# 轮廓

## 1.轮廓入门

## 目标：

* 了解什么是轮廓
* 学习查找轮廓，绘制轮廓
* 函数: cv.findContours(), cv.drawContours()

## 什么是轮廓？

轮廓可以简单地解释为连接所有具有相同的颜色或强度的连续点（沿着边界）的曲线。轮廓是形状分析和物体检测和识别的很有用的工具。

* 为了更好的准确性，使用二进制图像，因此，在找到轮廓之前，应用阈值或canny边缘检测。
* 从OpenCV 3.2开始，findContours()不再修改源图像，而是将修改后的图像作为三个返回参数中的第一个返回。
* 在OpenCV中，找到轮廓就像从黑色背景中找到白色物体。所以请记住，要找到的对象应该是白色，背景应该是黑色。
* 在cv.findContours()函数中有三个参数，第一个是源图像，第二个是轮廓检索模式，第三个是轮廓逼近方法。它输出轮廓和层次结构。contours是图像中所有轮廓的Python列表，每个单独的轮廓是对象边界点坐标(x,y)的Numpy数组。
* 注意：我们稍后将详细讨论第二和第三个参数以及层次结构。在此之前，代码示例中给出的值对所有图像都可以正常工作。

## 如何绘制轮廓？

* 要绘制轮廓，可以使用cv.drawContours函数。如果图像有边界点，它也可以用于绘制任何形状。它的第一个参数是源图像，第二个参数是应该作为Python列表传递的轮廓，第三个参数是轮廓索引（在绘制单个轮廓时很有用。绘制所有轮廓，传递-1），其余参数是颜色，厚度等等
* 要绘制图像中的所有轮廓：
* cv.drawContours(img, contours, -1, (0,255,0), 3)
* 要绘制单个轮廓，请输入四个轮廓点：
* cv.drawContours(img, contours, 3, (0,255,0), 3)
* 但大多数时候，下面的方法会很有用：
* cnt = contours[4]
* cv.drawContours(img, [cnt], 0, (0,255,0), 3)
* 注意：最后两种方法是相同的，但是当你继续前进时，你会发现最后一种方法更有用。

## 简单示例

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

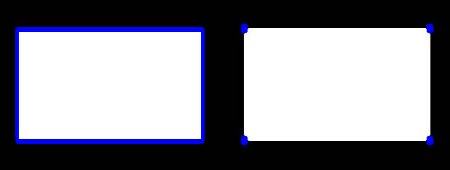
## 轮廓近似方法

这是cv.findContours函数中的第三个参数。它实际上表示什么？

在上面，我们告诉轮廓是具有相同强度的形状的边界。它存储形状边界的（x，y）坐标。但是它存储了所有坐标吗？这由该轮廓近似方法指定。

如果传递cv.CHAIN\_APPROX\_NONE，则存储所有边界点。但实际上我们需要所有的积分吗？例如，你找到了直线的轮廓，你是否需要线上的所有点来表示该线？不，我们只需要该线的两个端点。这就是cv.CHAIN\_APPROX\_SIMPLE的作用。它删除所有冗余点并压缩轮廓，从而节省内存。

下面的矩形图像展示了这种技术。只需在轮廓阵列中的所有坐标上绘制一个圆圈（以蓝色绘制）。第一张图片显示了我用cv.CHAIN\_APPROX\_NONE（734点）获得的点数，第二张图片显示了一张带有cv.CHAIN\_APPROX\_SIMPLE（仅4点）的点数，它节省了不少内存！

[](https://camo.githubusercontent.com/c09c87e170356bd52d4cd9442e2d99b6686f7cf87d52af9e0c5791dd8013e5e4/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f6e6f6e652e6a7067)

## 2.轮廓特征

## 目标：

本章节你需要学习以下内容:

* 查找轮廓的不同特征，如面积，周长，质心，边界框等

### 1. 矩

图像的矩可帮助你计算某些特征，如对象的质心，对象的面积等特征。

|  |
| --- |
|  |

### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

#### 从这一刻起，你可以提取有用的数据，如面积，质心等

|  |
| --- |
|  |

### 2. 轮廓面积

轮廓区域由函数cv.contourArea()或时刻M['m00']给出。

area = cv.contourArea(cnt)

