**OpenCV实验手册**

**OpenCV实验手册**

一、OpenCV安装

1. OpenCV介绍
2. 安装

二、OpenCV基本操作

1. 图像读取与保存

1）读取、图像、保存图像

1. 图像色彩操作
   1. 彩色图像转换为灰度图像
   2. 色彩通道操作
   3. 灰度直方图均衡化
   4. 彩色亮度直方图均衡化
   5. 色彩提取
   6. 二值化与反二值化
2. 图像形态操作
3. 图像翻转
4. 图像位置变换
5. 图像缩放
6. 图像裁剪
7. 图像相加
8. 图像相减
9. 透视变换
10. 图像腐蚀
11. 图像膨胀
12. 图像开运算
13. 图像闭运算
14. 形态学梯度
15. 图像梯度处理
16. 模糊处理
17. 图像锐化处理
18. 边沿检测
19. 轮廓处理
20. 查找并绘制轮廓
21. 绘制矩形包围框
22. 绘制圆形包围框
23. 绘制最佳拟合椭圆
24. 逼近多边形三、综合案例
25. 利用OpenCV实现图像校正
    1. 任务描述
    2. 代码
    3. 执行结果
26. 利用OpenCV检测芯片瑕疵
27. 任务描述
28. 代码
29. 执行结果

# 一、OpenCV安装

## OpenCV介绍

1. **安装**
2. # 安装opencv核心库
3. pip3 install --user opencv-python==3.4.2.16 --index-url https://pypi.tuna.tsinghua.edu.cn/simple/ --trusted-host https://pypi.tuna.tsinghua.edu.cn

3

1. # 安装opencv贡献库
2. pip3 install --user opencv-contrib-python==3.4.2.16 --index-url https://pypi.tuna.tsinghua.edu.cn/simple/ --trusted-host https://pypi.tuna.tsinghua.edu.cn

**二、OpenCV基本操作**

1. **图像读取与保存**

### 1）读取、图像、保存图像

im = cv2.imread("../data/Linus.png", 1) # 1表示3通道彩色，0表示单通道灰度

cv2.imshow("test", im) # 在test窗口中显示图像

print(type(im)) # 打印数据类型

print(im.shape) # 打印图像尺寸

cv2.imwrite("../data/Linus\_2.png", im) # 将图像保存到指定路径

cv2.waitKey() # 等待用户按键反馈

cv2.destroyAllWindows() # 销毁所有创建的窗口

# 读取图像

import cv2

1

2

3

4

5

6

7

8

9

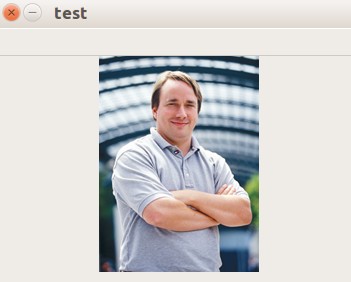
10

11

12

13

执行结果：



1. **图像色彩操作**
2. **彩色图像转换为灰度图像**

im = cv2.imread("../data/Linus.png", 1) cv2.imshow("RGB", im) # 在test窗口中显示图像

# 使用cvtColor进行颜色空间变化，COLOR\_BGR2GRAY表示BGR to GRAY img\_gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY) # 彩色图像灰度化cv2.imshow("Gray", img\_gray)

cv2.waitKey() # 等待用户按键反馈

cv2.destroyAllWindows() # 销毁所有创建的窗口

# 彩色图像转换为灰度图像示例

import cv2

1

2

3

4

5

6

7

8

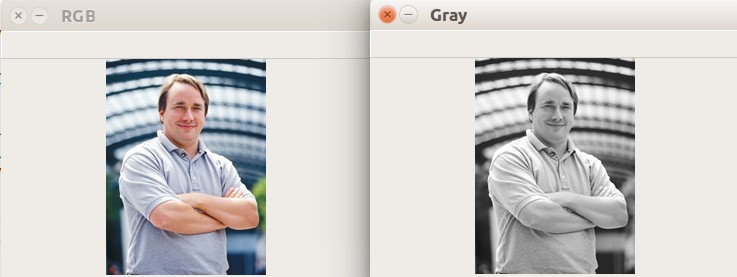
9

10

11

12

执行结果：



1. **色彩通道操作**

1 # 色彩通道操作：通道表示为BGR

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

import numpy as np

import cv2

im = cv2.imread("../data/opencv2.png") print(im.shape)

cv2.imshow("im", im)

# 取出蓝色通道，当做单通道图像显示

b = im[:, :, 0]

cv2.imshow("b", b)

# 去掉蓝色通道(索引为0的通道) im[:, :, 0] = 0

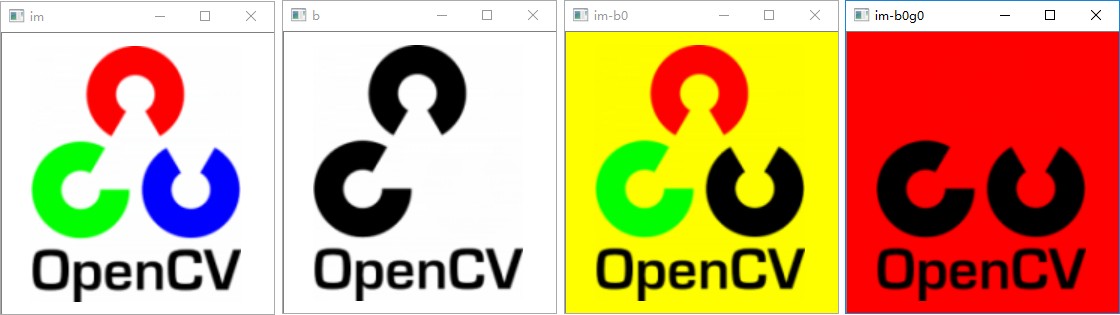
cv2.imshow("im-b0", im)

# 去掉绿色通道(索引为1的通道) im[:, :, 1] = 0

cv2.imshow("im-b0g0", im)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **灰度直方图均衡化**

1

2

3

4

5

6

7

8

9

10

11

12

# 直方图均衡化示例

import numpy as np import cv2

from matplotlib import pyplot as plt

im = cv2.imread("../data/sunrise.jpg", 0) cv2.imshow("orig", im)

# 直方图均衡化

im\_equ = cv2.equalizeHist(im) cv2.imshow("equ1", im\_equ)

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

# 绘制灰度直方图

## 原始直方图print(im.ravel()) plt.subplot(2, 1, 1)

plt.hist(im.ravel(), #ravel返回一个连续的扁平数组256, [0, 256], label="orig")

plt.legend()

## 均衡化处理后的直方图

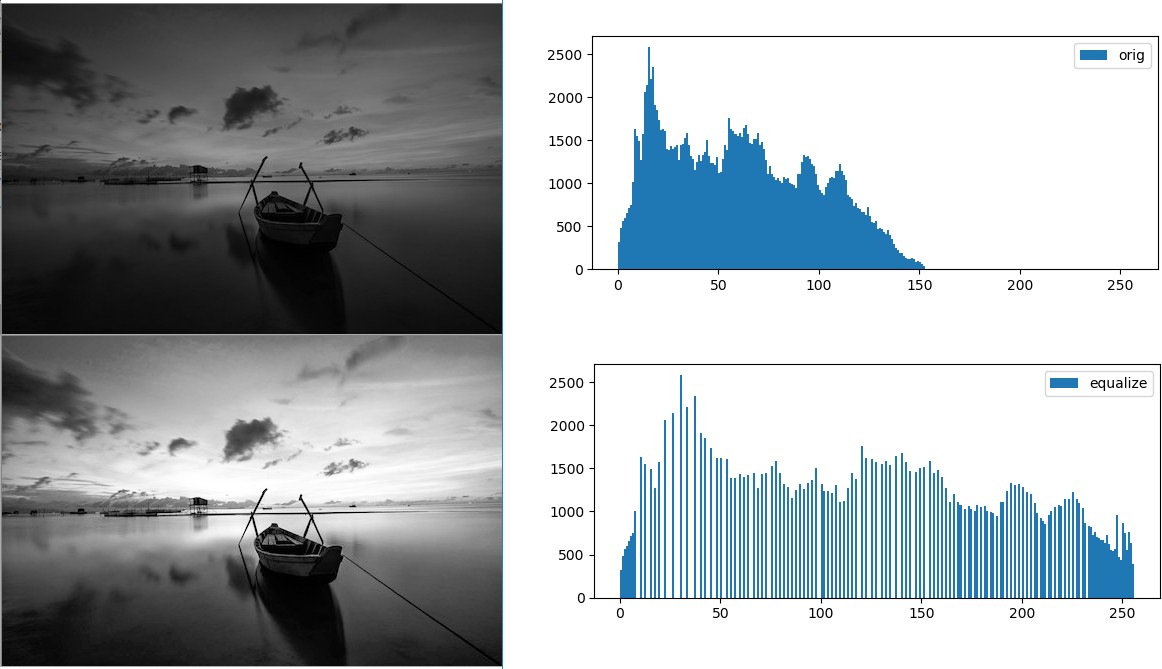
plt.subplot(2, 1, 2)

plt.hist(im\_equ.ravel(), 256, [0, 256], label="equalize") plt.legend()

plt.show()

