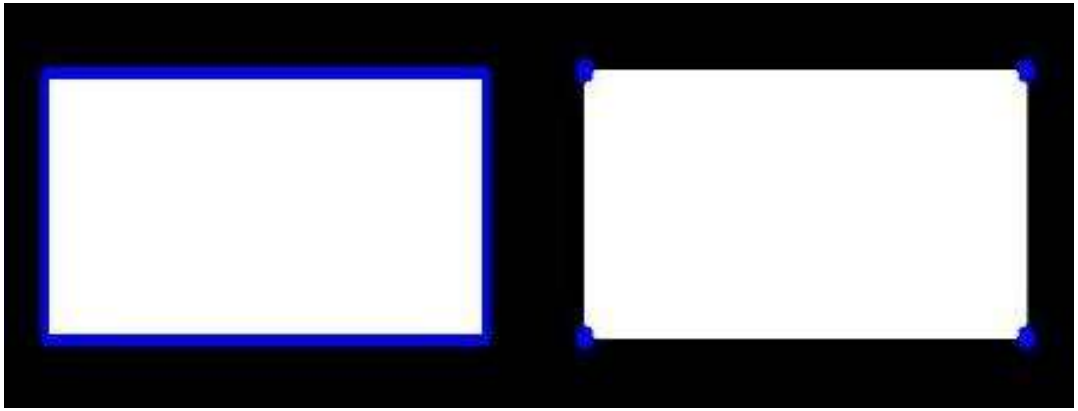
cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

Contour Approximation Method: the third argument in cv2.findContours function.

cv2.CHAIN_APPROX_NONE: all the boundary points are stored

cv2.CHAIN_APPROX_SIMPLE:It removes all redundant points and compresses the contour, thereby saving memory.

cv2.CHAIN_APPROX_NONE cv2.CHAIN_APPROX_SIMPLE



To draw all the contours in an image:
img = cv2.drawContours(img, contours, -1, (0,255,0), 3)

To draw an individual contour, say 4th contour:
img = cv2.drawContours(img, contours, 3, (0,255,0), 3)

cv2.drawContours:
first argument is source image,
second argument is the contours,
third argument is index of contours (useful when drawing individual contour.
To draw all contours, pass -1)
and remaining arguments are color, thickness.

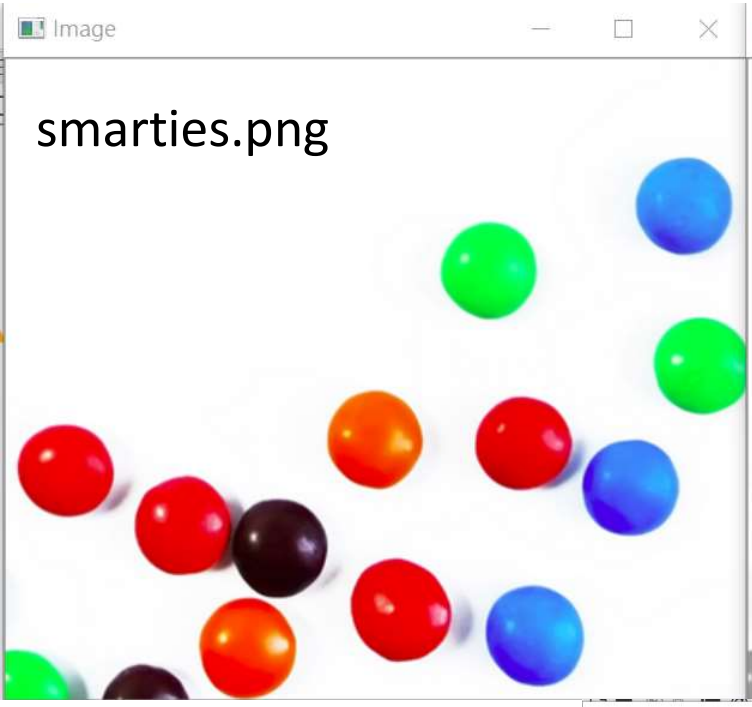


Image
smarties.png

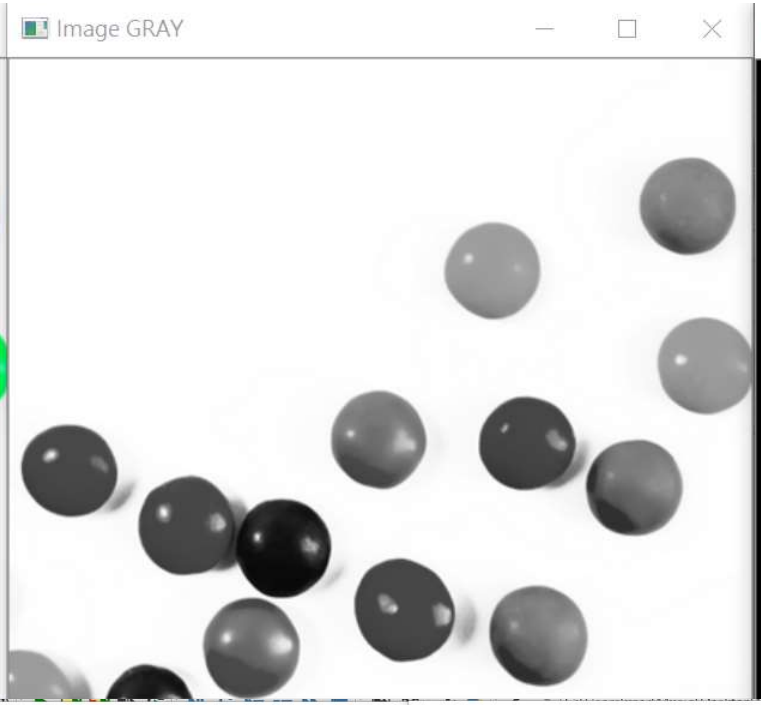


Image GRAY

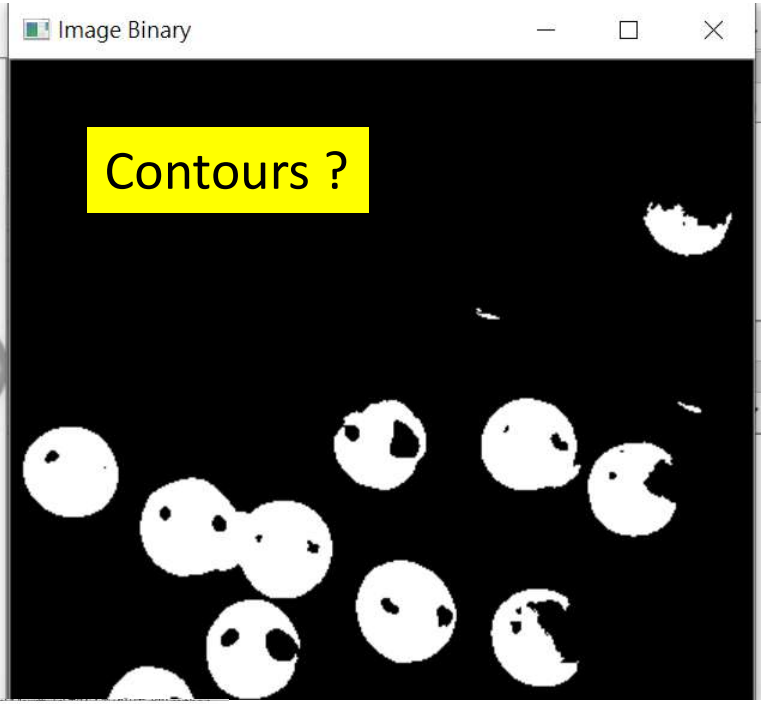
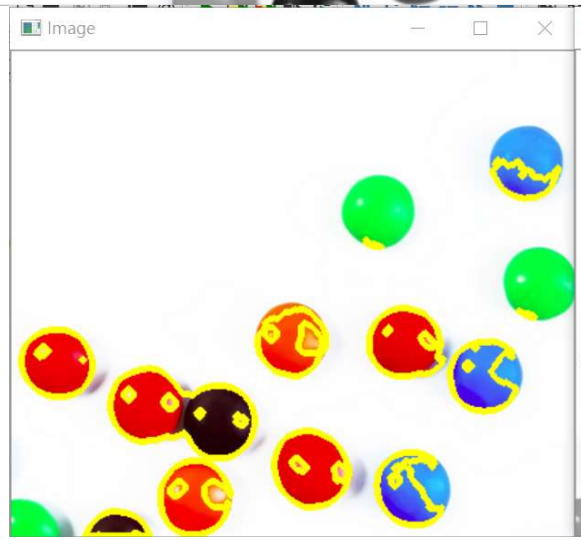


Image Binary
Contours ?

+Contours



Image


Image GRAY

ex29.py

Number of contours = 31

```
 8 import numpy as np
 9 import cv2
10
11 img = cv2.imread('smarties.png')
12 imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13 ret, thresh = cv2.threshold(imgray, 127, 255, 1) # cv2.THRESH_BINARY_INV
14 contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
15 print("Number of contours = " + str(len(contours)))
16 print(contours[0])
17
18 cv2.drawContours(img, contours, -1, (0, 255, 255), 3) # yellow contour
19 cv2.drawContours(imgray, contours, -1, (0, 255, 255), 3)
20 cv2.drawContours(thresh, contours, -1, (0, 255, 255), 3)
21
22 cv2.imshow('Image', img)
23 cv2.imshow('Image GRAY', imgray)
24 cv2.imshow('Image Binary', thresh)
25 cv2.waitKey(0)
26 cv2.destroyAllWindows()
```

contours[0].shape
Out[2]: (104, 1, 2)

24. Motion Detection and Tracking Using Opencv Contours

(Tracking moving objects with a bounding rectangle)

Optical flow (光流) is the motion of objects between consecutive frames of sequence, caused by the relative movement between the object and camera.

```
dst=cv2.absdiff(src1, src2[, dst])
```

```
area = cv2.contourArea(cnt)
```

```
x, y, w, h = cv.boundingRect(cnt)
```

(x, y) the top-left coordinate of the rectangle
(w, h) its width and height.

