核心操作

# 图像读写

## 知识点

图像的读取、显示与保存

## 读取图像

import cv2

img = cv2.imread(r"1.jpg") cv2.imshow("pic show", img) cv2.waitKey(0)

图像读取的返回值是一个numpy数据类型



print(type(img))

#out:<class 'numpy.ndarray'>

C++

#include<opencv2/opencv.hpp>

int main() {

cv::Mat img = cv::imread("1.jpg");

cv::imshow("pic show", img); cv::waitKey(0);

}

## 创建图片并保存

python（day01/test02.py)

|  |  |  |  |
| --- | --- | --- | --- |
| img[..., | 0] | = | 255 |
| img[..., | 1] | = | 0 |
| img[..., | 2] | = | 0 |

打开保存的图片，我们发现图片是纯蓝色的，由此我们可以指导opencv读入的图片是BGR格式

import numpy as np import cv2

img = np.empty((200, 200, 3), np.uint8)

cv2.imwrite("2.jpg", img)



C++

#include<opencv2/opencv.hpp>

int main() {

cv::Mat img = cv::Mat(200, 200, CV\_8UC3, cv::Scalar(255, 0, 0)); cv::imwrite("2.jpg", img);

}

# 视频读取

## 知识点

读取视频或摄像头，并显示

## 读取视频与摄像头

python（day01/test10.py)

import cv2

cap = cv2.VideoCapture(0)

#cap = cv2.VideoCapture("1.mp4") while True:

ret, frame = cap.read() cv2.imshow('frame', frame)

if cv2.waitKey(1) & 0xFF == ord('q'): break

cap.release() cv2.destroyAllWindows()

# 色彩空间与转换

## 知识点

色彩空间 RGB/RGBA/GRAY/HSV/YUV

通道分离理解HSV

## 色彩空间的转换

python（day01/test08.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.cvtColor(src, cv2.COLOR\_BGR2GRAY) # dst = cv2.cvtColor(src, cv2.COLOR\_BGR2YUV)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

## 通道分离

python（day01/test09.py)

import cv2

img = cv2.imread(r"1.jpg")

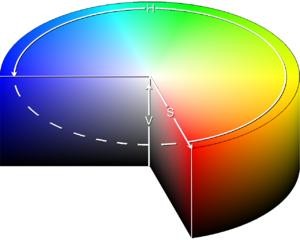
img[..., 0] = 0

img[..., 1] = 0

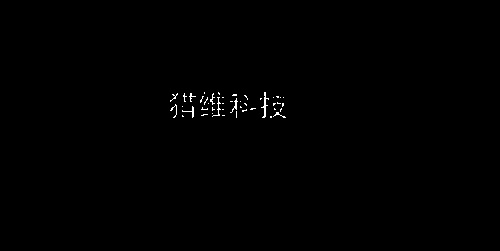
cv2.imshow("dst show", img) cv2.waitKey(0)

HSV**颜色空间**

HSV 格式中，H（色彩/色度）的取值范围是 [0，179]，S（饱和度）的取值范围 [0，255]，V（亮度）的取值范围 [0，255]。但是不同的软件使用的值可能不同。所以当你需要拿 OpenCV 的 HSV值与其他软件的 HSV 值进行对比时，一定要记得归一化。



HSV中，我们按颜色提取变得会更加方便例：提取图像中的字 python（day01/test11.py)



import cv2

import numpy as np

img = cv2.imread("3.jpg")

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

lower\_blue = np.array([100, 200, 100])

upper\_blue = np.array([200, 255, 200])

mask = cv2.inRange(hsv, lower\_blue, upper\_blue) res = cv2.bitwise\_and(img, img, mask=mask)

cv2.imshow('frame', img) cv2.imshow('mask', mask)

cv2.waitKey(0)



知道自己要提取的HSV颜色值

import cv2

import numpy as np

color=np.uint8([[[21,94,214]]]) hsv\_color=cv2.cvtColor(color,cv2.COLOR\_BGR2HSV) print(hsv\_color)

# 基本图形绘制

## 知识点

直线矩形圆 椭圆

多边形文字

## 直线、圆、椭圆、矩形

python（day01/test05.py)

import cv2

img = cv2.imread(r"1.jpg")

# cv2.line(img, (100, 30), (210, 180), color=(0, 0, 255), thickness=2)

# cv2.circle(img, (50, 50), 30, (0, 0, 255), 1)

cv2.rectangle(img,(100,30),(210,180),color=(0,0,255),thickness=2)

# cv2.ellipse(img, (100, 100), (100, 50), 0, 0, 360, (255, 0, 0), -1)

cv2.imshow("pic show", img) cv2.waitKey(0)

## 多边形

python（day01/test06.py)

import cv2

import numpy as np

img = cv2.imread(r"1.jpg") # 定义四个顶点坐标

pts = np.array([[10, 5], [50, 10], [70, 20], [20, 30]], np.int32)

# 顶点个数：4，矩阵变成4\*1\*2维

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

cv2.polylines(img, [pts], True, (0, 0, 255), 2)

cv2.imshow("pic show", img) cv2.waitKey(0)

## 文字

python（day01/test07.py)

import cv2

img = cv2.imread(r"1.jpg")

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(img, 'beautiful girl', (10, 30), font, 1, (0, 0, 255), 1, lineType=cv2.LINE\_AA)

cv2.imshow("pic show", img)cv2.waitKey(0)

# 阀值操作

## 知识点

理解二值化 OTSU二值化简单阀值

自适应阀值

OTSU**二值化**

python（day01/test00.py)

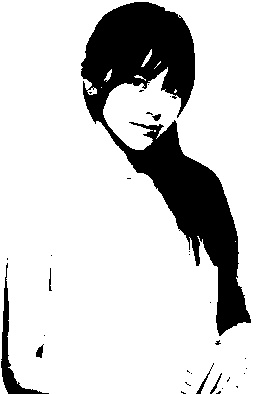
import cv2

img = cv2.imread("1.jpg")

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

ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)

cv2.imshow("gray",gray) cv2.imshow('binary', binary) cv2.waitKey(0)





## 简单阀值

与名字一样，这种方法非常简单。但像素值高于阈值时，我们给这个像素赋予一个新值（可能是白

色），否则我们给它赋予另外一种颜色（也许是黑色）。这个函数就是 cv2.threshhold()。这个函数的第一个参数就是原图像，原图像应该是灰度图。第二个参数就是用来对像素值进行分类的阈值。第三个参数就是当像素值高于（有时是小于）阈值时应该被赋予的新的像素值。OpenCV提供了多种不同的阈值方法，这是有第四个参数来决定的。这些方法包括：

* cv2.THRESH\_BINARY
* cv2.THRESH\_BINARY\_INV
* cv2.THRESH\_TRUNC
* cv2.THRESH\_TOZERO
* cv2.THRESH\_TOZERO\_INV

python（day01/test15.py)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('5.jpg',0)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY) ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV) ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC) ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO) ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)

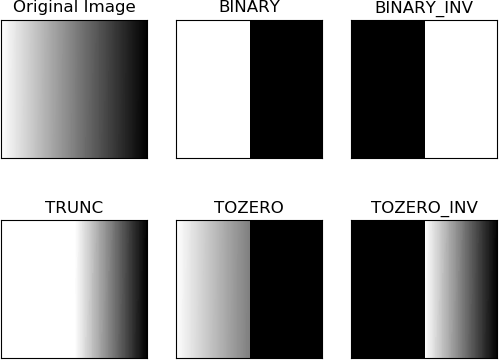
titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']

images = [img, 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()



## 自适应阀值

前面的部分我们使用是全局阈值，整幅图像采用同一个数作为阈值。当时这种方法并不适应与所有情 况，尤其是当同一幅图像上的不同部分的具有不同亮度时。这种情况下我们需要采用自适应阈值。此时的阈值是根据图像上的每一个小区域计算与其对应的阈值。因此在同一幅图像上的不同区域采用的是不同的阈值，从而使我们能在亮度不同的情况下得到更好的结果。

这种方法需要我们指定三个参数，返回值只有一个。

* Adaptive Method- 指定计算阈值的方法。
* cv2.ADPTIVE\_THRESH\_MEAN\_C：阈值取自相邻区域的平均值
* cv2.ADPTIVE\_THRESH\_GAUSSIAN\_C：阈值取值相邻区域的加权和，权重为一个高斯窗口。
* Block Size - 邻域大小（用来计算阈值的区域大小）。
* C - 这就是是一个常数，阈值就等于的平均值或者加权平均值减去这个常数

python（day01/test14.py)