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **彩色亮度直方图均衡化**

1

2

3

4

5

6

7

8

# 彩色图像亮度直方图均衡化

import cv2

# 读取原始图片

original = cv2.imread('../data/sunrise.jpg') cv2.imshow('Original', original)

# BRG空间转换为YUV空间

1. # YUV：亮度，色度，饱和度，其中Y通道为亮度通道
2. yuv = cv2.cvtColor(original, cv2.COLOR\_BGR2YUV)
3. print("yuv.shape:", yuv.shape) 12

13

14

15

16

17

18

yuv[..., 0] = cv2.equalizeHist(yuv[..., 0]) # 取出亮度通道，均衡化并赋回原

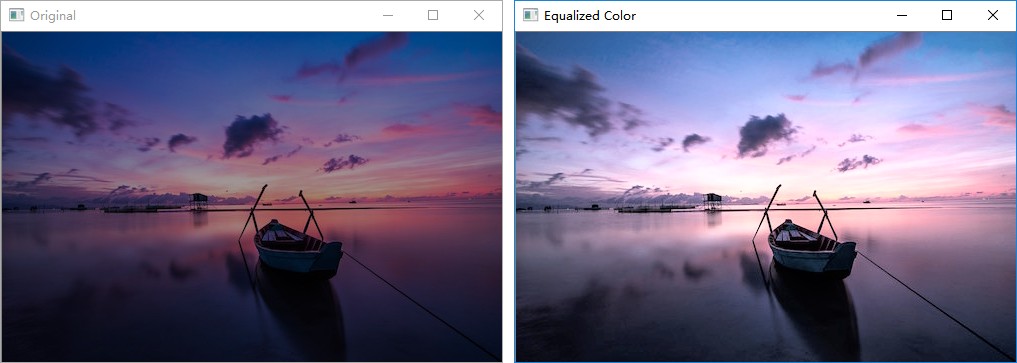
图像

equalized\_color = cv2.cvtColor(yuv, cv2.COLOR\_YUV2BGR) cv2.imshow('Equalized Color', equalized\_color)

cv2.waitKey()

cv2.destroyAllWindows()

执行结果：



1. **色彩提取**

从图片中提取特定颜色

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

import cv2

import numpy as np

im = cv2.imread("../data/opencv2.png") hsv = cv2.cvtColor(im, cv2.COLOR\_BGR2HSV) cv2.imshow('opencv', im)

# =============指定蓝色值的范围=============

# 蓝色H通道值为120，通常取120上下10的范围

# S通道和V通道通常取50~255间，饱和度太低、色调太暗计算出来的颜色不准确minBlue = np.array([110, 50, 50])

maxBlue = np.array([130, 255, 255]) # 确定蓝色区域

mask = cv2.inRange(hsv, minBlue, maxBlue) # 选取出掩模

# cv2.imshow("mask", mask)

# 通过掩码控制的按位与运算，锁定蓝色区域

blue = cv2.bitwise\_and(im, im, mask=mask) # 执行掩模运算

cv2.imshow('blue', blue)

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

# =============指定绿色值的范围=============

minGreen = np.array([50, 50, 50])

maxGreen = np.array([70, 255, 255]) # 确定绿色区域

mask = cv2.inRange(hsv, minGreen, maxGreen) # cv2.imshow("mask", mask)

# 通过掩码控制的按位与运算，锁定绿色区域

green = cv2.bitwise\_and(im, im, mask=mask) # 执行掩模运算

cv2.imshow('green', green)

# =============指定红色值的范围=============

minRed = np.array([0, 50, 50])

maxRed = np.array([30, 255, 255]) # 确定红色区域

mask = cv2.inRange(hsv, minRed, maxRed) # cv2.imshow("mask", mask)

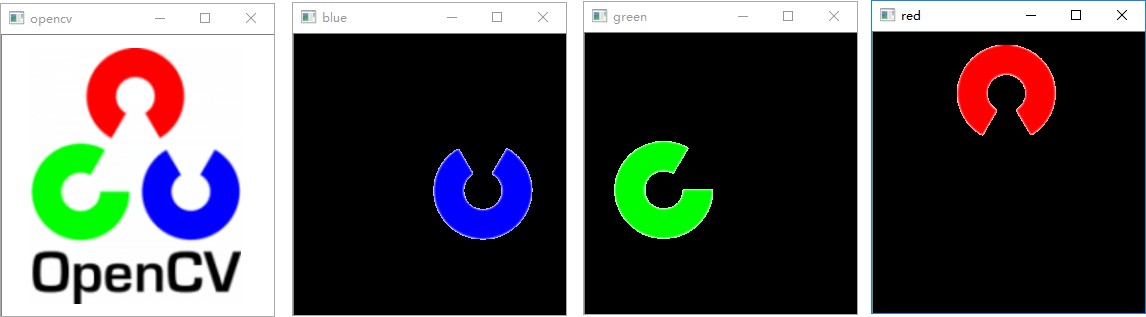
# 通过掩码控制的按位与运算，锁定红色区域

red = cv2.bitwise\_and(im, im, mask=mask) # 执行掩模运算

cv2.imshow('red', red)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **二值化与反二值化**

1

2

3

4

5

6

7

8

9

10

# 二值化处理

import cv2 as cv

# 读取图像

img = cv.imread("../data/lena.jpg", 0) cv.imshow("img", img) # 显示原始图像

# 二值化

t, rst = cv.threshold(img, 127, 255, cv.THRESH\_BINARY) cv.imshow("rst", rst) # 显示二值化图像

11

12

13

14

15

16

17

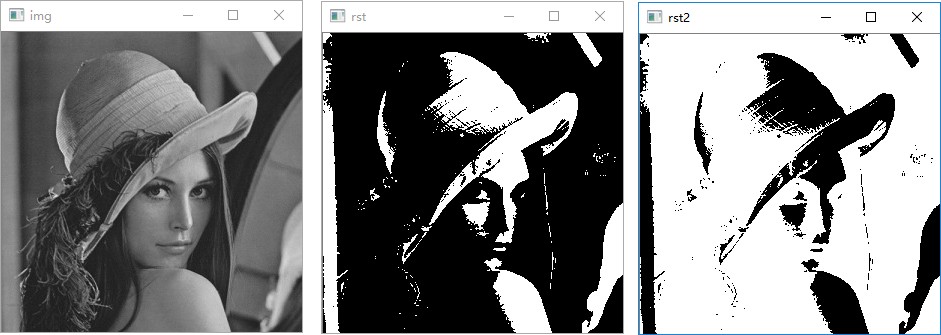
# 反二值化

t, rst2 = cv.threshold(img, 127, 255, cv.THRESH\_BINARY\_INV) cv.imshow("rst2", rst2) # 显示反二值化图像

cv.waitKey()

cv.destroyAllWindows()

执行结果：



1. **图像形态操作**
2. **图像翻转**

im = cv2.imread("../data/Linus.png") cv2.imshow("src", im)

# 0-垂直镜像

im\_flip0 = cv2.flip(im, 0) cv2.imshow("im\_flip0", im\_flip0)

# 1-水平镜像

im\_flip1 = cv2.flip(im, 1) cv2.imshow("im\_flip1", im\_flip1)

cv2.waitKey() cv2.destroyAllWindows()