24. Motion Detection and Tracking Using Opencv Contours

(Tracking moving objects with a bounding rectangle)

ex30.py

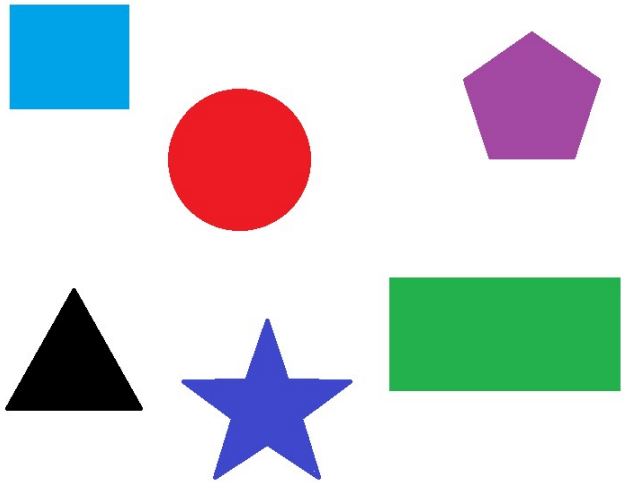
```
import cv2

cap = cv2.VideoCapture('vtest.avi')
ret, frame1 = cap.read()
ret, frame2 = cap.read()
```

ex30.py

```
while cap.isOpened():
    diff = cv2.absdiff(frame1, frame2)
    gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5,5), 0)
    _, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH_BINARY)
    dilated = cv2.dilate(thresh, None, iterations=3)
    contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:
        (x, y, w, h) = cv2.boundingRect(contour)
        if cv2.contourArea(contour) < 900:
            continue
        cv2.rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame1, 'Status:Movement', (10, 20), cv2.FONT_HERSHEY_SIMPLEX,
                    1, (0, 0, 255), 3)

    #cv2.drawContours(frame1, contours, -1, (0, 255, 0), 2)  # green  (Tracking moving objects with contours)
    cv2.imshow('feed1',dilated)
    cv2.imshow('feed2',frame1)
    frame1 = frame2
    ret, frame2 = cap.read()
    if ret == False:
        break
    if cv2.waitKey(40) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Tracking moving objects
with a bounding rectangle

25. Detect Simple Geometric Shapes



shape.png

Contour Approximation: cv2.approxPolyDP

cv2.drawContours

np.array cnt = contours[4]

list

img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3)

numpy.ravel: return a contiguous flattened array (1-D array).

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])        shape: (2, 3)
>>> np.ravel(x)
array([1, 2, 3, 4, 5, 6])          shape: (6, )
>>> x.ravel()
array([1, 2, 3, 4, 5, 6])          shape: (6, )
>>> len(x)
 2
```

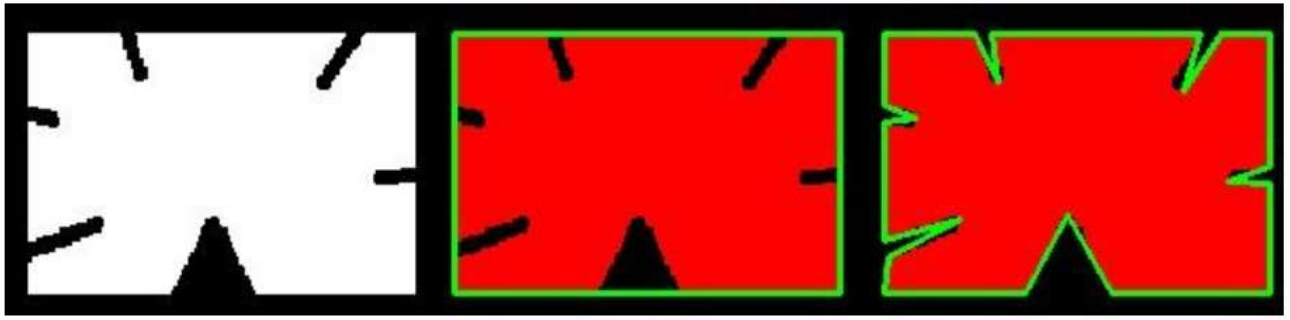
cv2.approxPolyDP

It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify. It is an implementation of Douglas-Peucker algorithm. Check the wikipedia page for algorithm and demonstration.

To understand this, suppose you are trying to find a square in an image, but due to some problems in the image, you didn't get a perfect square, but a "bad shape" (As shown in first image below). Now you can use this function to approximate the shape. In this, second argument is called `epsilon`, which is maximum distance from contour to approximated contour. It is an accuracy parameter. A wise selection of `epsilon` is needed to get the correct output.

```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)
```

Below, in second image, green line shows the approximated curve for `epsilon = 10% of arc length`. Third image shows the same for `epsilon = 1% of the arc length`. Third argument specifies whether curve is closed or not.



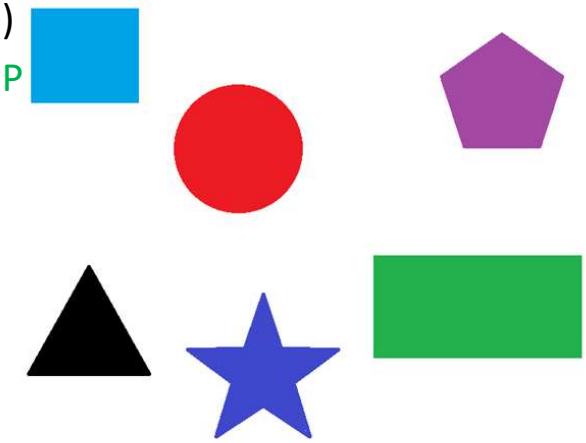
ex31.py

```
 8  import numpy as np
 9  import cv2
10
11  img = cv2.imread('shape.png')
12  imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13  _, thresh = cv2.threshold(imgGray, 240, 255, cv2.THRESH_BINARY_INV)
14  contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE) # 6 contours
15
16  cv2.imshow("img", img)
```

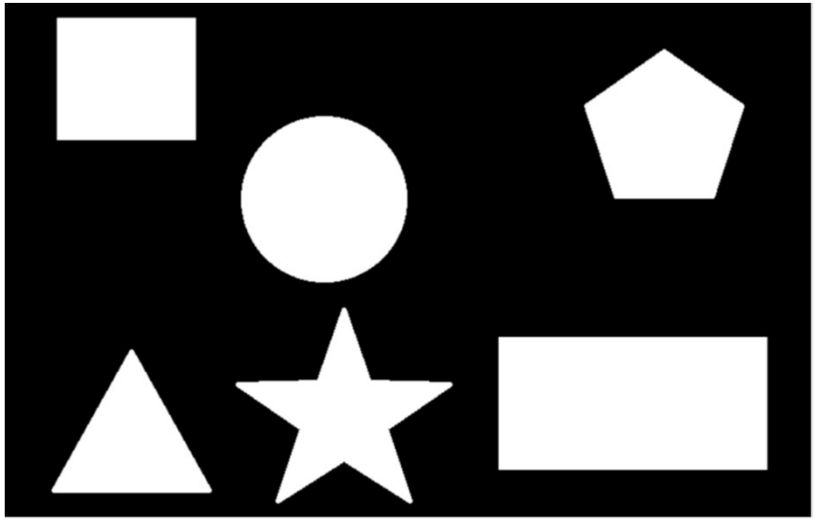
(496,1,2)

cv2.approxPolyDP

(4,1,2)