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('4.jpg', 0)

img = cv2.GaussianBlur(img, (5, 5), 0)

ret, th1 = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

th2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C,

cv2.THRESH\_BINARY, 11, 2)

th3 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

cv2.THRESH\_BINARY, 11, 2)

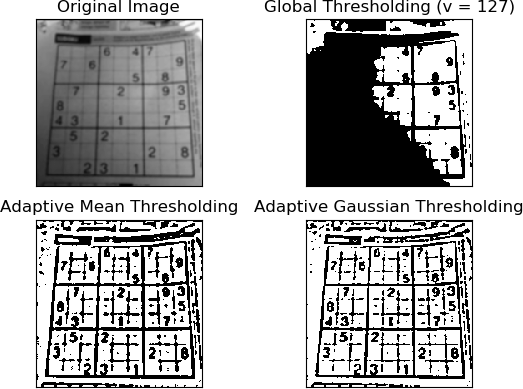
titles = ['Original Image', 'Global Thresholding (v = 127)',

'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding'] images = [img, th1, th2, th3]

for i in range(4):

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

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



# 图像上的运算

## 知识点

学习图像上的算术运算，加法，减法，图像混合等。

我们将要学习的函数与有：cv2.add()，cv2.addWeighted() 等。

## 加减法

python（day01/test16.py)

import cv2

import numpy as np

x = np.uint8([250]) y = np.uint8([10])

print(cv2.add(x,y)) print(cv2.subtract(x,y))

## 图像混合

这其实也是加法，但是不同的是两幅图像的权重不同，这就会给人一种混合或者透明的感觉。图像混合的计算公式如下：

g (x) = (1 − α) f0 (x) + α f1 (x)

通过修改 α 的值（0 → 1），可以实现非常酷的混合。

现在我们把两幅图混合在一起。第一幅图的权重是 0.7，第二幅图的权重是 0.3。函数 cv2.addWeighted() 可以按下面的公式对图片进行混合操作。

dst = α · img1 + β · img2 + γ

这里 γ 的取值为 0。 python\*\*（day01/test17.py)

import cv2

img1 = cv2.imread('1.jpg') img2 = cv2.imread('9.jpg')

dst = cv2.addWeighted(img1, 0.7, img2, 0.3, 0)

cv2.imshow('dst', dst) cv2.waitKey(0) cv2.destroyAllWindows()



## 按位运算

这里包括的按位操作有：AND，OR，NOT，XOR 等。当我们提取图像的一部分，选择非矩形 ROI 时这些操作会很有用（下一章你就会明白）。下面的例子就是教给我们如何改变一幅图的特定区域。我想把 OpenCV 的标志放到另一幅图像上。如果我使用加法，颜色会改变，如果使用混合，会得到透明效果，但是我不想要透明

python（day01/test18.py)

import cv2

img1 = cv2.imread('1.jpg') img2 = cv2.imread('9.jpg')

rows, cols, channels = img2.shape roi = img1[0:rows, 0:cols]

img2gray = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH\_BINARY) mask\_inv = cv2.bitwise\_not(mask)

# cv2.imshow("mask\_inv",mask\_inv)

img1\_bg = cv2.bitwise\_and(roi, roi, mask=mask\_inv) # cv2.imshow("img1\_bg",img1\_bg)

img2\_fg = cv2.bitwise\_and(img2, img2, mask=mask) # cv2.imshow("img2\_fg",img2\_fg)

dst = cv2.add(img1\_bg, img2\_fg) img1[0:rows, 0:cols] = dst

cv2.imshow('res', img1) cv2.waitKey(0)



# 图像的几何变换

## 知识点

resize/transpose/flip

仿射变换透视变换膨胀/腐蚀

开/闭/梯度/礼帽/黑帽

Resize/transport/flip

python（day01/test04.py)

import cv2

src = cv2.imread('1.jpg') rows, cols, channel = src.shape

dst = cv2.resize(src, (cols \* 2, rows \* 2), interpolation=cv2.INTER\_CUBIC) # dst = cv2.transpose(src)

# dst = cv2.flip(src, 0)

cv2.imshow('src pic', src) cv2.imshow('dst pic', dst) cv2.waitKey(0)

interpolation参数说明

INTER\_NEAREST - 最邻近插值

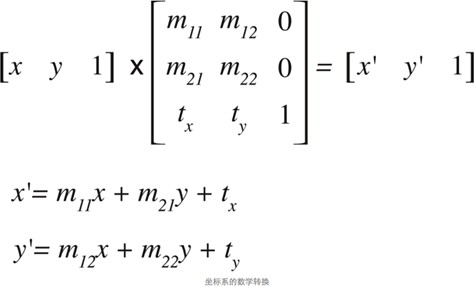
INTER\_LINEAR - 双线性插值，如果最后一个参数你不指定，默认使用这种方法

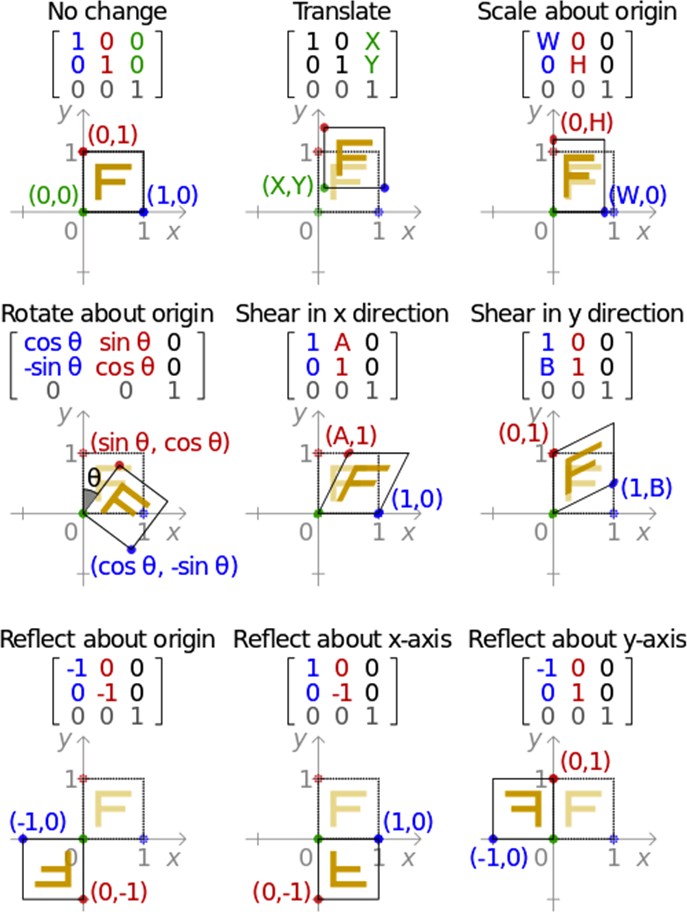
INTER\_AREA -区域插值

INTER\_CUBIC - 4x4像素邻域内的双立方插值 INTER\_LANCZOS4 - 8x8像素邻域内的Lanczos插值

## 仿射变换

任意一个二维图像，我们乘以一个仿射矩阵，就能得到仿射变换后的图像。变换包含：缩放、旋转、平移、倾斜、镜像。





python（day01/test03.py)

import cv2

import numpy as np

src = cv2.imread('1.jpg')

rows, cols, channel = src.shape

M = np.float32([[1, 0, 50], [0, 1, 50]])

# M = np.float32([[0.5, 0, 0], [0, 0.5, 0]])

# M = np.float32([[-0.5, 0, cols // 2], [0, 0.5, 0]])

# M = np.float32([[1, 0.5, 0], [0, 1, 0]])

# M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 0.7) dst = cv2.warpAffine(src, M, (cols, rows))

cv2.imshow('src pic', src) cv2.imshow('dst pic', dst)

cv2.waitKey(0)

## 透视变换

python（day01/test13.py)

import cv2

import numpy as np

img = cv2.imread("4.jpg")

pts1 = np.float32([[25, 30], [179, 25], [12, 188], [189, 190]])

pts2 = np.float32([[0, 0], [200, 0], [0, 200], [200, 200]])

M = cv2.getPerspectiveTransform(pts1, pts2) dst = cv2.warpPerspective(img, M, (200, 201))

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

cv2.waitKey(0)