# 图像翻转示例

import numpy as np import cv2

1

2

3

4

5

6

7

8

9

10

11

12

13

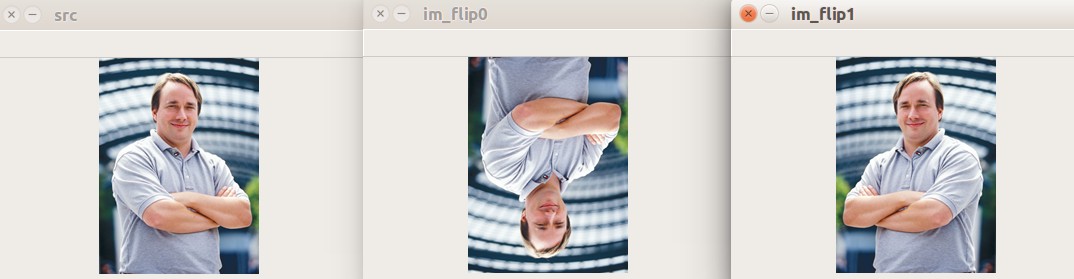
14

15

16

17

执行结果：



1. **图像位置变换**
   1. # 图像坐标位置变换
   2. import numpy as np
   3. import cv2 4

5 def translate(img, x, y):

6 """

1. 坐标平移变换
2. :param img: 原始图像数据
3. :param x:平移的x坐标
4. :param y:平移的y坐标
5. :return:返回平移后的图像

12 """

13 h, w = img.shape[:2] # 获取图像高、宽

14

15 # 定义平移矩阵

16 M = np.float32([[1, 0, x],

17 [0, 1, y]])

1. # 使用openCV仿射操作实现平移变换
2. shifted = cv2.warpAffine(img, M, (w, h)) # 第三个参数为输出图像尺寸

20

21 return shifted # 返回平移后的图像

22

23

24 def rotate(img, angle, center=None, scale=1.0): 25 """

1. 图像旋转变换
2. :param img: 原始图像数据
3. :param angle: 旋转角度
4. :param center: 旋转中心，如果为None则以原图中心为旋转中心
5. :param scale: 缩放比例，默认为1
6. :return: 返回旋转后的图像

32 """

33 h, w = img.shape[:2] # 获取图像高、宽

34

1. # 旋转中心默认为图像中心
2. if center is None:
3. center = (w / 2, h / 2)

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

# 计算旋转矩阵

M = cv2.getRotationMatrix2D(center, angle, scale)

# 使用openCV仿射变换实现函数旋转

rotated = cv2.warpAffine(img, M, (w, h))

return rotated # 返回旋转后的矩阵

if name == " main ": # 读取并显示原始图像

im = cv2.imread("../data/Linus.png") cv2.imshow("SrcImg", im)

# 图像向下移动50像素

shifted = translate(im, 0, 50) cv2.imshow("Shifted1", shifted)

# 图像向左移动40, 下移动40像素shifted = translate(im, -40, 40) cv2.imshow("Shifted2", shifted)

# 逆时针旋转45度

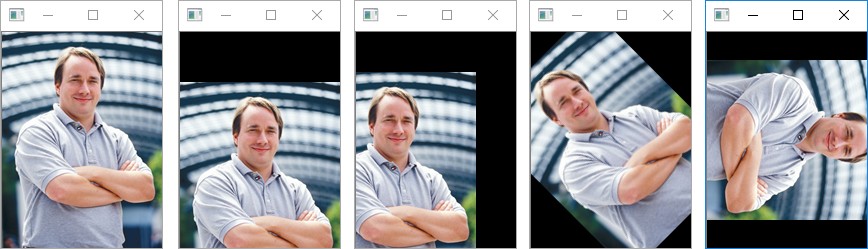
rotated = rotate(im, 45) cv2.imshow("rotated1", rotated)

# 顺时针旋转180度

rotated = rotate(im, -90) cv2.imshow("rorated2", rotated)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **图像缩放**
2. # 图像缩放示例
3. import numpy as np

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

import cv2

im = cv2.imread("../data/Linus.png") cv2.imshow("src", im)

h, w = im.shape[:2] # 获取图像尺寸

dst\_size = (int(w/2), int(h/2)) # 缩放目标尺寸，宽高均为原来1/2 resized = cv2.resize(im, dst\_size) # 执行缩放cv2.imshow("reduce", resized)

dst\_size = (200, 300) # 缩放目标尺寸，宽200，高300 method = cv2.INTER\_NEAREST # 最邻近插值

resized = cv2.resize(im, dst\_size, interpolation=method) # 执行缩放

cv2.imshow("NEAREST", resized)

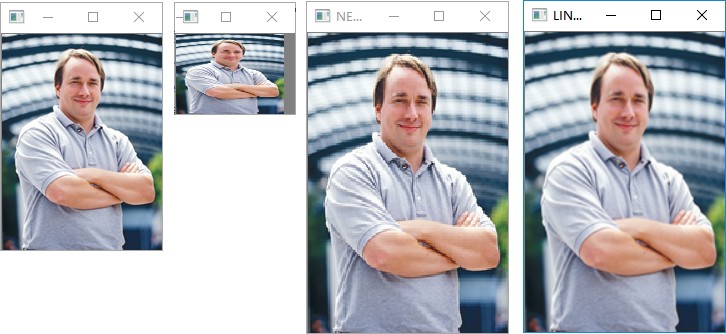
dst\_size = (200, 300) # 缩放目标尺寸，宽200，高300 method = cv2.INTER\_LINEAR # 双线性插值

resized = cv2.resize(im, dst\_size, interpolation=method) # 执行缩放

cv2.imshow("LINEAR", resized)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **图像裁剪**
2. import numpy as np
3. import cv2 3

4

5 # 图像随机裁剪

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

def random\_crop(im, w, h):

start\_x = np.random.randint(0, im.shape[1]) # 裁剪起始x像素

start\_y = np.random.randint(0, im.shape[0]) # 裁剪起始y像素

new\_img = im[start\_y:start\_y + h, start\_x: start\_x + w] # 执行裁剪

return new\_img

# 图像中心裁剪

def center\_crop(im, w, h):

start\_x = int(im.shape[1] / 2) - int(w / 2) # 裁剪起始x像素

start\_y = int(im.shape[0] / 2) - int(h / 2) # 裁剪起始y像素new\_img = im[start\_y:start\_y + h, start\_x: start\_x + w] # 执行裁剪return new\_img

im = cv2.imread("../data/banana\_1.png", 1)

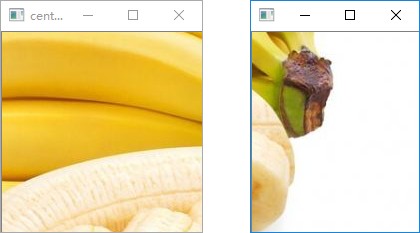
new\_img = random\_crop(im, 200, 200) # 随机裁剪

new\_img2 = center\_crop(im, 200, 200) # 中心裁剪

cv2.imshow("orig", im) cv2.imshow("random\_crop", new\_img) cv2.imshow("center\_crop", new\_img2)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **图像相加**

a = cv2.imread("../data/lena.jpg", 0)

b = cv2.imread("../data/lily\_square.png", 0)

dst1 = cv2.add(a, b) # 图像直接相加，会导致图像过亮、过白

# 加权求和：addWeighted

# 图像进行加权和计算时，要求src1和src2必须大小、类型相同

dst2 = cv2.addWeighted(a, 0.6, b, 0.4, 0) # 最后一个参数为亮度调节量

cv2.imshow("a", a)

cv2.imshow("b", b) cv2.imshow("dst1", dst1) cv2.imshow("dst2", dst2)

cv2.waitKey() cv2.destroyAllWindows()