还是上面的例子，我们来添加计算轮廓面积代码

|  |
| --- |
|  |

#### 结果：

|  |
| --- |
|  |

### 3. 轮廓周长

轮廓周长也被称为弧长。可以使用cv.arcLength()函数找到它。第二个参数指定形状是闭合轮廓（如果传递为True），还是仅仅是曲线。

perimeter = cv.arcLength(cnt,True)

### 还是上面的示例，添加角色轮廓周长的代码

|  |
| --- |
|  |

#### 结果：

|  |
| --- |
|  |

### 4. 轮廓近似

它根据我们指定的精度将轮廓形状近似为具有较少顶点数的另一个形状。它是Douglas-Peucker算法的一种实现方式。 要理解这一点，可以假设你试图在图像中找到一个正方形，但是由于图像中的一些问题，你没有得到一个完美的正方形，而是一个“坏形状”（如下图第一张图所示）。现在你可以使用此功能来近似形状。在这里，第二个参数称为epsilon，它是从轮廓到近似轮廓的最大距离。这是一个准确度参数。需要选择适当的epsilon才能获得正确的输出。

epsilon = 0.1\*cv.arcLength(cnt,True)

approx = cv.approxPolyDP(cnt,epsilon,True)

### 示例，还是上面的例子，我们需要使用循环，

|  |
| --- |
|  |

### 注意避坑：

### 1.轮廓近似值需要使用循环来做，

### 2.drawContours函数的第二个参数是一个数组，即使你只有一个元素，也需要用数组的形式传递进去，否则不能正确绘制

### 5.凸包

凸包(Convex Hull)是一个计算几何（图形学）中的概念，它的严格的数学定义为：在一个向量空间V中，对于给定集合X，所有包含X的凸集的交集S被称为X的凸包。  
  在图像处理过程中，我们常常需要寻找图像中包围某个物体的凸包。凸包跟多边形逼近很像，只不过它是包围物体最外层的一个凸集，这个凸集是所有能包围这个物体的凸集的交集。如下图所示：

在上图中，绿色线条所包围的凸集即为白色图形的凸包。  
  在opencv中，通过函数convexHulll能很容易的得到一系列点的凸包，比如由点组成的轮廓，通过convexHull函数，我们就能得到轮廓的凸包。寻找图像的凸包，能够让我们做一些有意思的事情，比如手势识别等。

凸包看起来类似于轮廓近似，但它不是（两者在某些情况下可能提供相同的结果）。这里，cv.convexHull()函数检查曲线的凸性缺陷并进行修正。一般而言，凸曲线是总是凸出或至少平坦的曲线。如果它在内部膨胀，则称为凸性缺陷。例如，检查下面的手形图像。红线表示手的凸包。双面箭头标记显示凸起缺陷，即船体与轮廓的局部最大偏差。

[](https://camo.githubusercontent.com/df66ebe7cac0870ef929b0e7086a5319a7d3b7aef348c8fd074dbd87a4a55067/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f636f6e766578697479646566656374732e6a7067)

下面我们要讨论它的一些语法：

hull = cv.convexHull(points[, hull[, clockwise[, returnPoints]]

参数详情：

* points：是我们传入的轮廓。
* hull：是输出，通常我们忽略它。
* clocwise：方向标志。如果为True，则输出凸包顺时针方向。否则，它逆时针方向。
* returnPoints：默认为True。然后它返回凸包点的坐标。如果为False，则返回与凸包点对应的轮廓点的索引。

因此，为了获得如上图所示的凸包，以下就足够了：

hull = cv.convexHull(cnt)

但是如果你想找到凸性缺陷，你需要传递returnPoints = False。为了理解它，我们将采用上面的矩形图像。首先，我发现它的轮廓为cnt。现在我发现它的凸包有returnPoints = True，我得到以下值：[[234 202],[51 202],[51 79],[234 79]]这四个角落 矩形点。 现在如果对returnPoints = False做同样的事情，我得到以下结果：[[129],[67],[0],[142]]。 这些是轮廓中相应点的索引。例如，检查第一个值：cnt [129] = [[234,202]]，它与第一个结果相同（对于其他结果，依此类推）。

### 实例

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 注意：在做二值化的时候，需要根据图片的明暗度来调整阈值，调到合适为止，否则效果不好