## 膨胀操作

膨胀操作可以让颜色值大的像素变得更粗膨胀操作前，需要二值化图像

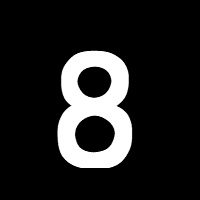
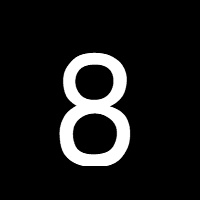
python（day03/test01.py)

import cv2 as cv

img = cv.imread("11.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (5, 5)) dst = cv.dilate(img, kernel)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 腐蚀操作

腐蚀操作可以让颜色值大的像素变得更细腐蚀操作前，需要二值化图像

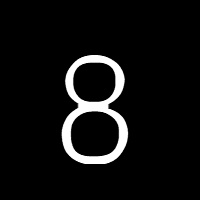
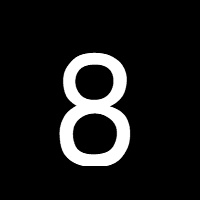
python（day03/test02.py)

import cv2 as cv

img = cv.imread("11.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (5, 5)) dst = cv.erode(img, kernel)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 开操作

开操作是先腐蚀再膨胀开操作可以用于去噪

python（day03/test03.py)

import cv2 as cv

img = cv.imread("10.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3))

dst = cv.morphologyEx(img, cv.MORPH\_OPEN, kernel, iterations=1)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 闭操作

闭操作是先膨胀再腐蚀闭操作可以用于补漏洞

python（day03/test04.py)

import cv2 as cv

img = cv.imread("10.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3))

dst = cv.morphologyEx(img, cv.MORPH\_CLOSE, kernel, iterations=1)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 梯度操作

膨胀减去腐蚀

python（day03/test05.py)

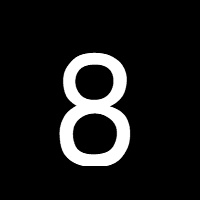
import cv2 as cv

img = cv.imread("11.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3)) dst = cv.morphologyEx(img, cv.MORPH\_GRADIENT, kernel)

cv.imwrite("21.jpg", dst)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 礼帽操作

礼帽操作=开运算图像-原图像获取噪音

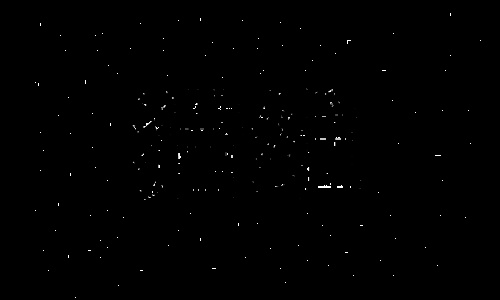
python（day03/test06.py)

import cv2 as cv

img = cv.imread("11.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3)) dst = cv.morphologyEx(img, cv.MORPH\_GRADIENT, kernel)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



## 黑帽操作

黑帽操作=闭运算图像 - 原图像获取漏洞

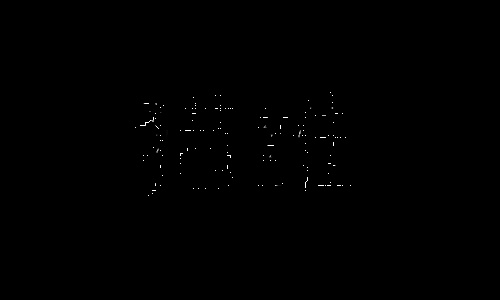
python（day03/test07.py)

import cv2 as cv

img = cv.imread("10.jpg", 0)

kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3)) dst = cv.morphologyEx(img, cv.MORPH\_BLACKHAT, kernel)

cv.imshow('src', img) cv.imshow('dst', dst) cv.waitKey(0)



# 图像滤波

## 知识点

滤波的概念卷积操作 平滑操作 锐化操作 梯度操作

## 滤波概念

滤波过程就是把不需要的信号频率去掉的过程

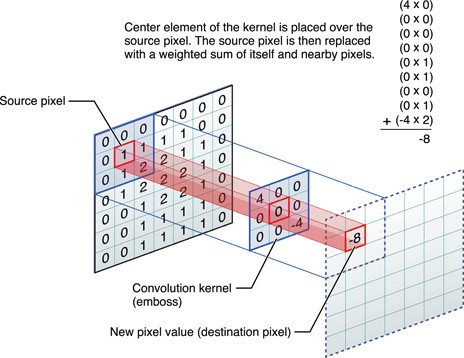
滤波操作一般用卷积操作来实现，卷积核一般称为滤波器滤波分：低通滤波，高通滤波，中通滤波，阻带滤波

低通滤波也叫平滑滤波，可以使图像变模糊，主要用于去噪高通滤波一般用于获取图像边缘、轮廓或梯度

中通滤波一般用于获取已知频率范围内的信号阻带滤波一般用于去掉已知频率范围内的信号滤波分析一般有时域分析和频域分析

时域分析是直接对信号本身进行分析 频域分析师对型号的变化快慢进行分析

## 卷积



python（day02/test00.py)

import cv2

import numpy as np

src = cv2.imread(r"1.jpg")

kernel = np.array([[1, 1, 0], [1, 0, -1], [0, -1, -1]], np.float32) # 定义一个核

dst = cv2.filter2D(src, -1, kernel=kernel)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

## 平滑算子

### 均值滤波

python（day02/test01.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.blur(src, (5,5))

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

### 高斯滤波

python（day02/test02.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.GaussianBlur(src, (5, 5), 0)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

### 中值滤波

python（day02/test03.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.medianBlur(src, 5)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

### 双边滤波

python（day02/test04.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.bilateralFilter(src,9,75,75)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

## 锐化算子

Laplacian**锐化** python（day02/test05.py)

import cv2

src = cv2.imread(r"1.jpg")

kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], np.float32) #定义一个核 dst = cv2.filter2D(src, -1, kernel=kernel)

cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

USM**锐化** python（day02/test06.py)

import cv2

src = cv2.imread(r"1.jpg")

dst = cv2.GaussianBlur(src, (5, 5), 0)

dst = cv2.addWeighted(src, 2, dst, -1, 0)

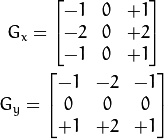
cv2.imshow("src show", src) cv2.imshow("dst show", dst) cv2.waitKey(0)

## 梯度算子

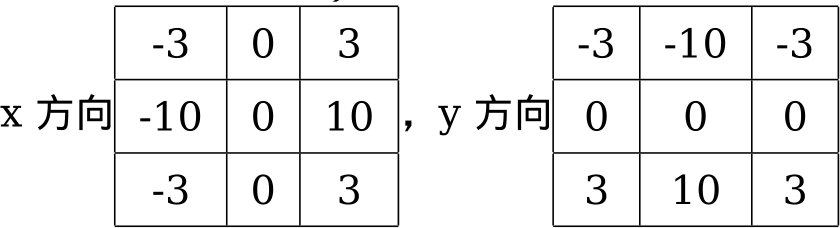
Sobel，Scharr 和 Laplacian梯度滤波器

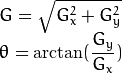
Sobel，Scharr 其实就是求一阶或二阶导数。Scharr 是对 Sobel（使用小的卷积核求解求解梯度角度时）的优化。Laplacian 是求二阶导数

Sobel 算子是高斯平滑与微分操作的结合体，所以它的抗噪声能力很好

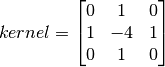
Scharr 滤波器是对Sobel滤波器的改进版本

Soble算子

Scharr算子

计算梯度幅值和方向

为了加快计算，用该公式简化计算 

拉普拉斯算子可以使用二阶导数的形式定义，可假设其离散实现类似于二阶 Sobel 导数，事实上，OpenCV 在计算拉普拉斯算子时直接调用 Sobel 算子

Laplacian 算子

python（canny/test00.py)

import cv2

from matplotlib import pyplot as plt

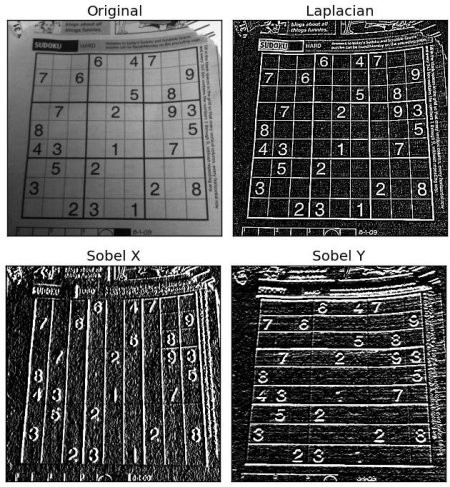
img = cv2.imread('6.jpg', 0)

laplacian = cv2.Laplacian(img, cv2.CV\_64F)

sobelx = cv2.Sobel(img, cv2.CV\_64F, 1, 0, ksize=5) sobely = cv2.Sobel(img, cv2.CV\_64F, 0, 1, ksize=5)

plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray') plt.title('Original'), plt.xticks([]), plt.yticks([]) plt.subplot(2, 2, 2), plt.imshow(laplacian, cmap='gray') plt.title('Laplacian'), plt.xticks([]), plt.yticks([]) plt.subplot(2, 2, 3), plt.imshow(sobelx, cmap='gray') plt.title('Sobel X'), plt.xticks([]), plt.yticks([]) plt.subplot(2, 2, 4), plt.imshow(sobely, cmap='gray') plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()