# 图像相加示例

import cv2

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

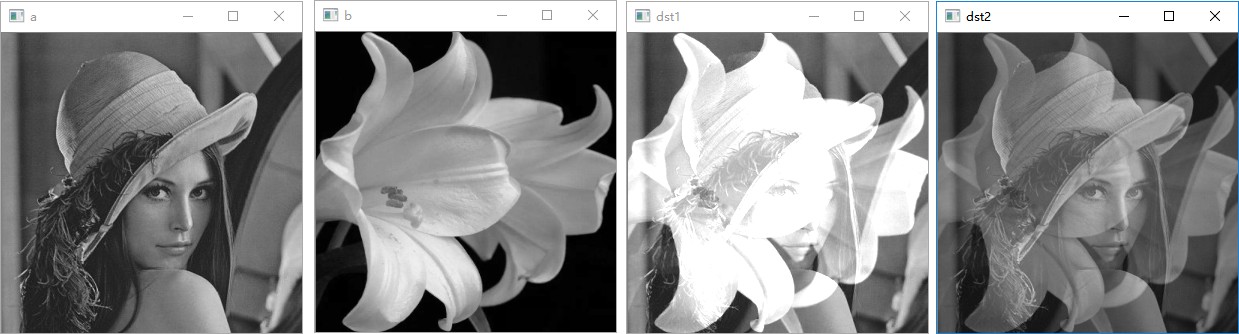
16

17

18

19

执行结果：



1. **图像相减**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# 图像相减运算示例

import cv2

a = cv2.imread("../data/3.png", 0) b = cv2.imread("../data/4.png", 0)

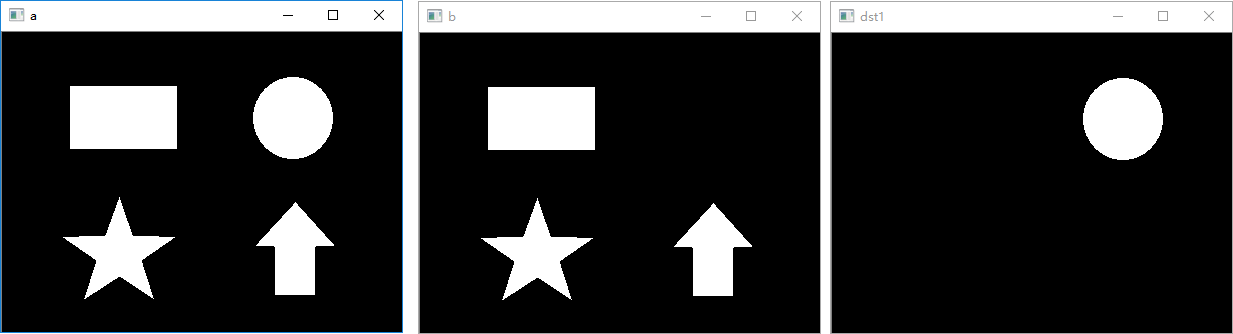
dst = cv2.subtract(a, b) # 两幅图像相减，是求出图像的差异

cv2.imshow("a", a)

cv2.imshow("b", b) cv2.imshow("dst1", dst)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **透视变换**

1

2

3

4

5

6

7

8

9

# 透视变换

import cv2

import numpy as np

img = cv2.imread('../data/pers.png')

rows, cols = img.shape[:2] print(rows, cols)

10

pts1 = np.float32([[58, 2], [167, 9], [8, 196], [126, 196]])# 输入图像四

个顶点坐标

pts2 = np.float32([[16, 2], [167, 8], [8, 196], [169, 196]])# 输出图像四

个顶点坐标

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

# 生成透视变换矩阵

M = cv2.getPerspectiveTransform(pts1, # 输入图像四个顶点坐标

pts2) # 输出图像四个顶点坐标

print(M.shape)

# 执行透视变换，返回变换后的图像

dst = cv2.warpPerspective(img, # 原始图像

M, # 3\*3的变换矩阵

(cols, rows)) # 输出图像大小

# 生成透视变换矩阵

M = cv2.getPerspectiveTransform(pts2, # 输入图像四个顶点坐标

pts1) # 输出图像四个顶点坐标

# 执行透视变换，返回变换后的图像

dst2 = cv2.warpPerspective(dst, # 原始图像

M, # 3\*3的变换矩阵

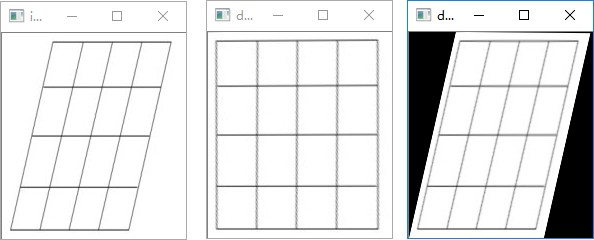
(cols, rows)) # 输出图像大小

cv2.imshow("img", img) cv2.imshow("dst", dst) cv2.imshow("dst2", dst2)

32

1. cv2.waitKey()
2. cv2.destroyAllWindows()

执行结果：



1. **图像腐蚀**

# 读取原始图像

im = cv2.imread("../data/5.png") cv2.imshow("im", im)

# 腐蚀

kernel = np.ones((3, 3), np.uint8) # 用于腐蚀计算的核

erosion = cv2.erode(im, # 原始图像

kernel, # 腐蚀核

iterations=3) # 迭代次数

cv2.imshow("erosion", erosion)

cv2.waitKey() cv2.destroyAllWindows()

# 图像腐蚀

import cv2

import numpy as np

1

2

3

4

5

6

7

8

9

10

11

12

13

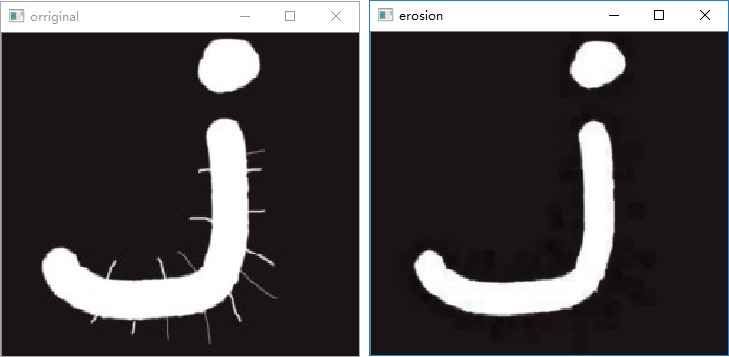
14

15

16

17

执行结果：



1. **图像膨胀**

# 读取原始图像

im = cv2.imread("../data/6.png") cv2.imshow("im", im)

# 膨胀

kernel = np.ones((3, 3), np.uint8) # 用于膨胀计算的核

dilation = cv2.dilate(im, # 原始图像

kernel, # 膨胀核

iterations=5) # 迭代次数

cv2.imshow("dilation", dilation)

cv2.waitKey() cv2.destroyAllWindows()

# 图像膨胀

import cv2

import numpy as np

1

2

3

4

5

6

7

8

9

10

11

12

13

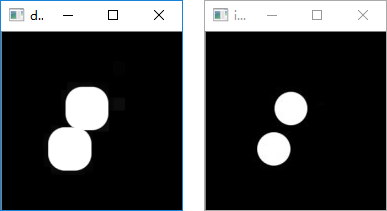
14

15

16

17

执行结果：



1. **图像开运算**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

# 开运算示例

import cv2

import numpy as np

# 读取原始图像

im1 = cv2.imread("../data/7.png") im2 = cv2.imread("../data/8.png")

# 执行开运算

k = np.ones((10, 10), np.uint8)

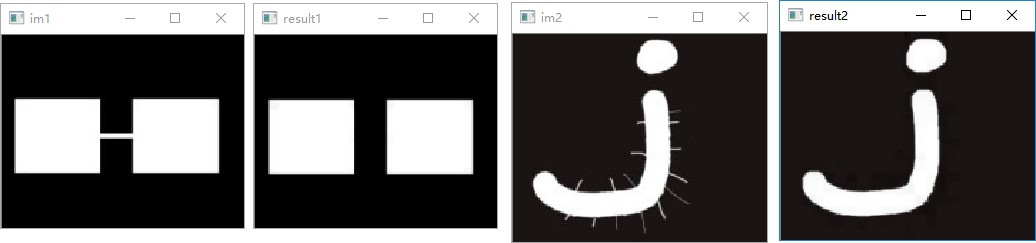
r1 = cv2.morphologyEx(im1, cv2.MORPH\_OPEN, k) r2 = cv2.morphologyEx(im2, cv2.MORPH\_OPEN, k)

cv2.imshow("im1", im1) cv2.imshow("result1", r1)

cv2.imshow("im2", im2) cv2.imshow("result2", r2)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **图像闭运算**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# 闭运算示例

import cv2

import numpy as np

# 读取图像

im1 = cv2.imread("../data/9.png") im2 = cv2.imread("../data/10.png")