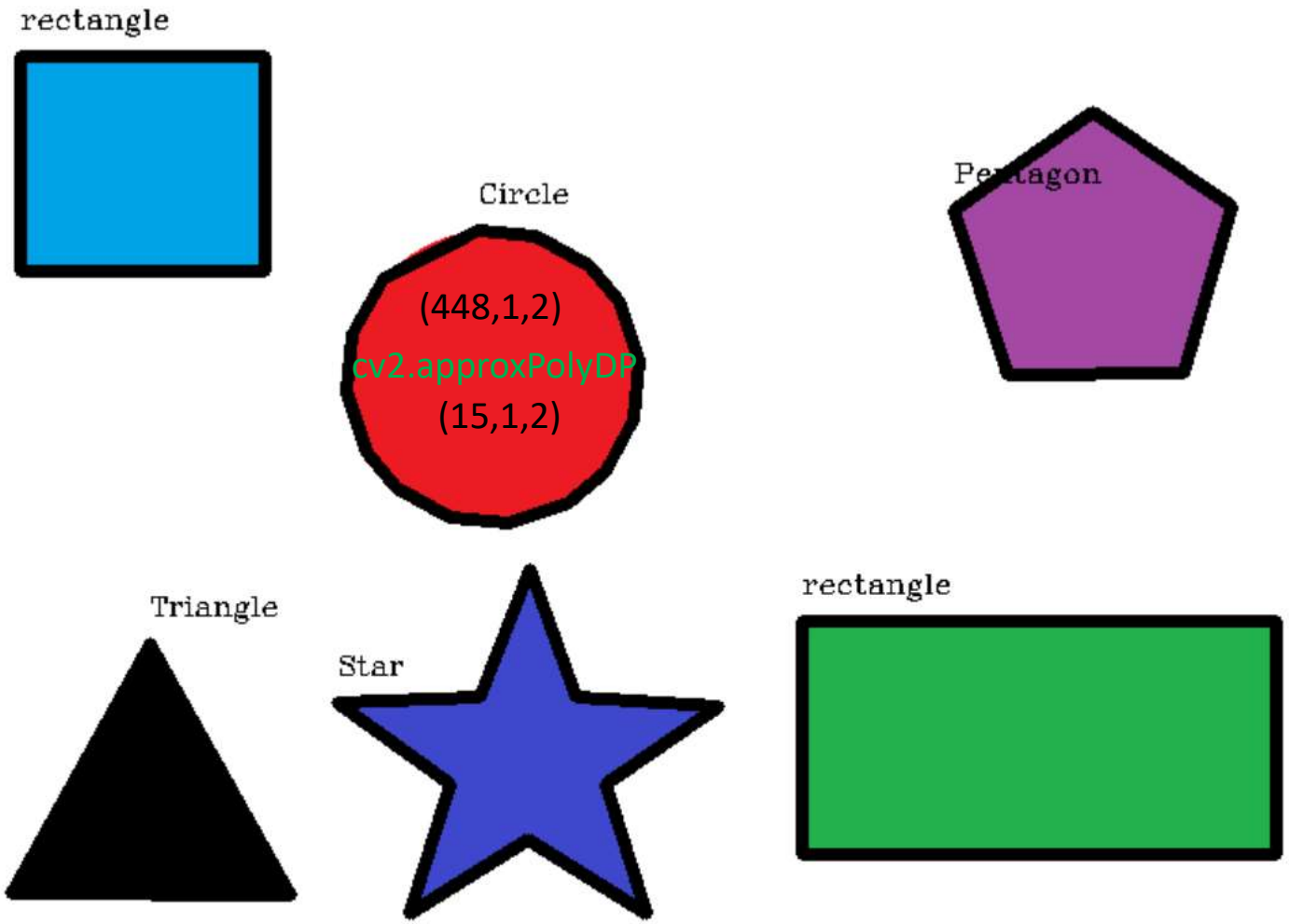


img



thresh

```python
for contour in contours:
    approx = cv2.approxPolyDP(contour, 0.01* cv2.arcLength(contour, True), True)
    cv2.drawContours(img, [approx], 0, (0, 0, 0), 5) # black
    x = approx.ravel()[0]
    y = approx.ravel()[1] - 15
    if len(approx) == 3:
        cv2.putText(img, "Triangle", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
    elif len(approx) == 4:
        x1 ,y1, w, h = cv2.boundingRect(approx)
        aspectRatio = float(w)/h
        print(aspectRatio)
        if aspectRatio >= 0.95 and aspectRatio <= 1.05:
          cv2.putText(img, "square", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
        else:
          cv2.putText(img, "rectangle", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
    elif len(approx) == 5:
        cv2.putText(img, "Pentagon", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
    elif len(approx) == 10:
        cv2.putText(img, "Star", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
    else:
        cv2.putText(img, "Circle", (x, y), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))

cv2.imshow("shape", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


rectangle
Circle
(448,1,2)
cv2.approxPolyDP
(15,1,2)
Pentagon
Triangle
Star
rectangle

26. Understanding image Histograms (直方圖)

Histogram: as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

By looking at the histogram of an image, you get intuition about contrast, brightness, intensity distribution etc of that image.

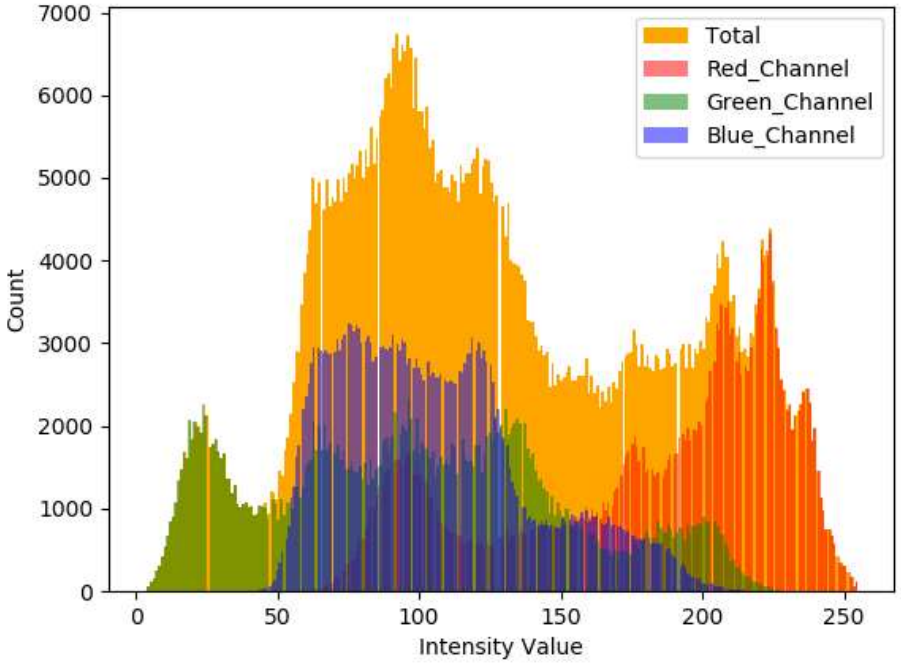
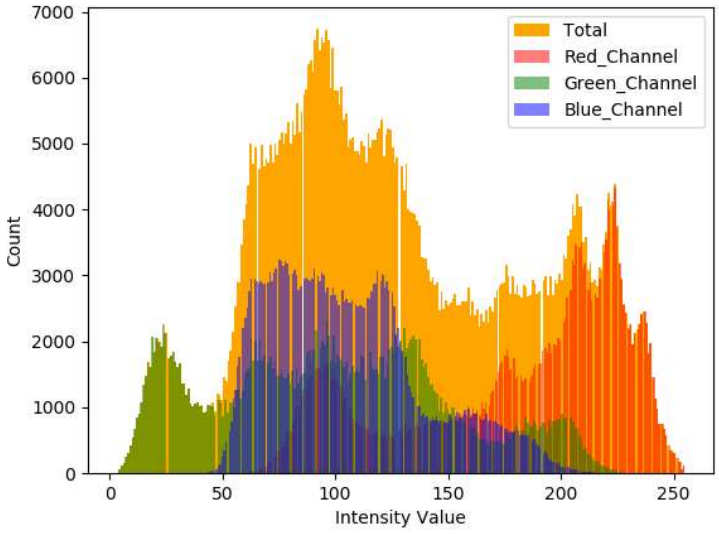


Image Histograms (直方圖)



ex32.py

Method 1: Matplotlib function

```
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread("lena.jpg")
b, g, r = cv.split(img)
cv.imshow("img", img)
cv.imshow("b", b)
cv.imshow("g", g)
cv.imshow("r", r)

plt.hist(b.ravel(), 256, [0, 256], color='b')
plt.hist(g.ravel(), 256, [0, 256], color='g')
plt.hist(r.ravel(), 256, [0, 256], color='r')
# the upper boundary 256 is exclusive
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```

Compute and draw the histogram of x

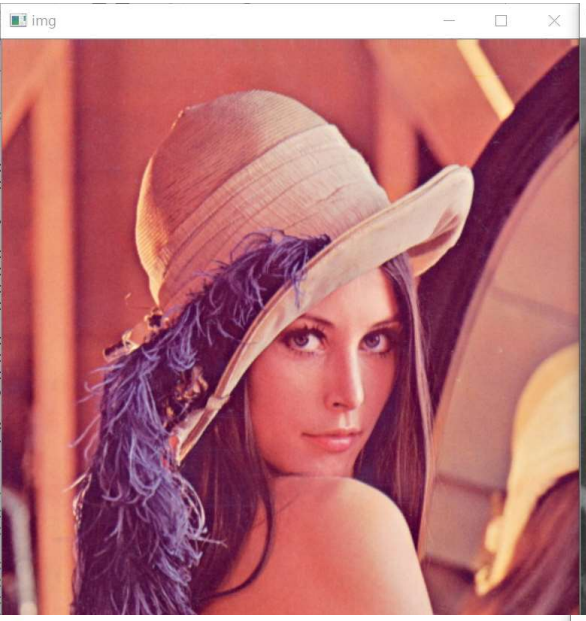
plt.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

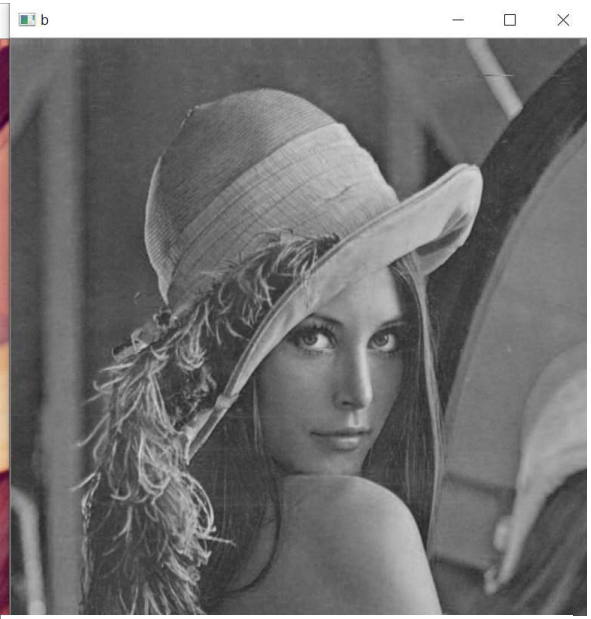
x: (n,) array or sequence of (n,) arrays

example:

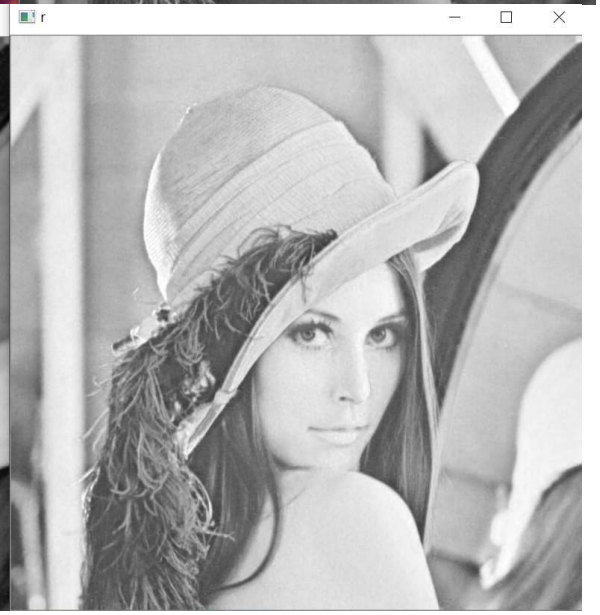
```
plt.hist(b.ravel(), 256, [0, 256], color='b')
plt.hist(g.ravel(), 256, [0, 256], color='g')
plt.hist(r.ravel(), 256, [0, 256], color='r')
```

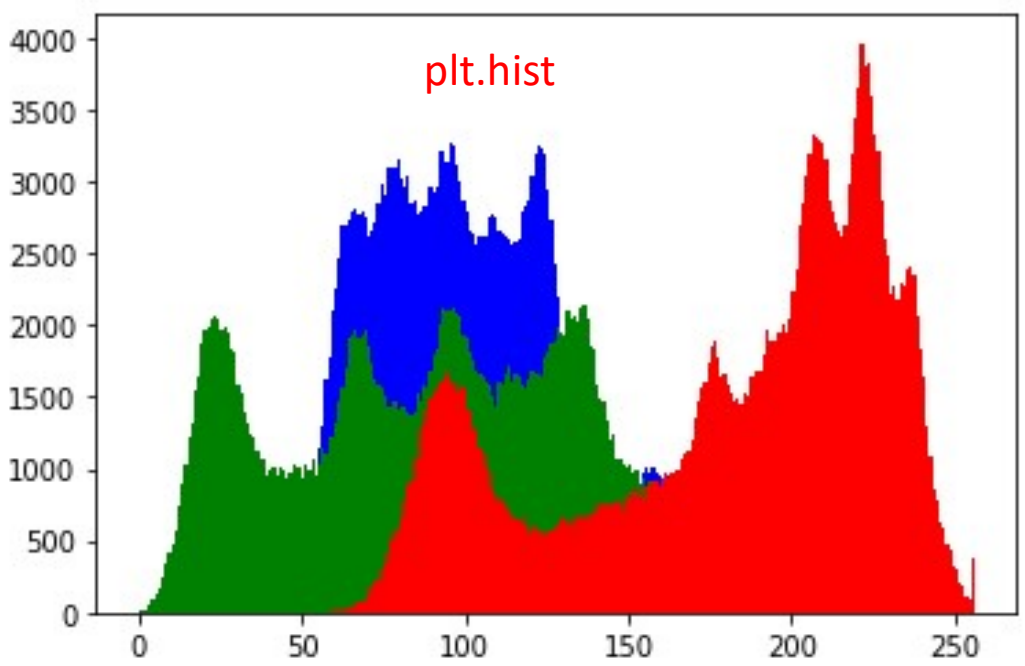
```
b.shape
Out[2]: (512, 512)

b.ravel().shape
Out[4]: (262144,)

array([128, 127, 126, ...,  81,  81,  84],
dtype=uint8)
```


img


b


g


r


plt.hist
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

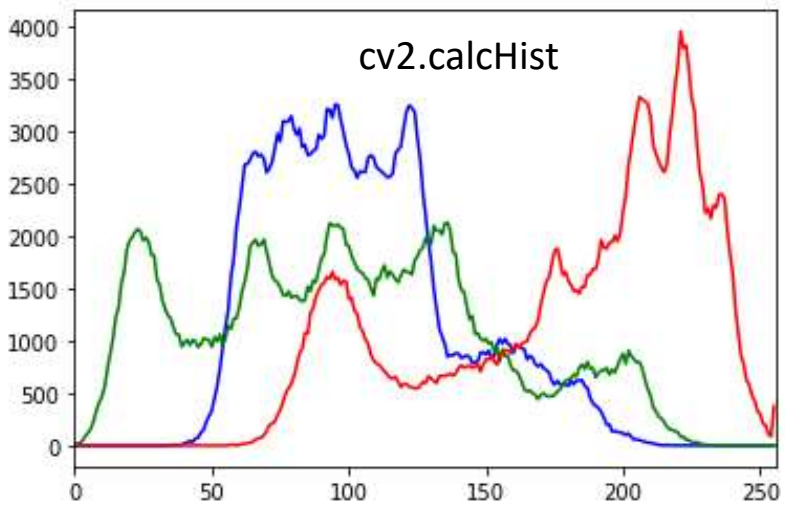

cv2.calcHist
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

Image Histograms (直方圖)

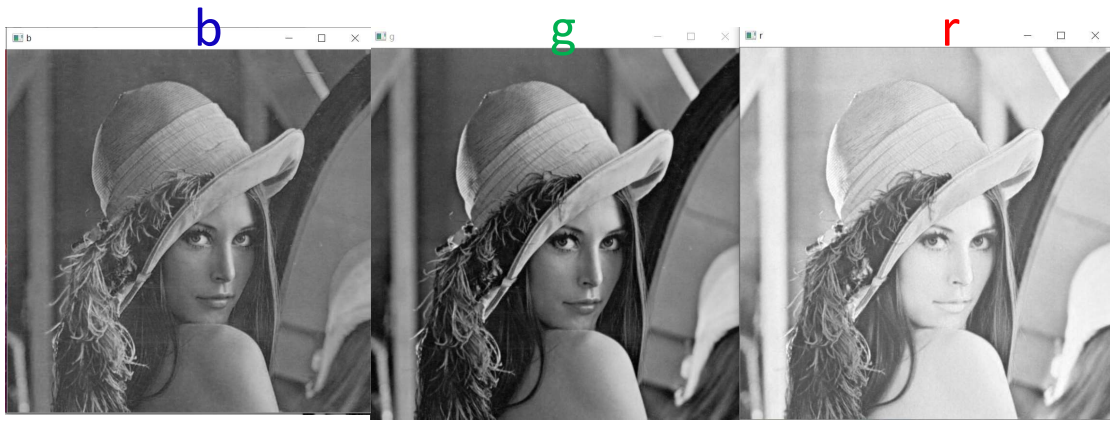
ex33.py

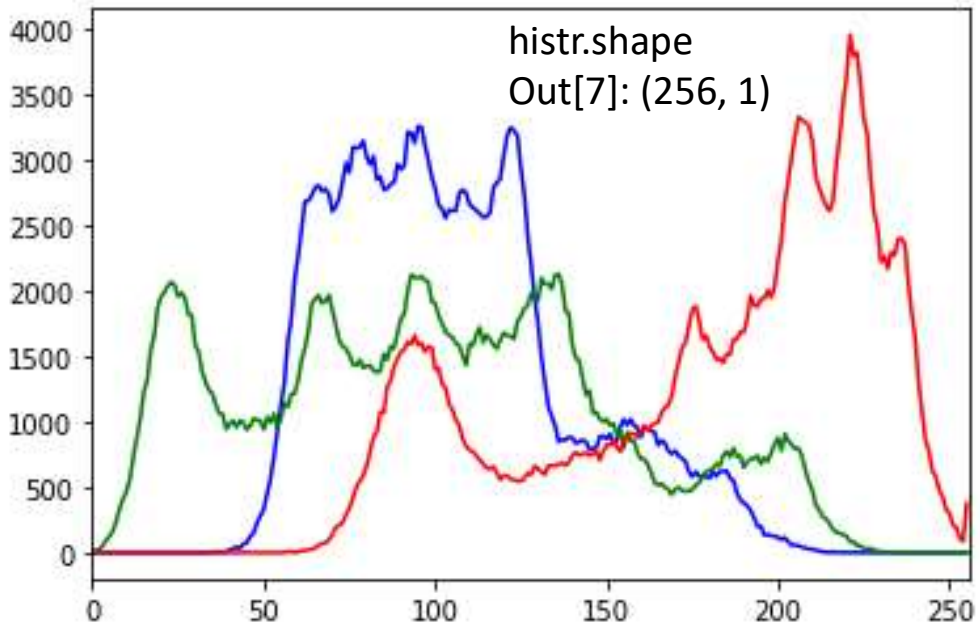
```
 8 import cv2
 9 from matplotlib import pyplot as plt
10
11 img = cv2.imread('lena.jpg')
12 color = ('b','g','r')
13 for i,col in enumerate(color):
14     histr = cv2.calcHist([img],[i],None,[256],[0,256]) # the upper boundary 256 is exclusive
15     plt.plot(histr,color = col)
16     plt.xlim([0,256])
17 plt.show()
```