## 傅里叶变换

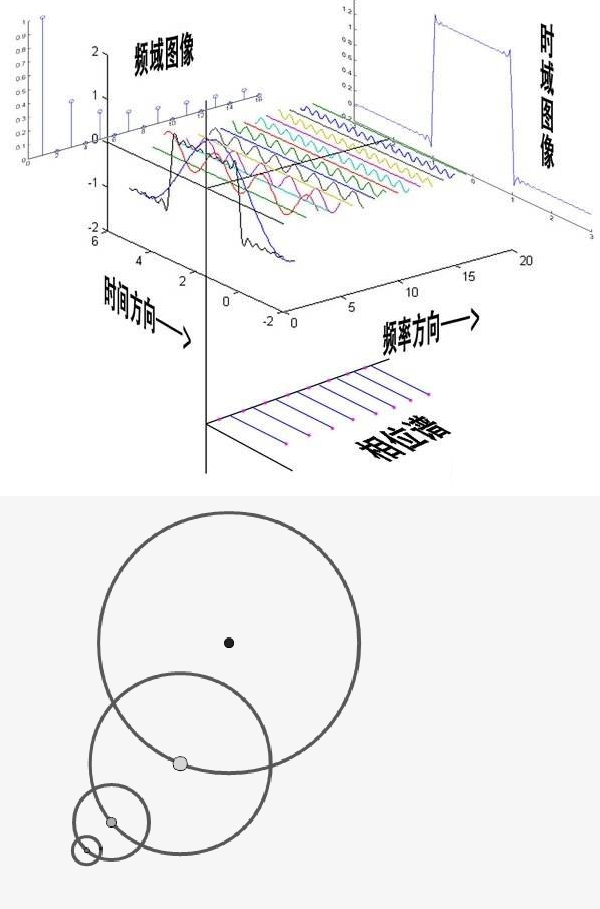
### 目标

理解傅里叶变换原理和作用

使用 OpenCV 对图像进行傅里叶变换

使用 Numpy 中 FFT（快速傅里叶变换）函数

### 原理



Numpy**中傅里叶变换** python

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('1.jpg', 0)

f = np.fft.fft2(img)

fshift = np.fft.fftshift(f)

magnitude\_spectrum = 20 \* np.log(np.abs(fshift))

plt.figure(figsize=(10, 10)) plt.subplot(221), plt.imshow(img, cmap='gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([]) plt.subplot(222), plt.imshow(magnitude\_spectrum, cmap='gray') plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

rows, cols = img.shape

crow, ccol = rows // 2, cols // 2

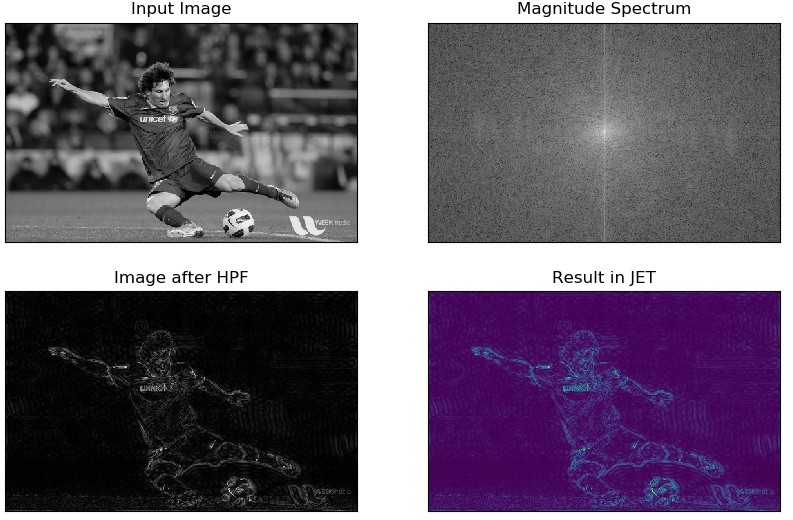
fshift[crow - 30:crow + 30, ccol - 30:ccol + 30] = 0 f\_ishift = np.fft.ifftshift(fshift)

img\_back = np.fft.ifft2(f\_ishift) img\_back = np.abs(img\_back)

plt.subplot(223), plt.imshow(img\_back, cmap='gray') plt.title('Image after HPF'), plt.xticks([]), plt.yticks([]) plt.subplot(224), plt.imshow(img\_back)

plt.title('Result in JET'), plt.xticks([]), plt.yticks([])

plt.show()



OpenCv**傅里叶变换** python

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('1.jpg', 0)

f = np.fft.fft2(img)

fshift = np.fft.fftshift(f)

magnitude\_spectrum = 20 \* np.log(np.abs(fshift))

plt.figure(figsize=(10, 10)) plt.subplot(221), plt.imshow(img, cmap='gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([]) plt.subplot(222), plt.imshow(magnitude\_spectrum, cmap='gray') plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

rows, cols = img.shape

crow, ccol = rows // 2, cols // 2

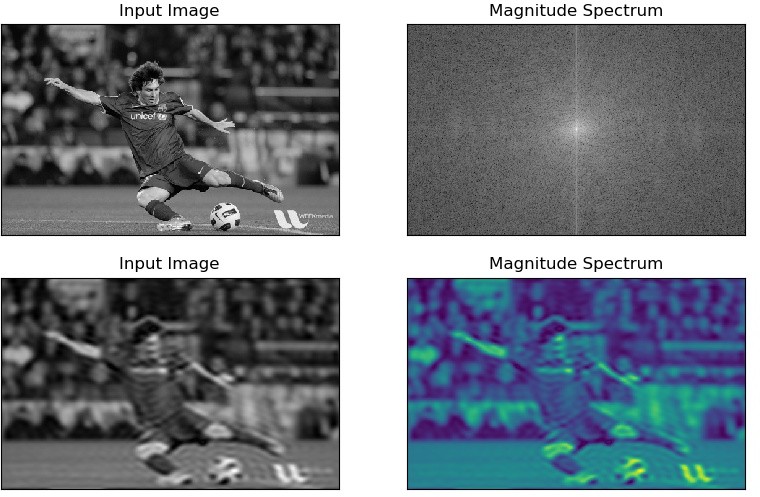
fshift[crow - 30:crow + 30, ccol - 30:ccol + 30] = 0 f\_ishift = np.fft.ifftshift(fshift)

img\_back = np.fft.ifft2(f\_ishift) img\_back = np.abs(img\_back)

plt.subplot(223), plt.imshow(img\_back, cmap='gray') plt.title('Image after HPF'), plt.xticks([]), plt.yticks([]) plt.subplot(224), plt.imshow(img\_back)

plt.title('Result in JET'), plt.xticks([]), plt.yticks([])

plt.show()



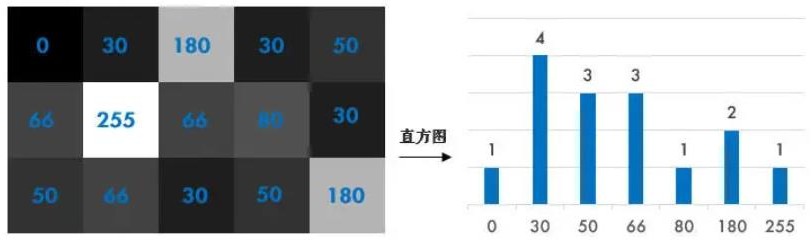
# 直方图

## 知识点

直方图

直方图均衡化

自适应直方图均衡化



## 直方图

python（day04/test01.py)