# 闭运算

k = np.ones((8, 8), np.uint8)

r1 = cv2.morphologyEx(im1, cv2.MORPH\_CLOSE, k, iterations=2) r2 = cv2.morphologyEx(im2, cv2.MORPH\_CLOSE, k, iterations=2)

cv2.imshow("im1", im1)

15

16

17

18

19

20

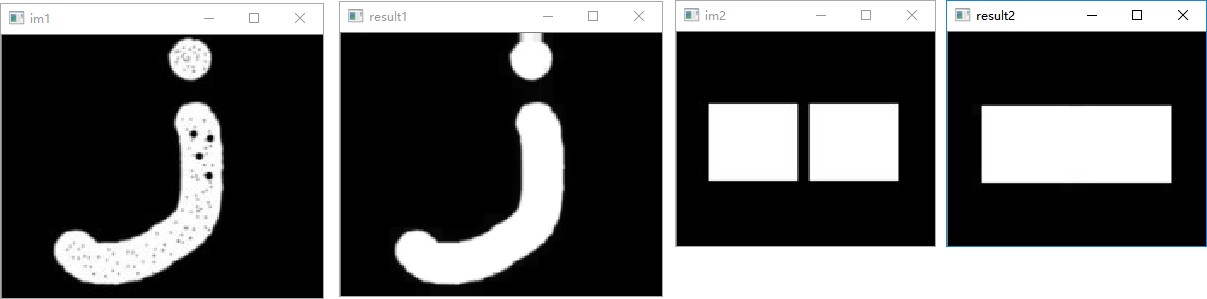
cv2.imshow("result1", r1) cv2.imshow("im2", im2)

cv2.imshow("result2", r2)

cv2.waitKey()

cv2.destroyAllWindows()

执行结果：



1. **形态学梯度**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# 形态学梯度示例

import cv2

import numpy as np

o = cv2.imread("../data/6.png")

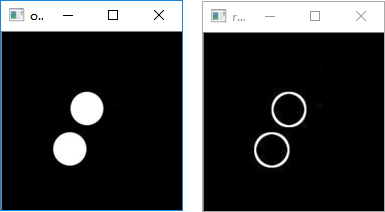
k = np.ones((3, 3), np.uint8)

r = cv2.morphologyEx(o, cv2.MORPH\_GRADIENT, k)

cv2.imshow("original", o) cv2.imshow("result", r)

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **图像梯度处理**
2. **模糊处理**

# 图像模糊处理示例

import cv2

import numpy as np

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

执行结果：

## 中值滤波

im = cv2.imread("../data/lena.jpg", 0) cv2.imshow("orig", im)

# 调用medianBlur中值模糊

# 第二个参数为滤波模板的尺寸大小，必须是大于1的奇数，如3、5、7 im\_median\_blur = cv2.medianBlur(im, 5) cv2.imshow('median\_blur', im\_median\_blur)

# 均值滤波

# 第二个参数为滤波模板的尺寸大小im\_mean\_blur = cv2.blur(im, (3, 3)) cv2.imshow("mean\_blur", im\_mean\_blur)

# 高斯滤波

# 第三个参数为高斯核在X方向的标准差

im\_gaussian\_blur = cv2.GaussianBlur(im, (5, 5), 3) cv2.imshow("gaussian\_blur", im\_gaussian\_blur)

# 使用高斯算子和filter2D自定义滤波操作

gaussan\_blur = np.array([ [1, 4, 7, 4, 1],

[4, 16, 26, 16, 4],

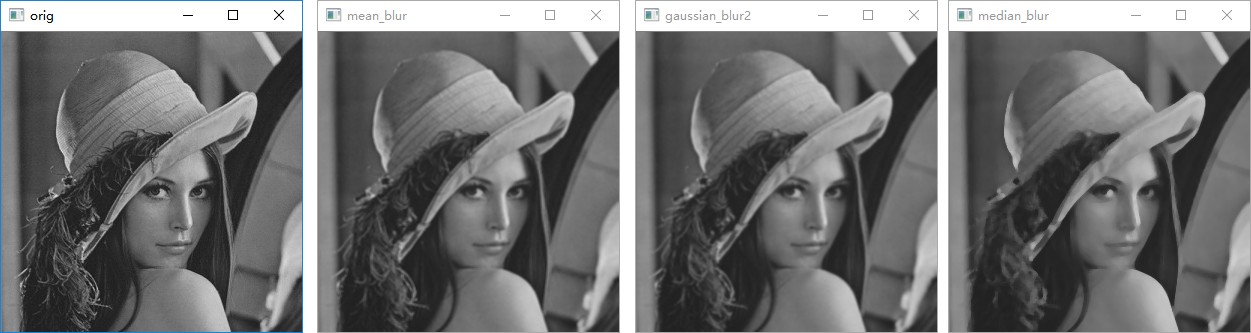
[7, 26, 41, 26, 7],

[4, 16, 26, 16, 4],

[1, 4, 7, 4, 1]], np.float32) / 273

# 使用filter2D, 第二个参数为目标图像的所需深度, -1表示和原图像相同im\_gaussian\_blur2 = cv2.filter2D(im, -1, gaussan\_blur) cv2.imshow("gaussian\_blur2", im\_gaussian\_blur2)

cv2.waitKey() cv2.destroyAllWindows()



1. **图像锐化处理**

im = cv2.imread("../data/lena.jpg", 0) cv2.imshow("orig", im)

# 锐化算子1

sharpen\_1 = np.array([[-1, -1, -1],

[-1, 9, -1],

[-1, -1, -1]])

# 使用filter2D进行滤波操作

im\_sharpen1 = cv2.filter2D(im, -1, sharpen\_1) cv2.imshow("sharpen\_1", im\_sharpen1)

# 锐化算子2

sharpen\_2 = np.array([[0, -1, 0],

[-1, 8, -1],

[0, 1, 0]]) / 4.0

# 使用filter2D进行滤波操作

im\_sharpen2 = cv2.filter2D(im, -1, sharpen\_2) cv2.imshow("sharpen\_2", im\_sharpen2)

cv2.waitKey() cv2.destroyAllWindows()

# 图像锐化示例

import cv2

import numpy as np

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

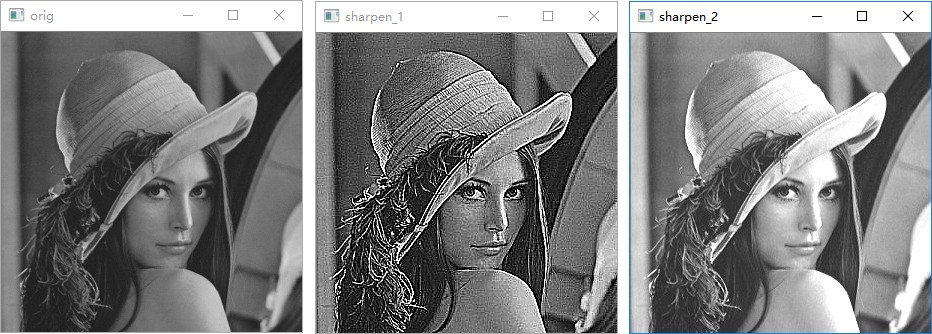
22

23

24

25

执行结果：



1. **边沿检测**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

# 边沿检测示例

import cv2 as cv

im = cv.imread('../data/lily.png', 0) cv.imshow('Original', im)

# # 水平方向滤波

# hsobel = cv.Sobel(im, cv.CV\_64F, 1, 0, ksize=5) # cv.imshow('H-Sobel', hsobel)

# # 垂直方向滤波

# vsobel = cv.Sobel(im, cv.CV\_64F, 0, 1, ksize=5) # cv.imshow('V-Sobel', vsobel)

# 两个方向滤波

sobel = cv.Sobel(im, cv.CV\_64F, 1, 1, ksize=5) cv.imshow('Sobel', sobel)