### 6.  检查凸性

#### 1、凸包检测与凸缺陷定义

**凸包是将最外层的点连接起来构成的凸多边形，它能包含点击中所有的点**。物体的凸包检测常应用在物体识别、手势识别及边界检测等领域。  
**凸包检测常常用在轮廓分析之后**。对二值图像进行轮廓分析之后，可以构建每个轮廓的凸包，构建完成之后会返回该凸包包含的点集。根据返回的凸包点集可以绘制该轮廓对应的凸包。**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**。下图可以更加直接的了解凸包和凸缺陷的定义：  


#### 2、opencv相关实现函数

函数cv.isContourConvex()可以检查曲线是否凸的，它只返回True或False，没有什么理解上的问题。

**k = cv.isContourConvex(cnt)**

OpenCV中提供了**函数convexHull** 用于对物体轮廓凸包进行检测，**函数convexityDefect**进行凸缺陷的检测。首先来了解一下各参数意义：

**hull = cv2.convexHull(points, clockwise, returnpoints)**

**各参数意义**：  
**points**：输入的轮廓点集  
**clockwise**：方向标志，如果设置为True，输出的凸包是顺时针方向的，否则为逆时针方向。  
**returnPoints**：是否返回点集  
**hull** : 输出凸包结果，当参数returnPoints为ture的时候返回凸包的顶点坐标是个点集、returnPoints为false的是返回与凸包点对应的轮廓上的点对应的index

下面这个函数可以帮助我们找到凸缺陷：

**convexityDefects = cv2.convexityDefect(contour， convexhull)**

**注意：进行凸检测时，凸包检测中的returnPoints要设置为False**  
**凸缺陷返回一个数组，每一行包含值是起点，终点，最远的点，到最远点的近似距离，返回的前三个点都是轮廓索引**。

**contour**： 检测到的轮廓，可以调用findContours函数得到  
**convexhull**：检测到的凸包，可以调用convexHull函数得到。  
**convexityDefects**：输出参数，检测到的最终结果，返回一个数组，其中每一行包含的值是[起点，终点，最远的点，到最远点的近似距离]。**前三个点都是轮廓索引**。  
**前三个值得含义分别为：凸缺陷的起始点，凸缺陷的终点，凸缺陷的最深点（即边缘点到凸包距离最大点）**

#### 3、代码实践

我们可以在一张图上显示轮廓的凸包和凸缺陷。  
我们将起点和终点用一条红线 连接，在最远点画一个蓝色的圆圈，**要记住的是返回结果的前三个值是轮廓点的索引。 所以我们还要到轮廓点中去找它们**。**前三个值得含义分别为：凸缺陷的起始点，凸缺陷的终点，凸缺陷的最深点（即边缘点到凸包距离最大点）**。

#### 实例

|  |  |
| --- | --- |
|  | *# 凸包检测和凸缺陷* **import** cv2 **as** cv  *# 读取图像* src1 = cv.imread(**"../mydata/hand2.jpg"**) *# 转换为灰度图像* gray = cv.cvtColor(src1, cv.COLOR\_BGR2GRAY) *# 二值化 # ret, binary = cv.threshold(gray, 0, 255, cv.THRESH\_BINARY | cv.THRESH\_OTSU)* ret, binary = cv.threshold(gray, 70, 255, cv.THRESH\_BINARY ) *# 获取结构元素* k = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3)) *# 开操作* binary = cv.morphologyEx(binary, cv.MORPH\_OPEN, k) *# 轮廓发现* \_,contours, hierarchy = cv.findContours(binary, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE) *# 在原图上绘制轮廓，以方便和凸包对比，发现凸缺陷* cv.drawContours(src1, contours, -1, (0, 225, 0), 3) **for** c **in** range(len(contours)):  *# 是否为凸包* ret = cv.isContourConvex(contours[c])  *# 凸缺陷  # 凸包检测，returnPoints为false的是返回与凸包点对应的轮廓上的点对应的index* hull = cv.convexHull(contours[c], returnPoints=**False**)  defects = cv.convexityDefects(contours[c], hull)  print(**'defects'**, defects)  **for** j **in** range(defects.shape[0]):  s, e, f, d = defects[j, 0]  start = tuple(contours[c][s][0])  end = tuple(contours[c][e][0])  far = tuple(contours[c][f][0])  *# 用红色连接凸缺陷的起始点和终止点* cv.line(src1, start, end, (0, 0, 225), 2)  *# 用蓝色最远点画一个圆圈* cv.circle(src1, far, 5, (225, 0, 0), -1)  *# 显示* cv.imshow(**"result"**, src1) cv.waitKey(0) cv.destroyAllWindows() |