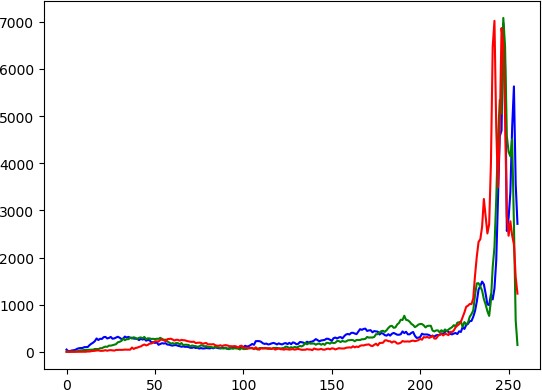
import cv2

import matplotlib.pyplot as plt

plt.plot(img\_R, label='R', color='r')

plt.show()

|  |  |  |  |
| --- | --- | --- | --- |
| img = cv2.imread('1.jpg') |  | | |
| img\_B = cv2.calcHist([img], [0], None, plt.plot(img\_B, label='B', color='b') | [256], | [0, | 256]) |
| img\_G = cv2.calcHist([img], [1], None, plt.plot(img\_G, label='G', color='g') | [256], | [0, | 256]) |
| img\_R = cv2.calcHist([img], [2], None, | [256], | [0, | 256]) |



## 直方图均衡化

python（day01/test02.py)



import cv2

import matplotlib.pyplot as plt

img = cv2.imread('2.jpg', 0) cv2.imshow("src", img)

his = cv2.calcHist([img], [0], None, [256], [0, 256]) plt.plot(his, label='his', color='r')

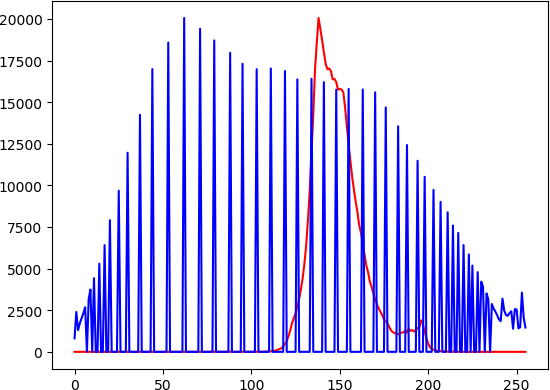
# plt.show()

dst = cv2.equalizeHist(img) cv2.imshow("dst", dst)

cv2.imwrite("15.jpg", dst)

his = cv2.calcHist([dst], [0], None, [256], [0, 256]) plt.plot(his, label='his', color='b')

plt.show()



## 自适应均衡化

自适应直方图均衡化是对图片的每一个局部进行均衡化操作

python（day01/test03.py)

import cv2

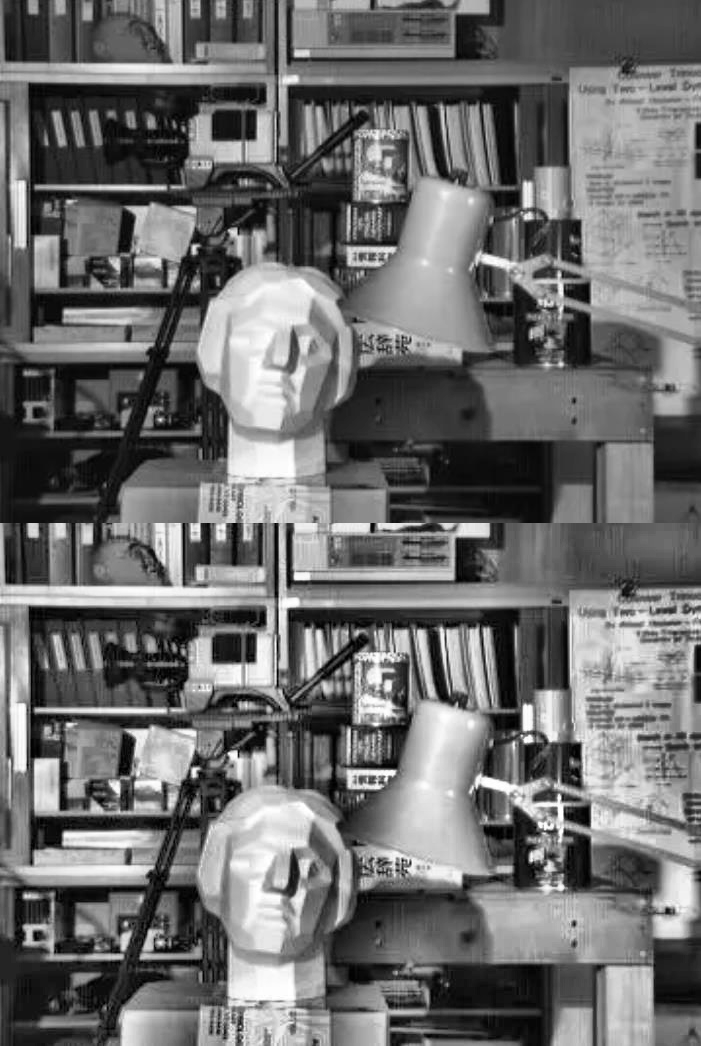
img = cv2.imread('3.jpg', 0) cv2.imshow("src", img)

dst1 = cv2.equalizeHist(img) cv2.imshow("dst1", dst1)

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8)) dst2 = clahe.apply(img)

cv2.imshow("dst2", dst2)

cv2.waitKey(0)



2D **直方图**

在前面的部分我们介绍了如何绘制一维直方图，之所以称为一维，是因为我们只考虑了图像的一个特征：灰度值。但是在 2D 直方图中我们就要考虑两个图像特征。对于彩色图像的直方图通常情况下我们需要考虑每个的颜色（Hue）和饱和度（Saturation）。根据这两个特征绘制 2D 直方图。

python（day01/test03.py)

import cv2

img = cv2.imread('3.jpg')

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

hist = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])

cv2.imshow("hist",hist) cv2.waitKey(0)

# 直方图反向投影

它可以用来做图像分割，或者在图像中找寻我们感兴趣的部分。简单来说，它会输出与输入图像

（待搜索）同样大小的图像，其中的每一个像素值代表了输入图像上对应点属于目标对象的概率。用更简单的话来解释，输出图像中像素值越高（越白）的点就越可能代表我们要搜索的目标（在输入图像所在的位置）。这是一个直观的解释。直方图投影经常与 camshift算法等一起使用。

我们要查找的对象要尽量占满这张图像（换句话说，这张图像上最好是有且仅有我们要查找的对 象）。最好使用颜色直方图，因为一个物体的颜色要比它的灰度能更好的被用来进行图像分割与对象识别。接着我们再把这个颜色直方图投影到输入图像中寻找我们的目标，也就是找到输入图像中的每一个像素点的像素值在直方图中对应的概率，这样我们就得到一个概率图像，最后设置适当的阈值对概率图像进行二值化

python（day01/test03.py)

import cv2

import numpy as np

roi = cv2.imread('7.jpg')

hsv = cv2.cvtColor(roi,cv2.COLOR\_BGR2HSV) target = cv2.imread('6.jpg')

hsvt = cv2.cvtColor(target,cv2.COLOR\_BGR2HSV)

roihist = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )

cv2.normalize(roihist,roihist,0,255,cv2.NORM\_MINMAX)

dst = cv2.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)

disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5)) dst=cv2.filter2D(dst,-1,disc)

ret,thresh = cv2.threshold(dst,50,255,0)

thresh = cv2.merge((thresh,thresh,thresh))

res = cv2.bitwise\_and(target,thresh) res = np.hstack((target,thresh,res)) cv2.imwrite('65.jpg',res)

cv2.imshow('img',res) cv2.waitKey(0)



# 图像金字塔

## 知识点

学习图像金字塔

使用图像创建一个新水果：“橘子苹果”

将要学习的函数有：cv2.pyrUp()，cv2.pyrDown()高斯金字塔和拉普拉斯金字塔

## 高斯金字塔

python（day05/test01.py)



import cv2

img = cv2.imread(r"1.jpg") for i in range(3):

cv2.imshow(f"img{i}",img) img = cv2.pyrDown(img)

cv2.waitKey(0)

python（day05/test02.py)

import cv2

img = cv2.imread(r"1.jpg") for i in range(3):

cv2.imshow(f"img{i}",img) img = cv2.pyrDown(img)

cv2.waitKey(0)



## 拉普拉斯金字塔

拉普拉斯金字塔由高斯金字塔计算得来



python（day05/test03.py)

import cv2

img = cv2.imread(r"1.jpg") img\_down = cv2.pyrDown(img) img\_up = cv2.pyrUp(img\_down)

img\_new = cv2.subtract(img, img\_up) #为了更容易看清楚，做了个提高对比度的操作

img\_new = cv2.convertScaleAbs(img\_new, alpha=5, beta=0) cv2.imshow("img\_LP", img\_new)

cv2.waitKey(0)