# Laplacian滤波：对细节反映更明显laplacian = cv.Laplacian(im, cv.CV\_64F) cv.imshow('Laplacian', laplacian)

# Canny边沿提取

canny = cv.Canny(im,

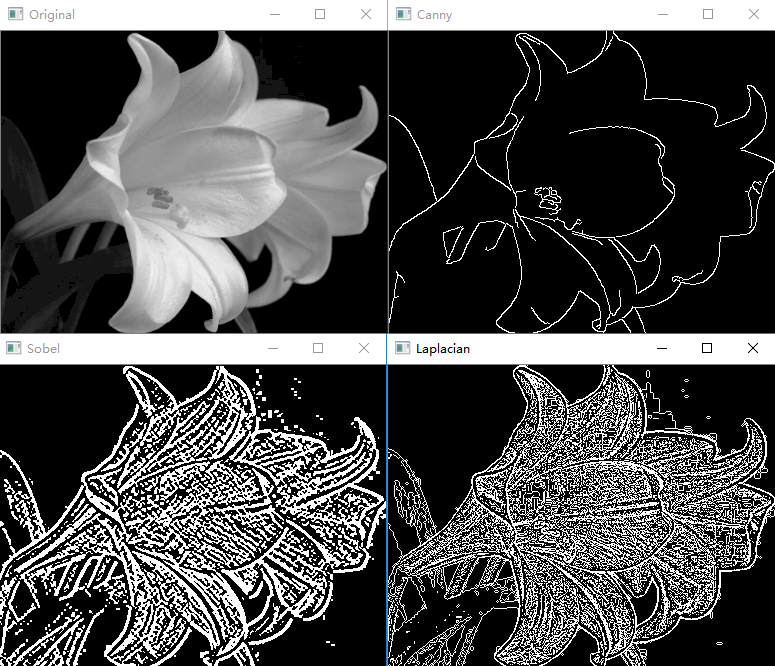
50, # 滞后阈值

240) # 模糊度

cv.imshow('Canny', canny)

cv.waitKey()

执行结果：



1. **轮廓处理**

边缘检测虽然能够检测出边缘，但边缘是不连续的，检测到的边缘并不是一个整体。图像轮廓是指将边 缘连接起来形成的一个整体，用于后续的计算。

OpenCV提供了查找图像轮廓的函数cv2.ﬁndContours（），该函数能够查找图像内的轮廓信息，而函 数cv2.drawContours（）能够将轮廓绘制出来。图像轮廓是图像中非常重要的一个特征信息，通过对 图像轮廓的操作，我们能够获取目标图像的大小、位置、方向等信息。一个轮廓对应着一系列的点，这 些点以某种方式表示图像中的一条曲线。

1. **查找并绘制轮廓**

查找轮廓函数：cv2.ﬁndContours

语法格式：image,contours,hierarchy=cv2.ﬁndContours（image,mode,method） 返回值

image： 与 函 数 参 数 中 的 原 始 图 像 image 一 致 contours：返回的轮廓。该返回值返回的是一组轮廓信息，每个轮廓都是由若干个点所 构成的（每个轮廓为一个list表示）。例如，contours[i]是第i个轮廓（下标从0开

始 ）,contours[i][j] 是 第 i 个 轮 廓 内 的 第 j 个 点 hierarchy：图像的拓扑信息（反映轮廓层次）。图像内的轮廓可能位于不同的位置。比 如，一个轮廓在另一个轮廓的内部。在这种情况下，我们将外部的轮廓称为父轮廓，内 部的轮廓称为子轮廓。按照上述关系分类，一幅图像中所有轮廓之间就建立了父子关

系。每个轮廓contours[i]对应4个元素来说明当前轮廓的层次关系。其形式为： [Next,Previous,First\_Child,Parent]，分别表示后一个轮廓的索引编号、前一个轮廓的 索引编号、第1个子轮廓的索引编号、父轮廓的索引编号

参数

image：原始图像。灰度图像会被自动处理为二值图像。在实际操作时，可以根据需 要，预先使用阈值处理等函数将待查找轮廓的图像处理为二值图像。 mode：轮廓检索模式，有以下取值和含义：

|  |  |
| --- | --- |
| **取值** | **含义** |
| cv2.RETR\_EXTERNAL | 只检测外轮廓 |
| cv2.RETR\_LIST | 对检测到的轮廓不建立等级关系 |
| cv2.RETR\_CCOMP | 检索所有轮廓并将它们组织成两级层次结构，上面的一层为外边界，下面的一层为内孔的边界 |
| cv2.RETR\_TREE | 建立一个等级树结构的轮廓 |

method：轮廓的近似方法，主要有如下取值：

|  |  |
| --- | --- |
| **取值** | **含义** |
| cv2.CHAIN\_APPROX\_NONE | 存储所有的轮廓点，相邻两个点的像素位置差不超过1，即max（abs（x1-x2）,abs（y2-y1））  =1 |
| cv2.CHAIN\_APPROX\_SIMPLE | 压缩水平方向、垂直方向、对角线方向的元素， 只保留该方向的终点坐标 |
| cv2.CHAIN\_APPROX\_TC89\_L1 | 使用teh-Chinl chain近似算法的一种风格 |
| cv2.CHAIN\_APPROX\_TC89\_KCOS | 使用teh-Chinl chain近似算法的一种风格 |

注意事项

待处理的源图像必须是灰度二值图

都是从黑色背景中查找白色对象。因此，对象必须是白色的，背景必须是黑色的 在OpenCV 4.x中，函数cv2.ﬁndContours（）仅有两个返回值

绘制轮廓：drawContours函数

语法格式：image=cv2.drawContours(image, contours,contourIdx, color)

参数

image：待绘制轮廓的图像

contours：需要绘制的轮廓，该参数的类型与函数 cv2.ﬁndContours（）的输出contours 相 同 ， 都 是 list 类 型contourIdx：需要绘制的边缘索引，告诉函数cv2.drawContours（）要绘制某一条轮 廓还是全部轮廓。如果该参数是一个整数或者为零，则表示绘制对应索引号的轮廓；如 果该值为负数（通常为“-1”），则表示绘制全部轮廓。

color：绘制的颜色，用BGR格式表示

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

# 查找图像轮廓

import cv2

import numpy as np

im = cv2.imread("../data/3.png") cv2.imshow("orig", im)

gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

# 图像二值化处理，将大于阈值的设置为最大值，其它设置为0

ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

16

17

18

19

20

21

22

23

24

25

26

27

# 查找图像边沿：cv2.findContours

img, contours, hierarchy = cv2.findContours(binary, # 二值化处理后的图像

cv2.RETR\_EXTERNAL, # 只检测

外轮廓

cv2.CHAIN\_APPROX\_NONE) # 存

储所有的轮廓点

# 打印所有轮廓值

arr\_cnt = np.array(contours) print(arr\_cnt[0].shape) print(arr\_cnt[1].shape) print(arr\_cnt[2].shape) print(arr\_cnt[3].shape)

# print(arr\_cnt[0])

# 绘制边沿

im\_cnt = cv2.drawContours(im, # 绘制图像

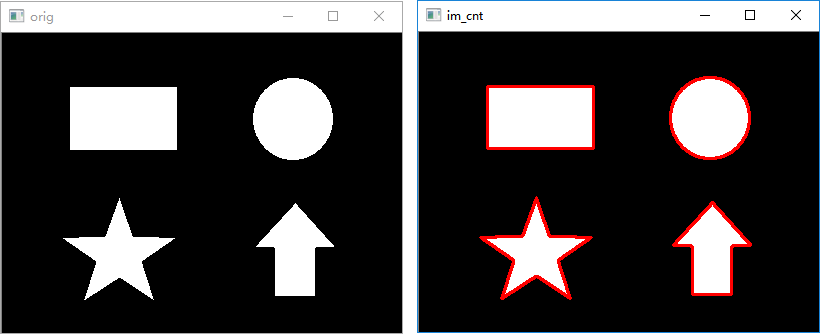
contours, # 轮廓点列表

|  |  |  |
| --- | --- | --- |
| 28 | -1, | # 绘制全部轮廓 |
| 29 | (0, | 0, 255), # 轮廓颜色：红色 |
| 30 | 2) | # 轮廓粗细 |
| 31 | cv2.imshow("im\_cnt", im\_cnt) |  |

32

1. cv2.waitKey()
2. cv2.destroyAllWindows()