#### 效果：红色为凸包，蓝色点为凸缺陷的最深点（即边缘点到凸包距离最大点），绿色是轮廓。红色与绿色之间的区域即为凸缺陷

|  |
| --- |
|  |
| defects [[[ 453 455 454 162]]  [[ 455 457 456 114]]  [[ 457 590 525 34213]]  [[ 590 594 591 114]]  [[ 594 675 642 40129]]  [[ 675 677 676 186]]  [[ 0 2 1 186]]  [[ 2 6 3 114]]  [[ 6 91 35 33500]]  [[ 92 208 141 26172]]  [[ 208 210 209 114]]  [[ 210 212 211 162]]  [[ 213 215 214 217]]  [[ 215 263 260 2079]]  [[ 263 267 264 201]]  [[ 267 281 276 290]]  [[ 281 283 282 186]]  [[ 283 287 286 211]]  [[ 287 295 288 162]]  [[ 295 309 302 253]]  [[ 309 359 352 1949]]  [[ 360 450 373 6163]]  [[ 450 452 451 162]]] |

#### 凸缺陷的部分输出：前三个为凸缺陷的起始点，凸缺陷的终点，凸缺陷的最深点（即边缘点到凸包距离最大点）。凸包检测输出的也是轮廓上的点的索引。

#### 4、总结

（**1**）cv2.convexityDefect函数进行凸缺陷检测时，**凸包检测函数中cv2.convexHull中的returnPoints要设置为False**，返回轮廓中与凸包点对应的轮廓点的index。  
（**2**）cv2.convexityDefect函数的返回值是一系列数组，**[凸缺陷的起点，凸缺陷的终点，凸缺陷的最深的点，到最远点的近似距离]**，前三个是索引值，**终点和起点连接即得到凸包**。  
（**3**）**凸缺陷的最深的点：即边缘点到凸包距离最大点  
到最远点的近似距离：边缘点到凸包的最大距离**。  
（**4**）利用凸包检测可以得到进一步的轮廓信息，**可进行边界检测**，结合凸缺陷**可以实现手势识别和物体识别**。

### 7. 边界矩形

有两种类型的边界矩形。

#### 7.a.直边矩形

它是一个直的矩形，它不考虑对象的旋转。因此，边界矩形的面积不是最小的。它由函数cv.boundingRect()找到。

设(x，y)为矩形的左上角坐标，(w，h)为宽度和高度。

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

**cv.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)**

#### 直边界矩形实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

#### 7.b.旋转矩形

这里，以最小面积绘制边界矩形，因此它也考虑旋转。使用的函数是cv.minAreaRect()。它返回一个Box2D结构，其中包含以下detals - (center(x，y)，(width，height)，rotation of rotation)。但要画这个矩形，我们需要矩形的4个角。它是由函数cv.boxPoints()获得的

**rect = cv.minAreaRect(cnt)**

**box = cv.boxPoints(rect)**

**box = np.int0(box)**

**cv.drawContours(img,[box],0,(0,0,255),2)**

#### 旋转矩形实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

### 8. 最小外接圈

接下来，我们使用函数cv.minEnclosingCircle（）找到对象的外接圆。它是一个完全覆盖物体的圆圈，面积最小。

(x,y),radius = cv.minEnclosingCircle(cnt)

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

radius = int(radius)

cv.circle(img,center,radius,(0,255,0),2)

### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

### 9. 椭圆拟合

接下来是将椭圆拟合到一个对象上。它返回刻有椭圆的旋转矩形。

**ellipse = cv.fitEllipse(cnt)**

**cv.ellipse(img,ellipse,(0,255,0),2)**

### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

### 10.轮廓的拟合直线

**cv2.fitLine()**函数用于返回**轮廓的拟合直线**，其基本格式如下：

[复制代码](javascript:void(0);)

line = cv2.fitLine(contours, distType, param, reps, aeps)

line为返回的拟合直线

contours为用于计算拟合直线的轮廓

distType为距离参数类型, 决定如何计算拟合直线

param为距离参数, 与距离参数类型有关, 其设置为0时, 函数将自动选择最优值

reps为计算拟合直线需要的径向精度, 通常设置为0.01

aeps为计算拟合直线需要的轴向精度, 通常设置为0.01

[复制代码](javascript:void(0);)

param距离参数类型：



### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

### 11、轮廓的最小外包三角形

**cv2.minEnclosingTriangle()**函数用于返回**可容纳轮廓的最小外包三角形**，其基本格式如下：

retval, triangle = cv2.minEnclosingTriangle(contours)

retval为最小外包三角形的面积  
triangle为最小外包三角形  
contours为用于计算最小外包三角形的轮廓

### 实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

## 3.轮廓属性

### 1.长宽比

它是对象边界矩形的宽度与高度的比，我们在之前的教程中已经介绍了获取图像轮廓的x，y，w，h，它们由函数：

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

得到，当然，我们在这里谈论的是图像的边界矩形，并非是最小边界矩形。

现在来计算长宽比，公式：

image.png

### 实例

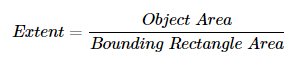
|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |
|  |

### 2.大小比

它是轮廓区域与边界矩形区域的比，公式：



### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |
|  |

### 3.密实比

密实比是轮廓区域与其凸包区域的比，也叫做稳固性

image.png

### 实例代码：

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np """ solidity，密实比或者叫做稳固性，计算方法轮廓面积/凸包面积 """  img = cv2.imread('../../mydata/tubao.jpeg') *# 填写图片名称* cv2.imshow("original", img) gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY) *# 二值化，取阈值为235* ret,th = cv2.threshold(gray,235,255,cv2.THRESH\_BINARY) *# 查找轮廓* \_,contours,hierary = cv2.findContours(th,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE) cnt = contours[0] *# 计算轮廓面积* area = cv2.contourArea(cnt) *# 计算凸包的面积* hull = cv2.convexHull(cnt) hull\_area = cv2.contourArea(hull) solidity = float(area)/hull\_area print("Solidity",solidity) img2 = img.copy() length = len(hull) for i in range(len(hull)):  cv2.line(img2,tuple(hull[i][0]),tuple(hull[(i+1)%length][0]),(0,0,255),2)  cv2.imshow("Result", img2) cv2.waitKey(0) cv2.destroyAllWindows()** |

#### 效果：

|  |
| --- |
|  |
|  |

### 4.等效直径

等效直径是圆的直径，其面积与轮廓面积相同：

image.png

### 实例

|  |
| --- |
|  |

### 结果：

|  |
| --- |
|  |

### 5.方向

其实际上就是指物体指向的角度，我们在之前的椭圆拟合里就讲过，它会返回三个参数：主轴和短轴长度以及角度

### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

### 6.掩模和像素点

在某些情况下，我们可能需要包含该对象的所有点，本质相当于用另一层图层覆盖当前的轮廓

### 实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |
|  |

#### 可以看到，轮廓内的图形都被蓝色的图层覆盖了，这在以后的综合项目中将非常有用，可以消除不必要的噪声

### 7.最大值，最小值及其位置

我们可以使用以下函数找到图像矩阵的最大值、最小值以及它们相应的索引位置：

**min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(gray)**

gray指代图像的灰度图像

### 实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

#### 事实上，该方法被广泛的应用在OpenCV的模板匹配中

### 8. 平均颜色或平均灰度

在这里，我们可以找到对象的平均颜色。或者它可以是灰度模式下物体的平均强度。我们再次使用相同的面具来做到这一点。

**mean\_val = cv.mean(im,mask = mask)**

### 实例

|  |
| --- |
|  |

#### 结果

|  |
| --- |
|  |
|  |

### 9.极点

极值点表示对象的最顶部，最底部，最右侧和最左侧的点。

### 实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

## 4.更多函数

### 1.凸缺陷

我们在前面学到了关于轮廓的凸包。物体与该凸包的任何偏差都可以被认为是凸缺陷。

OpenCV附带了一个现成的函数来查找它，cv.convexityDefects()。基本函数调用如下所示：

**hull = cv.convexHull(cnt,returnPoints = False)**

**defects = cv.convexityDefects(cnt,hull)**

**注意：我们必须在找到凸包时传递returnPoints = False，以便找到凸缺陷。**

它返回一个数组，其中每一行包含这些值 - [起点，终点，最远点，到最远点的近似距离]。我们可以使用图像将其可视化。我们绘制一条连接起点和终点的线，然后在最远点绘制一个圆。请记住，返回的前三个值是cnt的索引。所以我们必须从cnt中提取这些值。