# 模板匹配

## 知识点

使用模板匹配在一幅图像中查找目标单对象匹配

多对象匹配

cv2.TM\_CCOEFF、 cv2.TM\_CCOEFF\_NORMED、cv2.TM\_CCORR、cv2.TM\_CCORR\_NORMED、 cv2.TM\_SQDIFF、cv2.TM\_SQDIFF\_NORMED

函数：cv2.matchTemplate()，cv2.minMaxLoc()

注意：如果你使用的比较方法是 cv2.TM\_SQDIFF和cv2.TM\_SQDIFF\_NORMED，最小值对应的位置才是匹配的区域。

### 原理

模板匹配是用来在一副大图中搜寻查找模版图像位置的方法。OpenCV 为我们提供了函数： cv2.matchTemplate()。和 2D 卷积一样，它也是用模板图像在输入图像（大图）上滑动，并在每一个位置对模板图像和与其对应的输入图像的子区域进行比较。OpenCV 提供了几种不同的比较方法（细节请看文档）。返回的结果是一个灰度图像，每一个像素值表示了此区域与模板的匹配程度。

如果输入图像的大小是（WxH），模板的大小是（wxh），输出的结果的大小就是（W- w+1，H-h+1）。当你得到这幅图之后，就可以使用函数cv2.minMaxLoc() 来找到其中的最小值和最大值的位置了。第一个值为矩形左上角的点（位置），（w，h）为 moban 模板矩形的宽和高。这个矩形就是找到的模板区域了。

## 单对象匹配

python（day06/test01.py)

import cv2

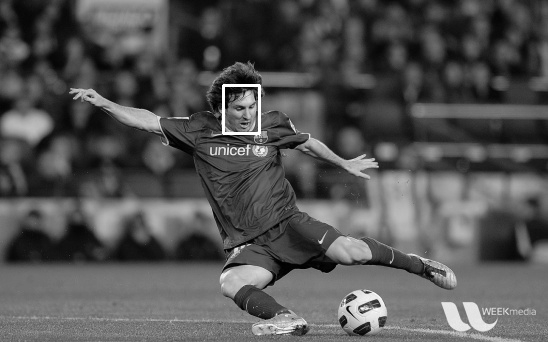
img = cv2.imread('1.jpg', 0)

template = cv2.imread('2.jpg', 0) h, w = template.shape

res = cv2.matchTemplate(img, template, cv2.TM\_CCOEFF) min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(res)

bottom\_right = (max\_loc[0] + w, max\_loc[1] + h) cv2.rectangle(img, max\_loc, bottom\_right, 255, 2)

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



## 多对象匹配

python（day06/test01.py)

import cv2

import numpy as np

img\_rgb = cv2.imread('3.jpg')

img\_gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY) template = cv2.imread('4.jpg', 0)

h, w = template.shape

res = cv2.matchTemplate(img\_gray, template, cv2.TM\_CCOEFF\_NORMED)

loc = np.where(res >= 0.8) for pt in zip(\*loc[::-1]):

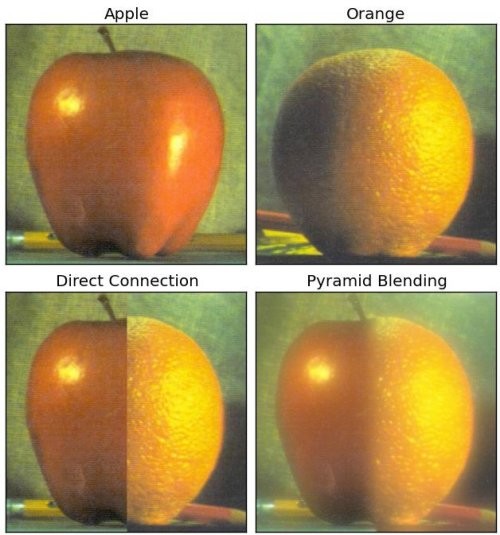
cv2.rectangle(img\_rgb, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)

cv2.imshow('img', img\_rgb) cv2.waitKey(0)



## 无缝融合

我们可以利用金字塔对两种物体进行无缝对接



融合步骤

1. 读入两幅图像，苹果和橘子
2. 构建苹果和橘子的高斯金字塔（6 层）
3. 根据高斯金字塔计算拉普拉斯金字塔
4. 在拉普拉斯的每一层进行图像融合（苹果的左边与橘子的右边融合）
5. 根据融合后的图像金字塔重建原始图像。

python（day05/test04.py)

import cv2

import numpy as np

A = cv2.imread('3.jpg')

B = cv2.imread('4.jpg')

# generate Gaussian pyramid for A G = A.copy()

gpA = [G]

for i in range(6):

G = cv2.pyrDown(G) gpA.append(G)

# generate Gaussian pyramid for B G = B.copy()

gpB = [G]

for i in range(6):

G = cv2.pyrDown(G) gpB.append(G)

# generate Laplacian Pyramid for A lpA = [gpA[5]]

for i in range(5, 0, -1): GE = cv2.pyrUp(gpA[i])

L = cv2.subtract(gpA[i - 1], GE) lpA.append(L)

# generate Laplacian Pyramid for B lpB = [gpB[5]]

for i in range(5, 0, -1): GE = cv2.pyrUp(gpB[i])

L = cv2.subtract(gpB[i - 1], GE) lpB.append(L)

# Now add left and right halves of images in each level LS = []

for la, lb in zip(lpA, lpB): rows, cols, dpt = la.shape

ls = np.hstack((la[:, 0:cols // 2], lb[:, cols // 2:])) LS.append(ls)

# now reconstruct ls\_ = LS[0]

for i in range(1, 6): ls\_ = cv2.pyrUp(ls\_)

ls\_ = cv2.add(ls\_, LS[i])

# image with direct connecting each half

real = np.hstack((A[:, :cols // 2], B[:, cols // 2:]))

cv2.imshow('Pyramid\_blending.jpg', ls\_) cv2.imshow('Direct\_blending.jpg', real)

cv2.waitKey(0)

Canny**边缘提取算法**

## 知识点

Canny算法是一种边缘提取算法学会使用OpenCV调用Canny算法 Canny算法的基本原理

## 代码

python（canny/test01.py)



import cv2

img = cv2.imread("1.jpg", 0)

img = cv2.GaussianBlur(img, (3, 3), 0)

canny = cv2.Canny(img, 50, 150)

cv2.imshow('Canny', canny) cv2.waitKey(0)

## 原理详解

### 步骤

彩色图像转换为灰度图像高斯滤波，滤除噪声点

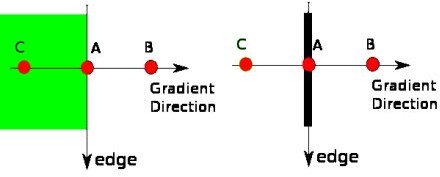
计算图像梯度，根据梯度计算边缘幅值与角度非极大值抑制

双阈值边缘连接处理二值化图像输出结果

### 非极大值抑制

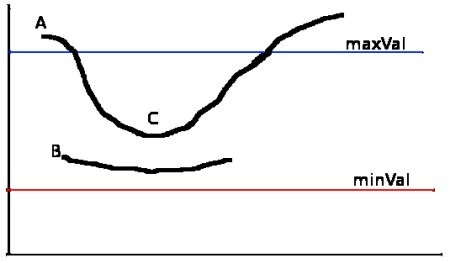
非极大值抑制是一种边缘稀疏技术，非极大值抑制的作用在于“瘦”边。对图像进行梯度计算后，仅仅基于梯度值提取的边缘仍然很模糊。而非极大值抑制则可以帮助将局部最大值之外的所有梯度值抑制为0

1. 将当前像素的梯度强度与沿正负梯度方向上的两个像素进行比较。
2. 如果当前像素的梯度强度与另外两个像素相比最大，则该像素点保留为边缘点，否则该像素点将被抑制。



### 双阈值边缘连接处理

这个阶段决定哪些边缘是边缘，哪些边缘不是边缘。为此，我们需要两个阈值，minVal和maxVal。强度梯度大于maxVal的边缘肯定是边缘，而minVal以下的边缘肯定是非边缘的，因此被丢弃. 两者之间的值要判断是否与真正的边界相连，相连就保留，不相连舍弃。



# 轮廓

## 知识点

理解什么是轮廓查找绘制轮廓 轮廓特征

## 轮廓查找与绘制

python（canny/test05.py)

import cv2

img = cv2.imread('2.jpg')