Method 2: OpenCV function

cv2.calcHist([src], [0], None, [histSize], [histRange])

channel　mask





27. Template Matching

模板匹配

To find objects in an image using Template Matching

40(w) x52(h)

https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html

res = cv2.matchTemplate(src, template, method)

(grayscale image)

6 methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR', 'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']

Take minimum value for TM_SQDIFF and TM_SQDIFF_NORMED

548 (w)x 342(h)

ex34.py

```
import cv2
import numpy as np

img = cv2.imread("messi5.jpg")  # shape (342, 548, 3)
grey_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # shape (342, 548)
template = cv2.imread("messi_face.jpg", 0)  # shape (52, 40)
w, h = template.shape[::-1] # w=40,  h=52
#h, w = template.shape[::]
#(h,w)=template.shape
res = cv2.matchTemplate(grey_img, template, cv2.TM_CCORR_NORMED )
print(res)     # shape (291, 509)
threshold = 0.99
loc = np.where(res >= threshold)
print(loc)  #(array([85], dtype=int64), array([220], dtype=int64))   top_left corner of rectangle
for pt in zip(*loc[::-1]):
    cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)

cv2.imshow("img", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

threshold = 0.99

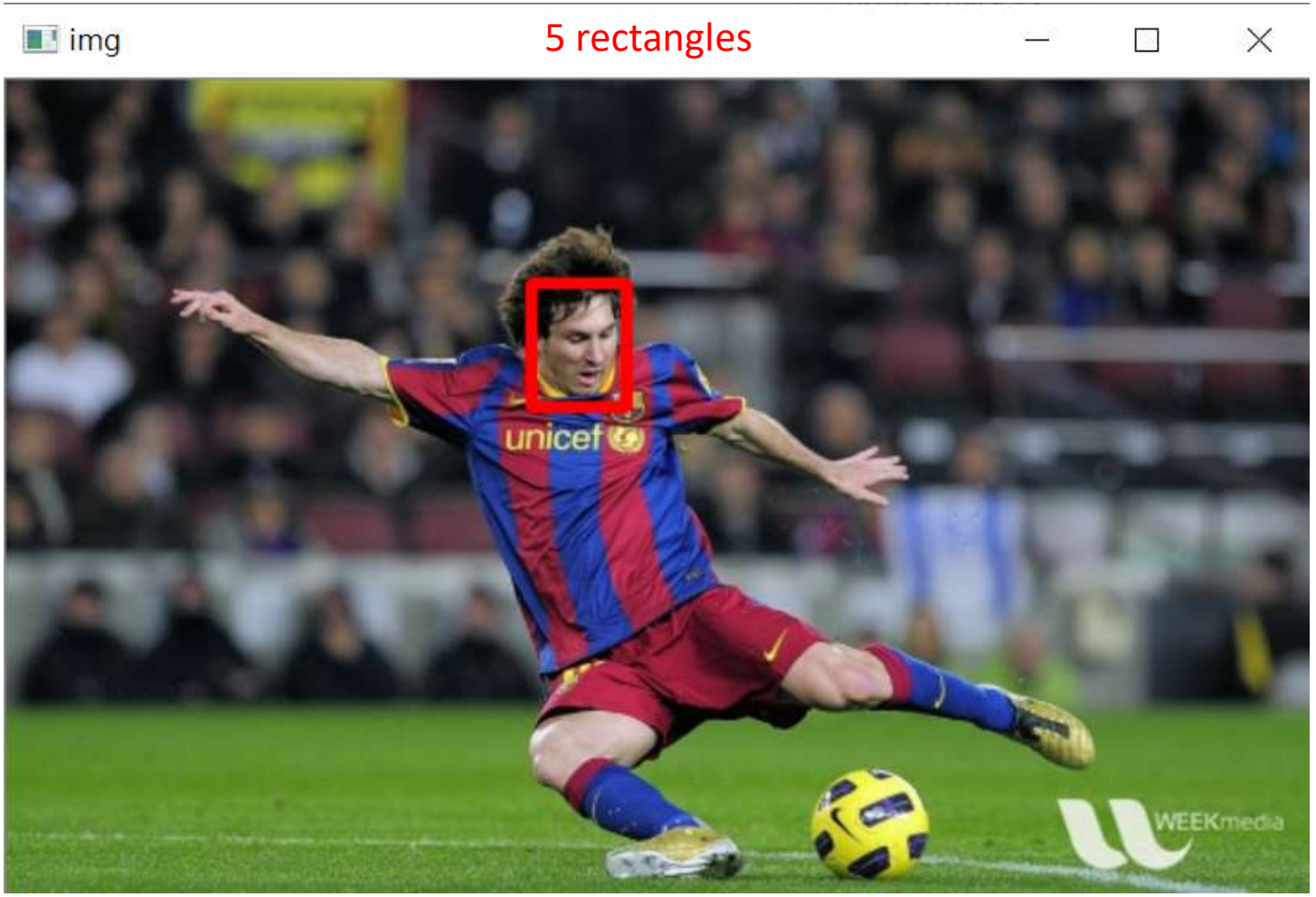
1 rectangle

(x, y)=(220, 85) top_left corner of rectangle

threshold = 0.95

5 loc : (array([84, 85, 85, 85, 86], dtype=int64),
array([220, 219, 220, 221, 220], dtype=int64))

top_left corner of rectangle

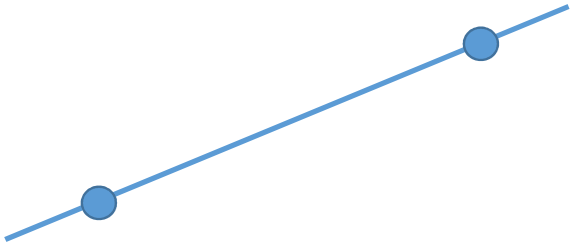


28. Hough Line Transform Theory

To understand the concept of the Hough Transform and Hough Line Transform Theory.

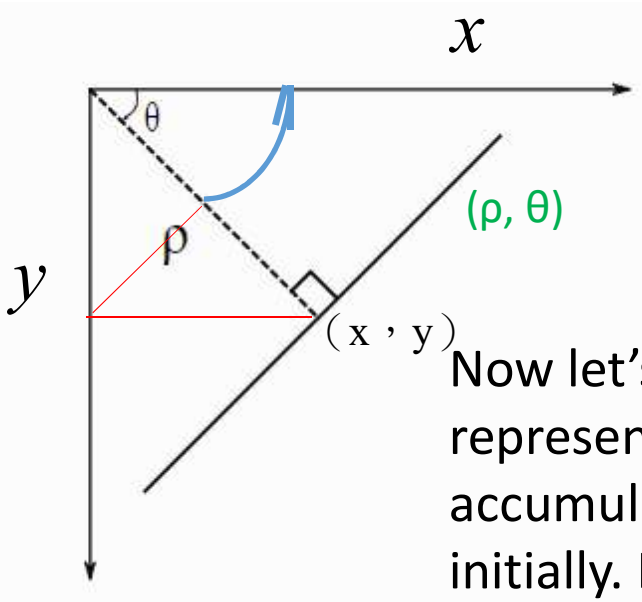
OpenCV implements two kind of Hough Line Transforms:
The Standard Hough Transform (HoughLines method)
The Probabilistic Hough Line Transform (HoughLinesP method)

Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit. We will see how it works for a line.

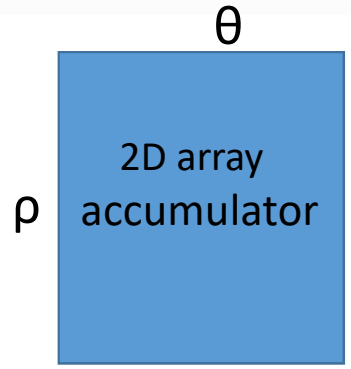


A line in Cartesian coordinate:

$y = m\,x + c$ $\quad (m,\ c)$

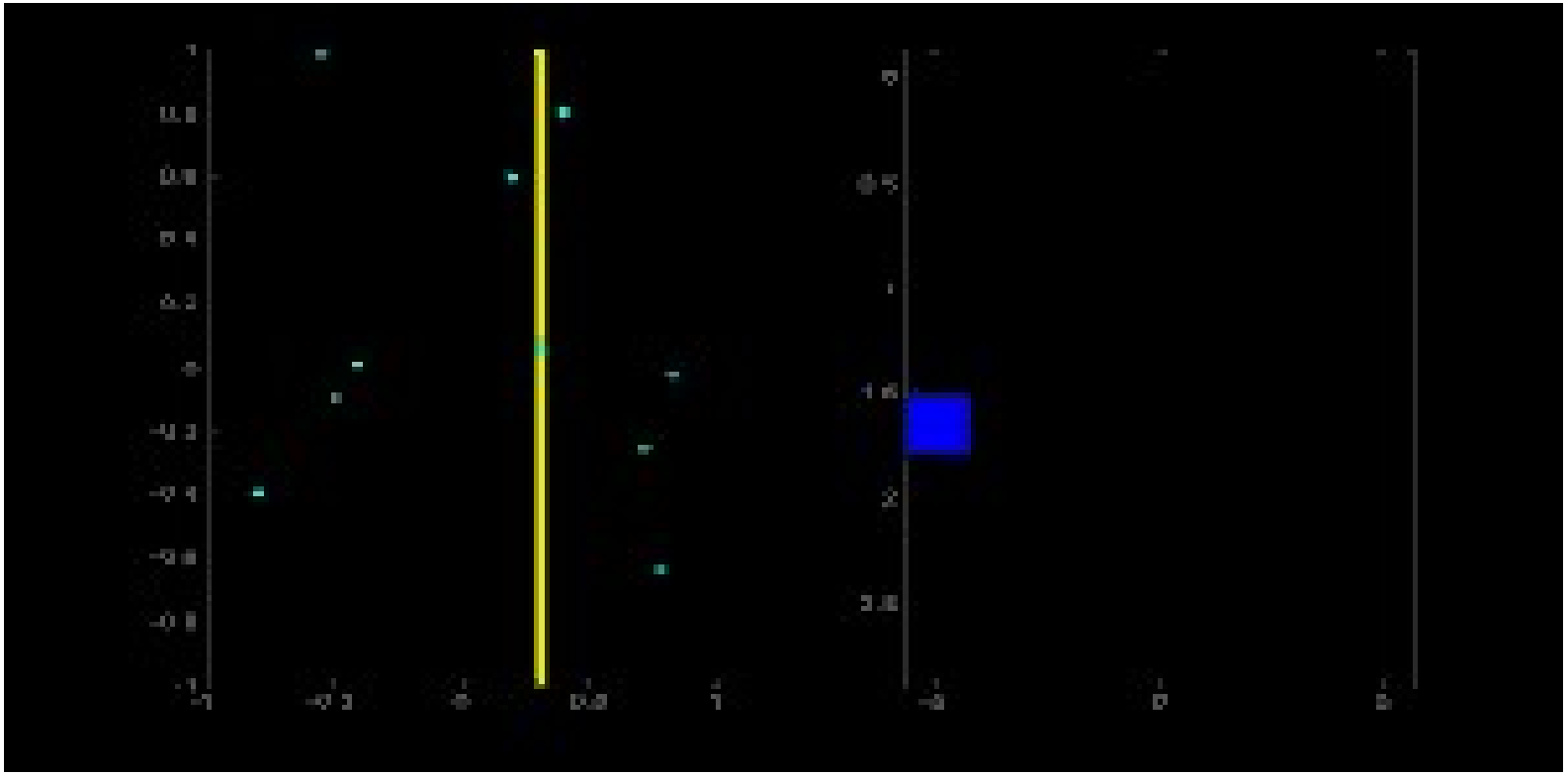


or in parametric form, as $\rho = x\cos\theta + y\sin\theta$ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise.

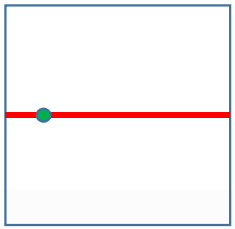


Now let's see how Hough Transform works for lines. Any line can be represented in these two terms, (ρ, θ). So first it creates a 2D array or accumulator (to hold values of two parameters) and it is set to 0 initially. Let rows denote the ρ and columns denote the θ. Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns. For ρ, the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.

So a line in Cartesian coordinate has a corresponding point (ρ, θ) in the 2D array or accumulator.