执行结果：



1. **绘制矩形包围框**

函数cv2.boundingRect（）能够绘制轮廓的矩形边界。该函数的语法格式为：

1. retval = cv2.boundingRect(array) # 格式一
2. x,y,w,h = cv2.boundingRect(array) # 格式二

3 """

1. 参数：
2. array：是灰度图像或轮廓
3. 返回值：
4. retval：表示返回的矩形边界的左上角顶点的坐标值及矩形边界的宽度和高度
5. x, y, w, h: 矩形边界左上角顶点的x坐标、y坐标、宽度、高度

9 """

代码：

1

2

3

4

5

6

7

8

9

10

11

# 绘制图像矩形轮廓

import cv2

import numpy as np

im = cv2.imread("../data/cloud.png", 0)

cv2.imshow("orig", im)

12

# 提取图像轮廓

ret, binary = cv2.threshold(im, 127, 255, cv2.THRESH\_BINARY) img, contours, hierarchy = cv2.findContours(binary,

cv2.RETR\_LIST, # 不建立等级

关系

cv2.CHAIN\_APPROX\_NONE) # 存

储所有的轮廓点

13

14

15

16

17

18

19

20

print("contours[0].shape:", contours[0].shape)

# 返回轮廓定点及边长

x, y, w, h = cv2.boundingRect(contours[0]) # 计算矩形包围框的x,y,w,h print("x:", x, "y:", y, "w:", w, "h:", h)

21

22

23

24

25

26

27

28

29

30

# 绘制矩形包围框

brcnt = np.array([[[x, y]], [[x + w, y]], [[x + w, y + h]], [[x, y + h]]])

cv2.drawContours(im, # 绘制图像

[brcnt], # 轮廓点列表

-1, # 绘制全部轮廓

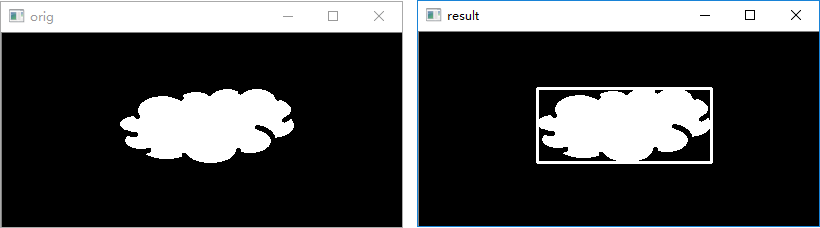
(255, 255, 255), # 轮廓颜色：白色

2) # 轮廓粗细

cv2.imshow("result", im) # 显示绘制后的图像

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **绘制圆形包围框**

函数 cv2.minEnclosingCircle（）通过迭代算法构造一个对象的面积最小包围圆形。该函数的语法格式为：

1 center,radius=cv2.minEnclosingCircle(points) 2 """

1. 参数：
2. points: 轮廓数组
3. 返回值：
4. center: 最小包围圆形的中心
5. radius: 最小包围圆形的半径

8 """

代码：

1 # 绘制最小圆形

2

3

4

5

6

7

8

9

10

import cv2

import numpy as np

im = cv2.imread("../data/cloud.png", 0) cv2.imshow("orig", im)

# 提取图像轮廓

ret, binary = cv2.threshold(im, 127, 255, cv2.THRESH\_BINARY)

img, contours, hierarchy = cv2.findContours(binary, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

11

12

13

14

15

16

17

18

19

20

(x, y), radius = cv2.minEnclosingCircle(contours[0]) center = (int(x), int(y))

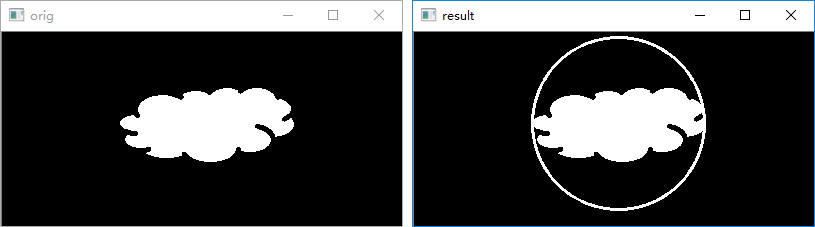
radius = int(radius)

cv2.circle(im, center, radius, (255, 255, 255), 2) # 绘制圆

cv2.imshow("result", im) # 显示绘制后的图像

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **绘制最佳拟合椭圆**

函数cv2.ﬁtEllipse（）可以用来构造最优拟合椭圆。该函数的语法格式是：

1 retval=cv2.fitEllipse(points) 2 """

1. 参数：
2. points: 轮廓
3. 返回值：
4. retval: 为RotatedRect类型的值，包含外接矩形的质心、宽、高、旋转角度等参数信息， 这些信息正好与椭圆的中心点、轴长度、旋转角度等信息吻合

7 """

代码：

1 # 绘制最优拟合椭圆

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

import cv2

import numpy as np

im = cv2.imread("../data/cloud.png")

gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY) cv2.imshow("orig", gray)

# 提取图像轮廓

ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY) img, contours, hierarchy = cv2.findContours(binary,

cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

ellipse = cv2.fitEllipse(contours[0]) # 拟合最优椭圆

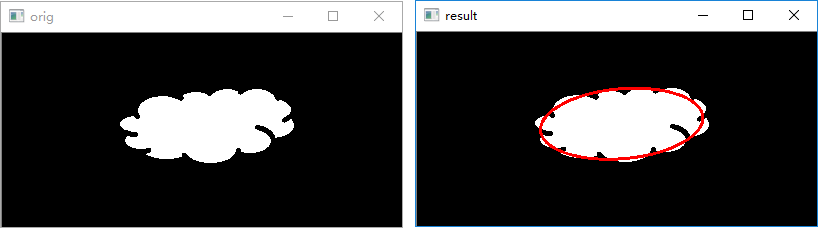
print("ellipse:", ellipse)

cv2.ellipse(im, ellipse, (0, 0, 255), 2) # 绘制椭圆

cv2.imshow("result", im) # 显示绘制后的图像

cv2.waitKey() cv2.destroyAllWindows()

执行结果：



1. **逼近多边形**

函数cv2.approxPolyDP（）用来构造指定精度的逼近多边形曲线。该函数的语法格式为：

1 approxCurve = cv2.approxPolyDP(curve,epsilon,closed) 2 """

1. 参数：
2. curve: 轮廓
3. epsilon: 精度，原始轮廓的边界点与逼近多边形边界之间的最大距离
4. closed: 布尔类型，该值为True时，逼近多边形是封闭的；否则，逼近多边形是不封闭的
5. 返回值：
6. approxCurve: 逼近多边形的点集

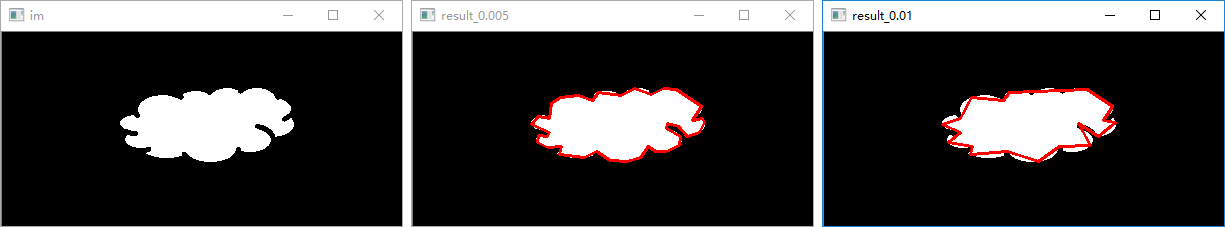
9 """

代码：

* 1. # 构建多边形，逼近轮廓
  2. import cv2
  3. import numpy as np 4

1. im = cv2.imread("../data/cloud.png")
2. cv2.imshow("im", im)
3. gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY) 8
4. # 提取图像轮廓
5. ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)
6. img, contours, hierarchy = cv2.findContours(binary,
7. cv2.RETR\_LIST,
8. cv2.CHAIN\_APPROX\_NONE)
9. # 精度一
10. adp = im.copy()
11. epsilon = 0.005 \* cv2.arcLength(contours[0], True) # 精度，根据周长计算
12. approx = cv2.approxPolyDP(contours[0], epsilon, True) # 构造多边形
13. adp = cv2.drawContours(adp, [approx], 0, (0, 0, 255), 2) # 绘制多边形
14. cv2.imshow("result\_0.005", adp)
15. # 精度二
16. adp2 = im.copy()
17. epsilon = 0.01 \* cv2.arcLength(contours[0], True) # 精度，根据周长计算
18. approx = cv2.approxPolyDP(contours[0], epsilon, True) # 构造多边形
19. adp = cv2.drawContours(adp2, [approx], 0, (0, 0, 255), 2) # 绘制多边形
20. cv2.imshow("result\_0.01", adp2) 26
21. cv2.waitKey()
22. cv2.destroyAllWindows()