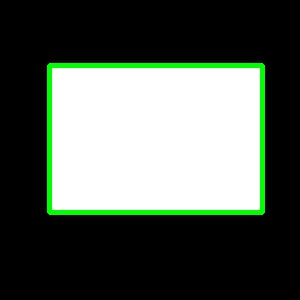
imggray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY) ret,thresh = cv2.threshold(imggray,127,255,0)

image, contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# image, contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

img\_contour= cv2.drawContours(img, contours, -1, (0, 255, 0), 3)

cv2.imshow("img\_contour", img\_contour) cv2.waitKey(0)



## 面积，周长，重心

python（canny/test06.py)

import cv2

img = cv2.imread('3.jpg', 0)

ret, thresh = cv2.threshold(img, 127, 255, 0)

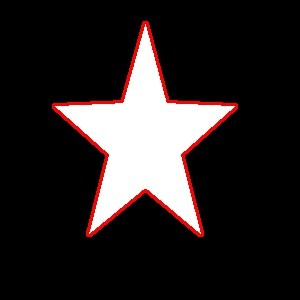
\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

M = cv2.moments(contours[0]) # 矩

cx, cy = int(M['m10'] / M['m00']), int(M['m01'] / M['m00']) print("重心:", cx, cy)

area = cv2.contourArea(contours[0]) print("面积:", area)

perimeter = cv2.arcLength(contours[0], True) print("周长:", perimeter)



## 轮廓近似

将轮廓形状近似到另外一种由更少点组成的轮廓形状，新轮廓的点的数目由我们设定的准确度来决定。

python（canny/test07.py)

import cv2

img = cv2.imread('3.jpg')

imggray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY) ret,thresh = cv2.threshold(imggray,127,255,0)

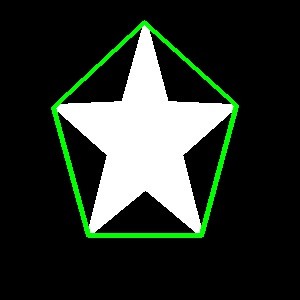
\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

epsilon = 60 #精度

approx = cv2.approxPolyDP(contours[0],epsilon,True)

img\_contour= cv2.drawContours(img, [approx], -1, (0, 255, 0), 3)

cv2.imshow("img\_contour", img\_contour) cv2.waitKey(0)



## 凸包与凸性检测

凸包与轮廓近似相似，但不同，虽然有些情况下它们给出的结果是一样的。

函数 cv2.convexHull() 可以用来检测一个曲线是否具有凸性缺陷，并能纠正缺陷。一般来说，凸性曲线总是凸出来的，至少是平的。如果有地方凹进去了就被叫做凸性缺陷。

函数 cv2.isContourConvex() 可以可以用来检测一个曲线是不是凸的。它只能返回 True 或

False。

python（canny/test08.py)

import cv2

img = cv2.imread('3.jpg')

imggray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY) ret,thresh = cv2.threshold(imggray,127,255,0)

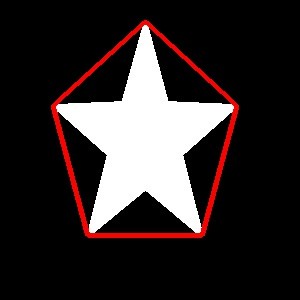
\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

hull = cv2.convexHull(contours[0]) print(cv2.isContourConvex(contours[0]), cv2.isContourConvex(hull)) #False True

#说明轮廓曲线是非凸的，凸包曲线是凸的

img\_contour= cv2.drawContours(img, [hull], -1, (0, 0, 255), 3)

cv2.imshow("img\_contour", img\_contour) cv2.waitKey(0)



## 边界检测

边界矩形 最小矩形 最小外切圆

python（canny/test9.py)

import cv2

import numpy as np

img = cv2.imread('4.jpg')

imggray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) ret, thresh = cv2.threshold(imggray, 127, 255, 0)

\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# 边界矩形

x, y, w, h = cv2.boundingRect(contours[0])

img\_contour = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

# 最小矩形

rect = cv2.minAreaRect(contours[0]) box = cv2.boxPoints(rect)

box = np.int0(box)

img\_contour = cv2.drawContours(img, [box], 0, (0, 0, 255), 2)

# 最小外切圆

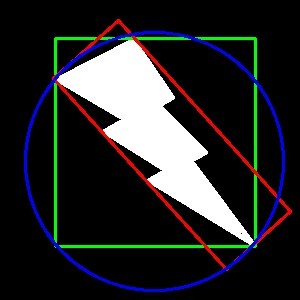
(x, y), radius = cv2.minEnclosingCircle(contours[0]) center = (int(x), int(y))

radius = int(radius)

img\_contour = cv2.circle(img, center, radius, (255, 0, 0), 2)

cv2.imshow("img\_contour", img\_contour)

cv2.waitKey(0)



python（canny/test10.py)

import cv2

import numpy as np

img = cv2.imread('4.jpg')

imggray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) ret, thresh = cv2.threshold(imggray, 127, 255, 0)

\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# 椭圆拟合

ellipse = cv2.fitEllipse(contours[0]) cv2.ellipse(img, ellipse, (255, 0, 0), 2)

# 直线拟合

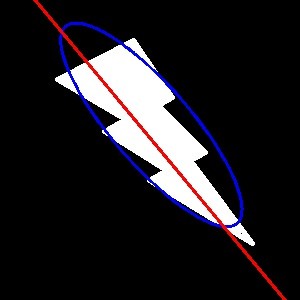
h, w, \_ = img.shape

[vx, vy, x, y] = cv2.fitLine(contours[0], cv2.DIST\_L2, 0, 0.01, 0.01) lefty = int((-x \* vy / vx) + y)

righty = int(((w - x) \* vy / vx) + y)

cv2.line(img, (w - 1, righty), (0, lefty), (0, 0, 255), 2)

cv2.imshow("img\_contour", img) cv2.waitKey(0)



## 轮廓性质

**边界矩形的宽高比**

x,y,w,h = cv2.boundingRect(cnt) aspect\_ratio = float(w)/h

**轮廓面积与边界矩形面积的比**

area = cv2.contourArea(cnt) x,y,w,h = cv2.boundingRect(cnt) rect\_area = w\*h

extent = float(area)/rect\_area

**轮廓面积与凸包面积的比**

area = cv2.contourArea(cnt) hull = cv2.convexHull(cnt)

hull\_area = cv2.contourArea(hull) solidity = float(area)/hull\_area

**与轮廓面积相等的圆形的直径**

area = cv2.contourArea(cnt) equi\_diameter = np.sqrt(4\*area/np.pi)

**对象的方向**

下面的方法还会返回长轴和短轴的长度

(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)

## 对象掩码

用于获取构成对象的所有像素点

python（canny/test11.py)

import cv2

import numpy as np

img = cv2.imread('3.jpg')

imggray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) ret, thresh = cv2.threshold(imggray, 127, 255, 0)

\_, contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

mask = np.zeros(imggray.shape, np.uint8) cv2.drawContours(mask, [contours[0]], 0, 255, -1)

pixelpoints = np.transpose(np.nonzero(mask)) print(pixelpoints)

cv2.imshow("mask", mask) cv2.waitKey(0)

**最大值和最小值及它们的位置**

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(imgray,mask = mask)

**使用相同的掩模求一个对象的平均颜色或平均灰度**

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(imgray,mask = mask)

## 形状匹配

python（canny/test12.py)

import cv2

import numpy as np

img1 = cv2.imread('4.jpg', 0) img2 = cv2.imread('5.jpg', 0)

ret, thresh = cv2.threshold(img1, 127, 255, 0)

\_, contours, hierarchy = cv2.findContours(thresh, 2, 1) cnt1 = contours[0]

ret, thresh2 = cv2.threshold(img2, 127, 255, 0)

\_, contours, hierarchy = cv2.findContours(thresh2, 2, 1) cnt2 = contours[0]

ret = cv2.matchShapes(cnt1, cnt2, 1, 0.0) print(ret)

Hough**变换**

## 知识点

在图像处理中，霍夫变换用来检测任意能够用数学公式表达的形状，即使这个形状被破坏或者有点扭曲

使用Hough变换进行直线检测使用Hough变换进行圆检测 了解Hough变换的原理

## 直线检测

python（hough/test01.py)

import cv2

import numpy as np

image = cv2.imread("1.jpg")

image = cv2.GaussianBlur(image, (5, 5), 50)

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) cv2.imshow("image\_gray", image\_gray)

edges = cv2.Canny(image\_gray, 100, 150) cv2.imshow("image\_edges", edges)

lines = cv2.HoughLines(edges, 1, np.pi / 180, 100) for line in lines:

rho, theta = line[0]