(Image Courtesy: Amos Storkey)



(ρ, θ) = (50, 90)

$\rho = x \cos\theta + y \sin\theta$

Consider a 100x100 image with a horizontal line at the middle. Take the first point of the line. You know its (x,y) values. Now in the line equation, put the values $\theta = 0, 1, 2, ..., 180$ and check the ρ you get. For every (ρ, θ) pair, you increment value by one in our accumulator in its corresponding (ρ, θ) cells. So now in accumulator, the cell (50,90) = 1 along with some other cells.

Now take the second point on the line. Do the same as above. Increment the the values in the cells corresponding to (ρ, θ) you got. This time, the cell (50,90) = 2. What you actually do is voting the (ρ, θ) values. You continue this process for every point on the line. At each point, the cell (50,90) will be incremented or voted up, while other cells may or may not be voted up. This way, at the end, the cell (50,90) will have maximum votes. So if you search the accumulator for maximum votes, you get the value (50,90) which says, there is a line in this image at distance 50 from origin and at angle 90 degrees.

Hough transformation Algorithm

- 1. Edge detection, e.g. using the Canny edge detector.
- 2. Mapping of edge points to the Hough space and storage in an accumulator.
- 3. Interpretation of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and possibly other constraints.
- 4. Conversion of infinite lines to finite lines.

29. Hough Line Transform using HoughLines method

cv2.Canny(grayimg, minVal, maxVal, aperture_size):
First argument is our input image.
Second and third arguments are our minVal and maxVal respectively. Fourth argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3.

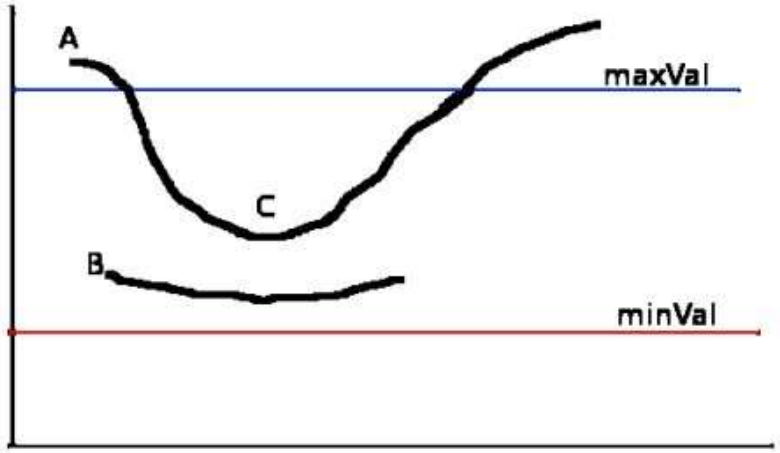
-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Sobel kernel 3x3



29. Hough Line Transform using HoughLines method

lines = cv.HoughLines(image, rho, theta, threshold)

image : source image. (binary image)

lines : Output vector of lines. Each line is represented by a 2 or 3 element vector (ρ,θ) or (ρ,θ,votes) . ρ is the distance from the coordinate origin (0,0) (top-left corner of the image). θ is the line rotation angle in radians . votes is the value of accumulator.

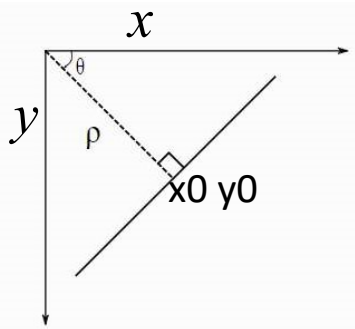
rho : Distance resolution of the accumulator in pixels.
theta : Angle resolution of the accumulator in radians.

threshold : Accumulator threshold parameter. Only those lines are returned that get enough votes (>threshold).

ex35.py

(x1, y1) and (x2, y2) satisfy the line equation

$\rho = x\cos\theta + y\sin\theta$



lines：

array([[[*, #]],

[[*, #]],

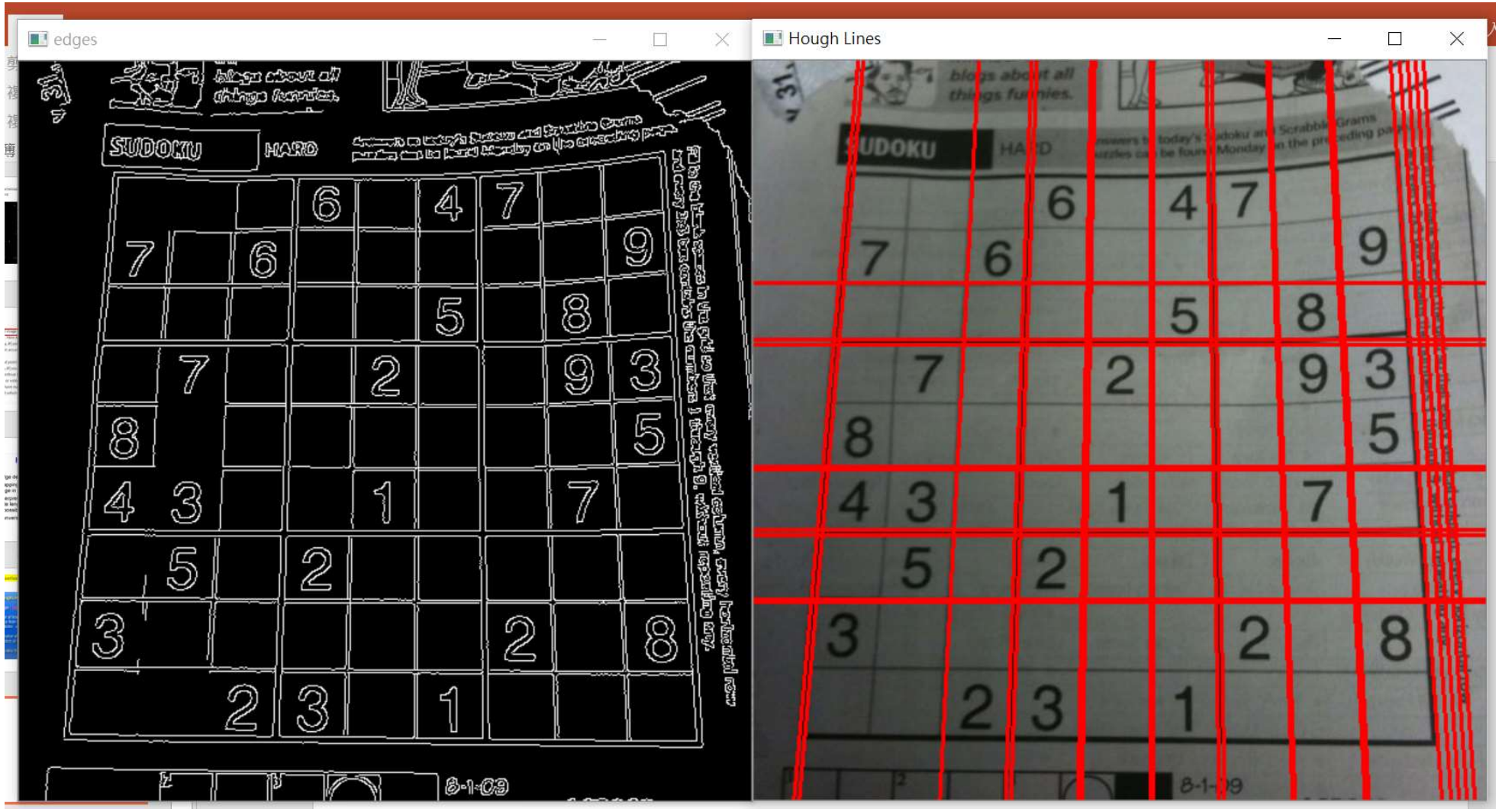
. . .

[[* , #]]],

dtype=float32)