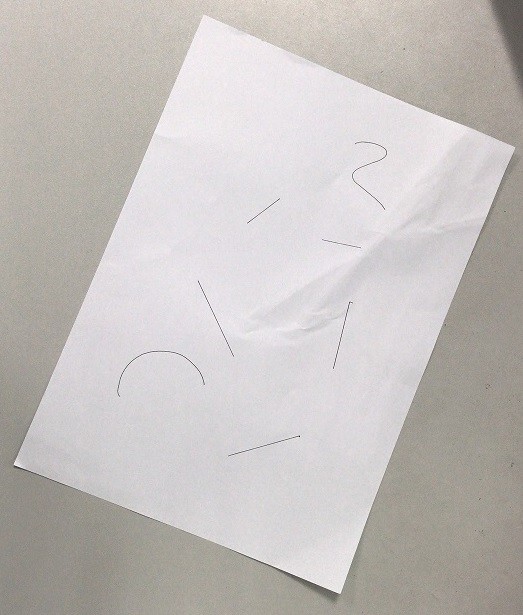
执行结果：



**三、综合案例**

1. **利用OpenCV实现图像校正**
2. **任务描述**

我们对图像中的目标进行分析和检测时，目标往往具有一定的倾斜角度，自然条件下拍摄的图像，完全 平正是很少的。因此，需要将倾斜的目标“扶正”的过程就就叫做图像矫正。该案例中使用的原始图像如 下：



1. **代码**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

# 图像校正示例

import cv2 import imutils

import numpy as np

im = cv2.imread("../data/paper.jpg")

gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY) cv2.imshow('im', im)

# 模糊

blurred = cv2.GaussianBlur(gray, (5, 5), 0) # 膨胀

dilate = cv2.dilate(blurred,

cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3)))

# 检测边沿

edged = cv2.Canny(dilate, # 原始图像

17

18

1. # 轮廓检测

30, 120, # 滞后阈值、模糊度

3) # 孔径大小

1. cnts = cv2.findContours(edged.copy(),
2. cv2.RETR\_EXTERNAL, # 只检测外轮廓
3. cv2.CHAIN\_APPROX\_SIMPLE) # 只保留该方向的终点坐标
4. cnts = cnts[0] if imutils.is\_cv2() else cnts[1] # 判断是opencv2还是

opencv3

1. docCnt = None 25
2. # 绘制轮廓
3. # im\_cnt = cv2.drawContours(im, # 绘制图像
4. # cnts, # 轮廓点列表
5. # -1, # 绘制全部轮廓

30 # (0, 0, 255), # 轮廓颜色：红色

1. # 2) # 轮廓粗细
2. # cv2.imshow("im\_cnt", im\_cnt) 33
3. # 计算轮廓面积，并排序
4. if len(cnts) > 0:
5. cnts = sorted(cnts, # 数据
6. key=cv2.contourArea, # 排序依据，根据contourArea函数结果排序
7. reverse=True)
8. for c in cnts:
9. peri = cv2.arcLength(c, True) # 计算轮廓周长
10. approx = cv2.approxPolyDP(c, 0.02 \* peri, True) # 轮廓多边形拟合
11. # 轮廓为4个点表示找到纸张
12. if len(approx) == 4:
13. docCnt = approx
14. break

46

47 print(docCnt) 48

1. # 用圆圈标记处角点
2. points = []
3. for peak in docCnt:
4. peak = peak[0]
5. # 绘制圆
6. cv2.circle(im, # 绘制图像
7. tuple(peak), 10, # 圆心、半径

56 (0, 0, 255), 2) # 颜色、粗细

1. points.append(peak) # 添加到列表
2. print(points) 59

60 # 校正

61

62

63

64

65

66

67

68

src = np.float32([points[0], points[1], points[2], points[3]]) # 原来逆

时针方向四个点

dst = np.float32([[0, 0], [0, 488], [337, 488], [337, 0]]) # 对应变换后

逆时针方向四个点

m = cv2.getPerspectiveTransform(src, dst) # 生成透视变换矩阵

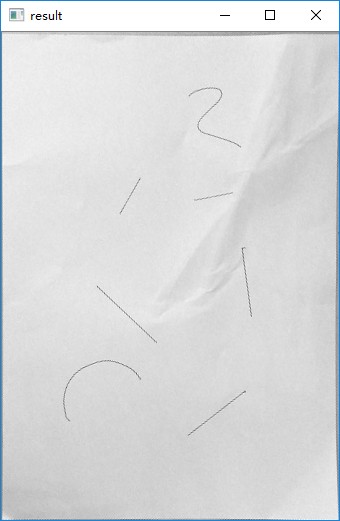
result = cv2.warpPerspective(gray.copy(), m, (337, 488)) # 透视变换

cv2.imshow("result", result) # 显示透视变换结果

cv2.waitKey()

cv2.destroyAllWindows()

### 执行结果



1. **利用OpenCV检测芯片瑕疵**
2. **任务描述**

利用图像技术，检测出芯片镀盘区域瑕疵。样本图像中，粉红色区域为镀盘区域，镀盘内部空洞为瑕疵 区域，利用图像技术检测镀盘是否存在瑕疵，如果存在则将瑕疵区域标记出来。



1. **代码**
   1. import cv2
   2. import numpy as np
   3. import math 4
2. # 第一步：图像预处理
3. ## 1. 转换成灰度图像，进行二值化处理
4. im\_cpu = cv2.imread("../../data/CPU3.png")
5. im\_gray = cv2.cvtColor(im\_cpu, cv2.COLOR\_BGR2GRAY) # 转换成灰度图像

9

1. # 提取出度盘轮廓
2. ret, im\_bin = cv2.threshold(im\_gray, 162, 255, cv2.THRESH\_BINARY) # 图像二值化
3. cv2.imshow("im\_cpu", im\_cpu)
4. cv2.imshow("im\_gray", im\_gray)
5. cv2.imshow("im\_bin", im\_bin) 15
6. # 提取轮廓、绘制边沿
7. img, contours, hierarchy = cv2.findContours(im\_bin, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

18

1. # 绘制前景对象轮廓
2. mask = np.zeros(im\_bin.shape, np.uint8)
3. mask = cv2.drawContours(mask, contours, -1, (255, 0, 0), -1) # 绘制实心轮廓
4. cv2.imshow("mask", mask) 23
5. # 前景实心轮廓图和二值化图相减
6. im\_sub = cv2.subtract(mask, im\_bin)
7. cv2.imshow("im\_sub", im\_sub) 27

28 # 图像闭运算，先膨胀后腐蚀，去除内部毛刺

29 k = np.ones((10, 10), np.uint8)

1. im\_close = cv2.morphologyEx(im\_sub, cv2.MORPH\_CLOSE, k, iterations=3)
2. cv2.imshow("im\_close", im\_close) 32
3. # 提取、绘制轮廓、计算面积
4. img, contours, hierarchy = cv2.findContours(im\_close, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

35

36 (x, y), radius = cv2.minEnclosingCircle(contours[1])

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

center = (int(x), int(y)) radius = int(radius)

print("center:", center, " radius:", radius) cv2.circle(im\_close, center, radius, (255, 0, 0), 2) # 绘制圆

cv2.imshow("im\_gaussian\_blur2", im\_close)

# 在原始图片上绘制瑕疵

cv2.circle(im\_cpu, center, radius, (0, 0, 255), 2) # 绘制圆

cv2.imshow("im\_cpu2", im\_cpu)

#计算面积

area = math.pi \* radius \* radius print("area:", area)

if area > 12:

print("度盘表面有缺陷")

cv2.waitKey() cv2.destroyAllWindows()

### 执行结果