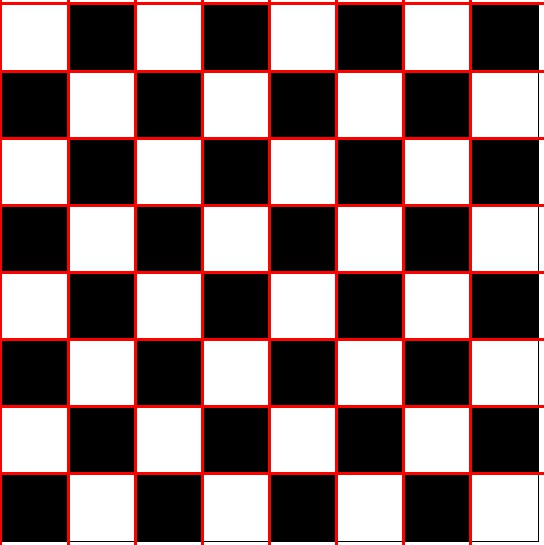
a = np.cos(theta)

b = np.sin(theta)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x0 | = | a \* rho |  | | | |
| y0 | = | b \* rho |
| x1 | = | int(x0 | + | 1000 | \* (-b)) | # 直线起点横坐标 |
| y1 | = | int(y0 | + | 1000 | \* (a)) | # 直线起点纵坐标 |
| x2 | = | int(x0 | - | 1000 | \* (-b)) | # 直线终点横坐标 |
| y2 | = | int(y0 | - | 1000 | \* (a)) | # 直线终点纵坐标 |

cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2) cv2.imshow("image\_lines", image)

cv2.waitKey(0)



## 圆检测

python（hough/test02.py)



import cv2

import numpy as np

image = cv2.imread("2.jpg")

dst = cv2.cvtColor(image, cv2.COLOR\_BGRA2GRAY)

circle = cv2.HoughCircles(dst, cv2.HOUGH\_GRADIENT, 1, 30, param1=40, param2=20, minRadius=20, maxRadius=300)

if not circle is None:

circle = np.uint16(np.around(circle)) for i in circle[0, :]:

cv2.circle(image, (i[0], i[1]), i[2], (0, 255, 0), 2)

cv2.imshow("circle", image) cv2.waitKey(0)

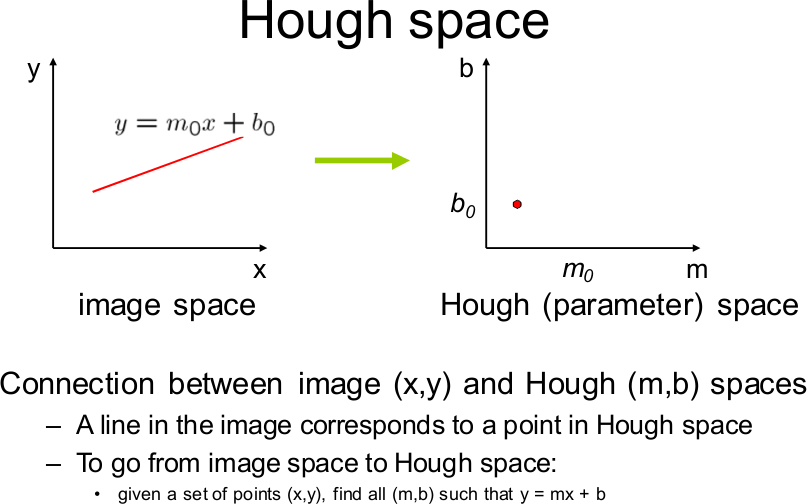
## 原理详解

### 步骤

轮廓检测算法检测出轮廓

投射到hough空间进行形状检测

**数学原理**



# 图像分割

## 知识点

使用分水岭算法基于掩模的图像分割函数：cv2.watershed()

### 原理

任何一副灰度图像都可以被看成拓扑平面，灰度值高的区域可以被看成是山峰，灰度值低的区域可以被看成是山谷。我们向每一个山谷中灌不同颜色的水。随着水的位的升高，不同山谷的水就会相遇汇合，为了防止不同山谷的水汇合，我们需要在水汇合的地方构建起堤坝。不停的灌水，不停的构建堤坝知道所有的山峰都被水淹没。我们构建好的堤坝就是对图像的分割。这就是分水岭算法的背后哲理。

但是这种方法通常都会得到过度分割的结果，这是由噪声或者图像中其他不规律的因素造成 的。为了减少这种影响，OpenCV 采用了基于掩模的分水岭算法，在这种算法中我们要设置那些 山谷点会汇合，那些不会。这是一种交互式的图像分割。我们要做的就是给我们已知的对象打上不同的标签。如果某个区域肯定是前景或对象，就使用某个颜色（或灰度值）标签标记它。如果某个区域肯定不是对象而是背景就使用另外一个颜色标签标记。而剩下的不能确定是前景还是背景的区域就用 0 标记。这就是我们的标签。然后实施分水岭算法。每一次灌水，我们的标签就会被更

新，当两个不同颜色的标签相遇时就构建堤坝，直到将所有山峰淹没，最后我们得到的边界对象

（堤坝）的值为 -1。

**代码**

import numpy as np import cv2

from matplotlib import pyplot as plt

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

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

ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU)

# noise removal

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

opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)

# sure background area

sure\_bg = cv2.dilate(opening, kernel, iterations=3)

dist\_transform = cv2.distanceTransform(opening, 1, 5)

ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0) # Finding unknown region

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg, sure\_fg)

# Marker labelling

ret, markers1 = cv2.connectedComponents(sure\_fg)

# Add one to all labels so that sure background is not 0, but 1 markers = markers1 + 1

# Now, mark the region of unknown with zero markers[unknown == 255] = 0

markers3 = cv2.watershed(img, markers) img[markers3 == -1] = [255, 0, 0]

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



# 项目1：简单车牌提取

## 需求

提取下面图片的车牌区域



## 步骤

1. 高斯模糊
2. 图片灰度化
3. Sobel算子
4. 图像二值化
5. 闭操作
6. 膨胀腐蚀
7. 中值滤波
8. 查找轮廓
9. 判断车牌区域

## 源码

python（canny/test13.py)

import cv2

# 读取图片

rawImage = cv2.imread("2.jpg")

# 高斯模糊，将图片平滑化，去掉干扰的噪声

image = cv2.GaussianBlur(rawImage, (3, 3), 0) # 图片灰度化

image = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY) # Sobel算子（X方向）

Sobel\_x = cv2.Sobel(image, cv2.CV\_16S, 1, 0)

# Sobel\_y = cv2.Sobel(image, cv2.CV\_16S, 0, 1) absX = cv2.convertScaleAbs(Sobel\_x) # 转回uint8 # absY = cv2.convertScaleAbs(Sobel\_y)

# dst = cv2.addWeighted(absX, 0.5, absY, 0.5, 0) image = absX

# 二值化：图像的二值化，就是将图像上的像素点的灰度值设置为0或255,图像呈现出明显的只有黑和白

ret, image = cv2.threshold(image, 0, 255, cv2.THRESH\_OTSU) # 闭操作：闭操作可以将目标区域连成一个整体，便于后续轮廓的提取。

kernelX = cv2.getStructuringElement(cv2.MORPH\_RECT, (17, 5)) image = cv2.morphologyEx(image, cv2.MORPH\_CLOSE, kernelX)

# 膨胀腐蚀(形态学处理)

kernelX = cv2.getStructuringElement(cv2.MORPH\_RECT, (20, 1))

kernelY = cv2.getStructuringElement(cv2.MORPH\_RECT, (1, 19)) image = cv2.dilate(image, kernelX)

image = cv2.erode(image, kernelX) image = cv2.erode(image, kernelY)

image = cv2.dilate(image, kernelY)

# 平滑处理，中值滤波

image = cv2.medianBlur(image, 15) # 查找轮廓

tmp, contours, w1 = cv2.findContours(image, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

for item in contours:

rect = cv2.boundingRect(item) x = rect[0]

y = rect[1] weight = rect[2] height = rect[3]

if weight > (height \* 2): # 裁剪区域图片

chepai = rawImage[y:y + height, x:x + weight] cv2.imshow('chepai'+str(x), chepai)

# 绘制轮廓

image = cv2.drawContours(rawImage, contours, -1, (0, 0, 255), 3) cv2.imshow('image', image)

cv2.waitKey(0) cv2.destroyAllWindows()