```python
import cv2
import numpy as np

img = cv2.imread('sudoku.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
cv2.imshow('edges', edges)
lines = cv2.HoughLines(edges, 1, np.pi / 180, 200) # (29, 1, 2)
# (no of lines, rho, theta)
for line in lines:
    rho,theta = line[0]        # [rho,theta] = line[0]
    c = np.cos(theta)
    s = np.sin(theta)
    x0 = c * rho
    y0 = s * rho
    # x1 stores the rounded off value of (r * cos(theta) - 1000 * sin(theta))
    x1 = int(x0 + 1000 * (-s))
    # y1 stores the rounded off value of (r * sin(theta)+ 1000 * cos(theta))
    y1 = int(y0 + 1000 * (c))
    # x2 stores the rounded off value of (r * cos(theta)+ 1000 * sin(theta))
    x2 = int(x0 - 1000 * (-s))
    # y2 stores the rounded off value of (r * sin(theta)- 1000 * cos(theta))
    y2 = int(y0 - 1000 * (c))
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2.imshow('Hough Lines', img)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

29 lines



30. Probabilistic Hough Transform using HoughLinesP

```
lines=cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])
```

rho : Distance resolution of the accumulator in pixels.

theta: Angle resolution of the accumulator in radians.

threshold:Accumulator threshold parameter. Only those lines are returned that get enough votes (> threshold).

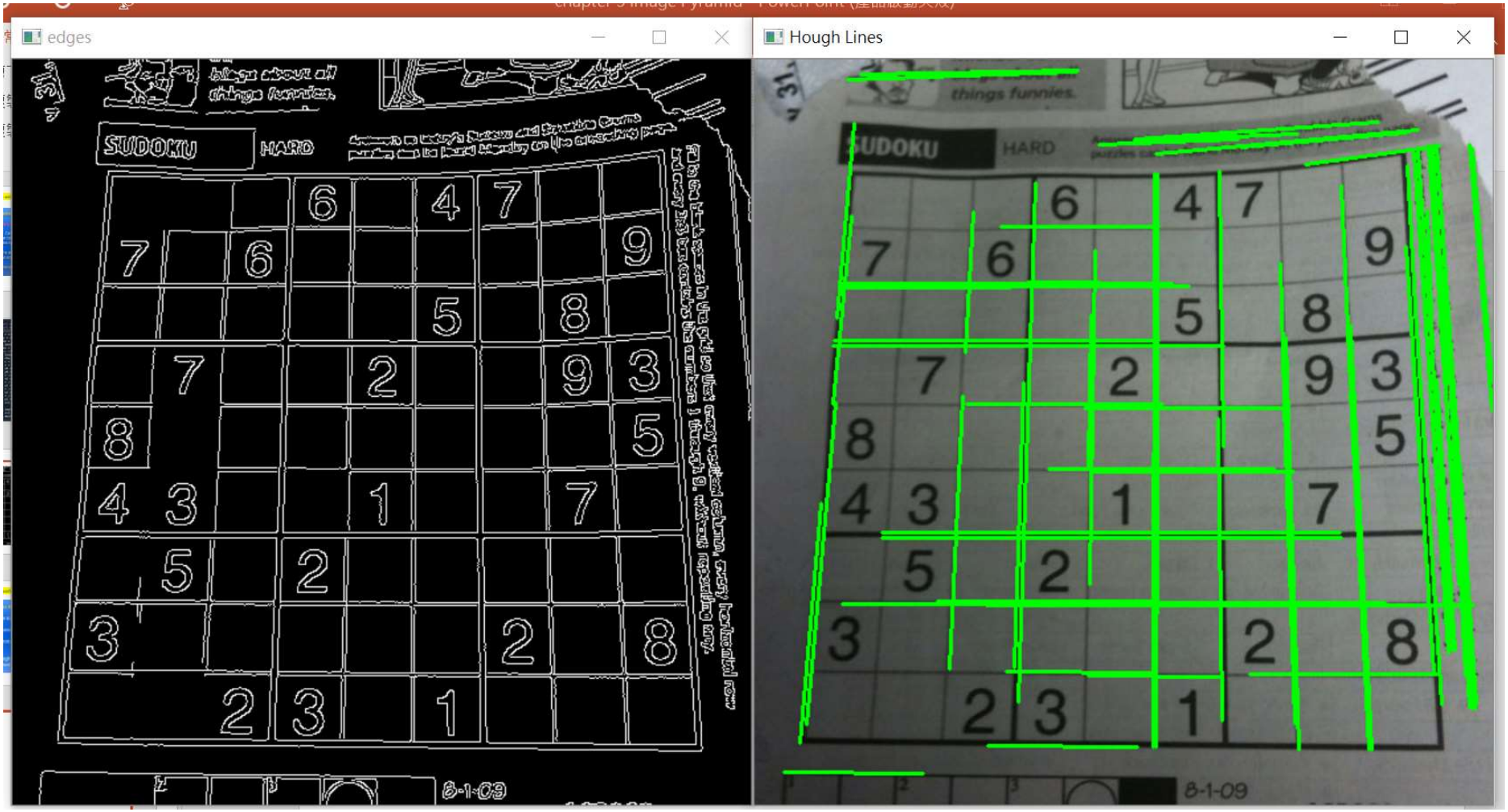
minLineLength : Minimum length of line. Line segments shorter than this are rejected.
maxLineGap : Maximum allowed gap between line segments to treat them as a single line.

ex36.py

72 lines

```
import cv2
import numpy as np
img = cv2.imread('sudoku.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
cv2.imshow('edges', edges)
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength=100,maxLineGap=10)
# (72, 1, 4)
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2) # green

cv2.imshow('Hough Lines', img)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

72 lines



31. Road Lane Line Detection (I)

ex37.py

Img = cv.fillPoly(img, pts, color[, lineType[, shift[, offset]]])

Fills the area bounded by one or more polygons.

cv2.fillPoly(mask, vertices, match_mask_color)

Parameters

img	Image.
pts	Array of polygons where each polygon is represented as an array of points.
color	Polygon color.
lineType	Type of the polygon boundaries. See LineTypes
shift	Number of fractional bits in the vertex coordinates.
offset	Optional offset of all points of the contours.

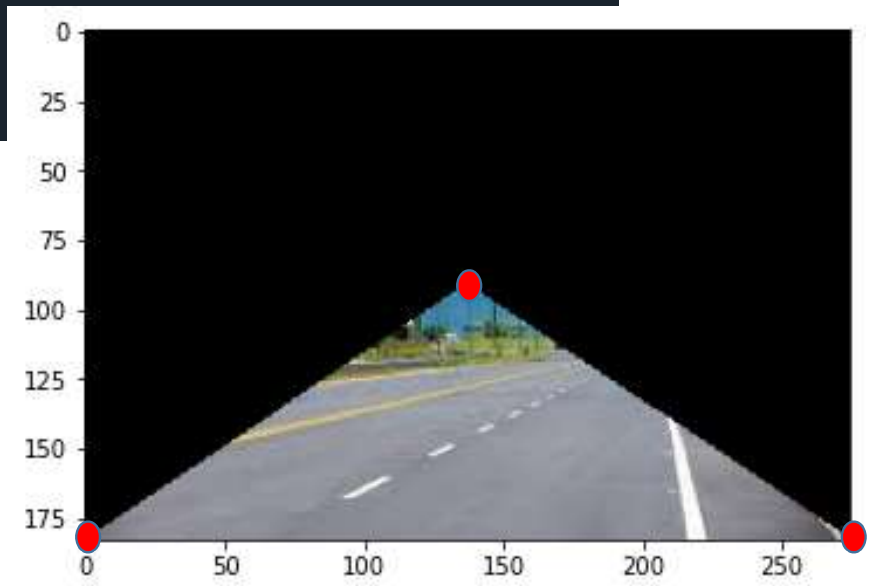
ex37.py

```python
import matplotlib.pyplot as plt
import cv2
import numpy as np

image = cv2.imread('road.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.show()

print(image.shape)  #(183, 275, 3)
height = image.shape[0]
width = image.shape[1]

region_of_interest_vertices = [
    (0, height),
    (width/2, height/2),
    (width, height)
]
```

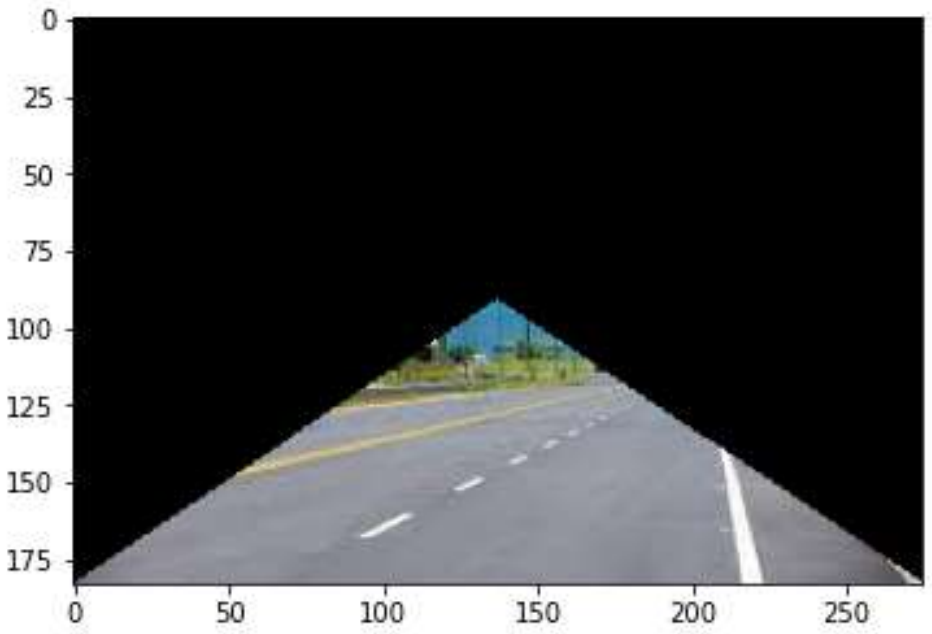
275 x 183

ex37.py

```
27      def region_of_interest(img, vertices):
28          mask = np.zeros_like(img)  # (183, 275, 3) black
29          channel_count = img.shape[2]   # 3
30          match_mask_color = (255,) * channel_count # (255, 255, 255) white
31          cv2.fillPoly(mask, vertices, match_mask_color)
32          masked_image = cv2.bitwise_and(img, mask)
33          return masked_image
34
35      cropped_image = region_of_interest(image,
36                      np.array([region_of_interest_vertices], np.int32))
37                    # (1, 3, 2)
38      plt.imshow(cropped_image)
39      plt.show()
```




0
25
50
75
100
125
150
175
0
50
100
150
200
250

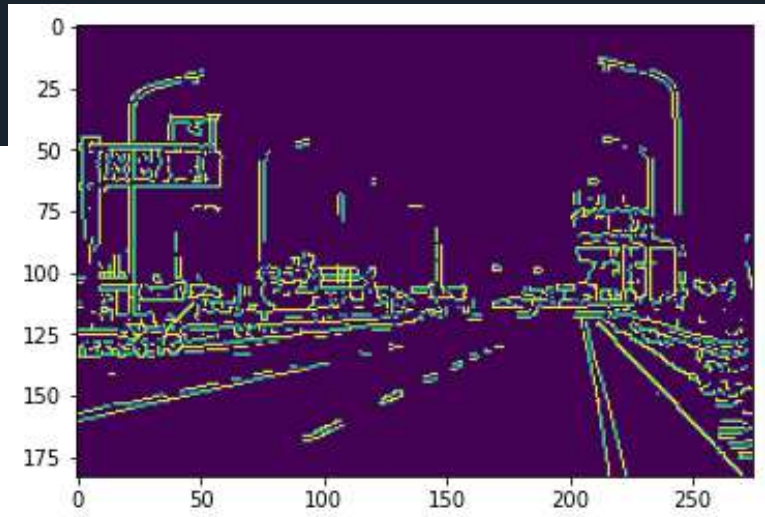

0
25
50
75
100
125
150
175
0
50
100
150
200
250

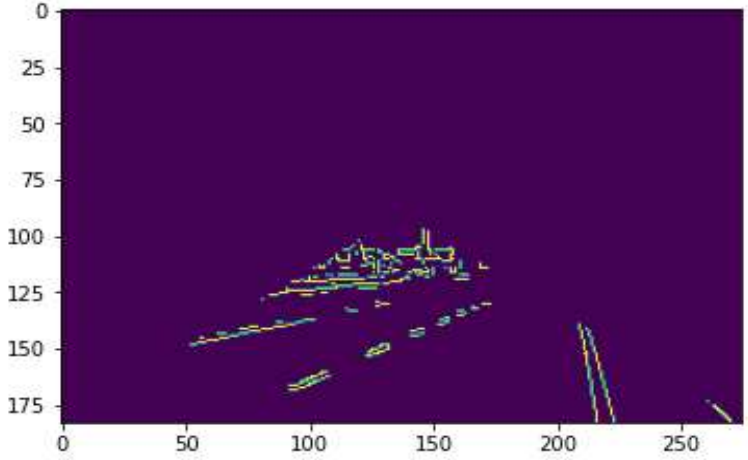
32. Road Lane Line Detection (II)

ex38.py

```
import matplotlib.pyplot as plt
import cv2
import numpy as np

def region_of_interest(img, vertices):
    mask = np.zeros_like(img)
    #channel_count = img.shape[2]
    match_mask_color = 255
    cv2.fillPoly(mask, vertices, match_mask_color)
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image

def drow_the_lines(img, lines):
    blank_image = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)

    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(blank_image, (x1,y1), (x2,y2), (0, 255, 0), thickness=1)

    img = cv2.addWeighted(img, 0.8, blank_image, 1, 0.0)
    return img
```

```python
image = cv2.imread('road.jpg')  # b,g,r
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # r,g,b
print(image.shape) # (183, 275, 3)
height = image.shape[0]
width = image.shape[1]
region_of_interest_vertices = [
    (0, height),
    (width/2, height/2),
    (width, height)]
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
canny_image = cv2.Canny(gray_image, 160, 200)
cropped_image = region_of_interest(canny_image,
                np.array([region_of_interest_vertices], np.int32))
                # shape = (1,3,2)
plt.imshow(canny_image)
plt.show()
plt.imshow(cropped_image)
plt.show()
```





```
48      lines = cv2.HoughLinesP(cropped_image,
49                              rho=1,
50                              theta=np.pi/180,
51                              threshold=35,
52                              minLineLength=35,
53                              maxLineGap=25)
54      print(lines.shape) # (5, 1, 4)
55      image_with_lines = drow_the_lines(image, lines)
56      plt.imshow(image_with_lines)
57      plt.show()
```

lines.shape: (5, 1, 4)

33. Road Lane Line Detection (III)

test.mp4: You can use any clip which contains the lane. Try this
https://www.youtube.com/watch?v=6q5_A5wOwDM

VideoCapture → frame (color) → grayscale → Canny edges → cropped image (ROI) → Lines (ROI) → draw lines in frame

ex39.py

```
 8    import cv2
 9    import numpy as np
10    import time
11
12    def region_of_interest(img, vertices):
13        mask = np.zeros_like(img)
14        #channel_count = img.shape[2]
15        match_mask_color = 255
16        cv2.fillPoly(mask, vertices, match_mask_color)
17        masked_image = cv2.bitwise_and(img, mask)
18        return masked_image
```

```
20  def drow_the_lines(img, lines):
21      img = np.copy(img)
22      blank_image = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
23
24      for line in lines:
25          for x1, y1, x2, y2 in line:
26              cv2.line(blank_image, (x1,y1), (x2,y2), (0, 255, 0), thickness=4)
27
28      img = cv2.addWeighted(img, 0.8, blank_image, 1, 0.0)
29      return img
```

```
31      def process(image):
32          height = image.shape[0]
33          width = image.shape[1]
34          region_of_interest_vertices = [
35              (0, height),
36              (width/2, height*3/4),
37              (width, height)
38          ]
39          gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
40          canny_image = cv2.Canny(gray_image, 100, 200)
41          cropped_image = region_of_interest(canny_image,
42                          np.array([region_of_interest_vertices], np.int32),)
43          lines = cv2.HoughLinesP(cropped_image,
44                                  rho=2,
45                                  theta=np.pi/180,
46                                  threshold=50,
47                                  lines=np.array([]),
48                                  minLineLength=40,
49                                  maxLineGap=100)
50          image_with_lines = drow_the_lines(image, lines)
51          return image_with_lines
```

```
53      cap = cv2.VideoCapture('test.mp4')
54
55      while cap.isOpened():
56          ret, frame = cap.read()
57          frame = process(frame)
58          cv2.imshow('frame', frame)
59          time.sleep(0.034)
60          if cv2.waitKey(1) & 0xFF == ord('q'):
61              break
62
63      cap.release()
64      cv2.destroyAllWindows()
```