### 实例

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

### 2. 点多边形测试

cv2.pointPolygonTest(contour, pt, measureDist) ->retval

参数：  
**contour**：输入轮廓。  
**pt**：要测试的点。  
**measureDist**：如果为真，该函数估计从点到最近的轮廓边缘的符号距离。否则，函数只检查点是否在轮廓线内。

该函数确定点是在轮廓线内、外，还是位于边缘(或与顶点重合)。它会相应地返回正(内部)、负(外部)或零(边缘)值。当**measureDist=false**时，返回值分别是\*\*+1\*\*，**-1**和**0**。否则，返回值是点和最近的轮廓边缘之间的带符号距离。

**此功能可查找图像中的点与轮廓之间的最短距离。当点在轮廓外时返回负值，当点在内部时返回正值，如果点在轮廓上则返回零。**

例如，我们可以检查点(50,50)如下：

dist = cv.pointPolygonTest(cnt,(50,50),True)

在函数中，第三个参数是measureDist。如果为True，则查找签名距离。如果为False，则查找该点是在内部还是外部或在轮廓上（它分别返回+1，-1，0）。

**注意：如果你不想找到距离，请确保第三个参数为False，因为这是一个耗时的过程。因此，将其设为False可提供2-3倍的加速。**

### 实例1

|  |
| --- |
| **from** \_\_future\_\_ **import** print\_function,division **import** cv2 **import** numpy **as** np  *#创建一幅图片* r = 100 src = np.zeros((4\*r,4\*r),np.uint8) *#创建6个顶点 # 创建六边形的6个顶点* vert = [**None**]\*6 vert[0] = (3\*r//2, int(1.34\*r)) vert[1] = (1\*r, 2\*r) vert[2] = (3\*r//2, int(2.866\*r)) vert[3] = (5\*r//2, int(2.866\*r)) vert[4] = (3\*r, 2\*r) vert[5] = (5\*r//2, int(1.34\*r)) *# 根据六个顶点画6边形* **for** i **in** range(6):  cv2.line(src,vert[i],vert[(i+1)%6],(255),3) *# 轮廓查找* \_,contours,\_ = cv2.findContours(src,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE) *# 计算图上点到六边形的距离（带符号）* raw\_dist = np.empty(src.shape,dtype=np.float32) **for** i **in** range(src.shape[0]):  **for** j **in** range(src.shape[1]):  raw\_dist[i,j] = cv2.pointPolygonTest(contours[0],(j,i),**True**)  *# 查找最大值和最小值* min\_val,max\_val,min\_loc,max\_loc = cv2.minMaxLoc(raw\_dist) minVal = abs(min\_val) maxVal = abs(max\_val) drawing = np.zeros((src.shape[0],src.shape[1],3),np.uint8) **for** i **in** range(src.shape[0]):  **for** j **in** range(src.shape[1]):  **if** raw\_dist[i,j] < 0:  drawing[i,j,0] = 255 - abs(raw\_dist[i,j])\*255/minVal  **elif** raw\_dist[i,j] > 0:  drawing[i,j,2] = 255 - abs(raw\_dist[i,j])\*255/maxVal  **else**:  drawing[i, j, 0] = 255  drawing[i, j, 1] = 255  drawing[i, j, 2] = 255 cv2.circle(drawing,max\_loc,int(maxVal),(0,0,255),2) cv2.imshow(**'Source'**,src) cv2.imshow(**'Distance and inscribed circle'**,drawing) cv2.waitKey(0) cv2.destroyAllWindows() |

#### 效果：

|  |
| --- |
|  |

### 3.匹配形状

OpenCV附带了一个函数cv.matchShapes()，它使我们能够比较两个形状或两个轮廓，并返回一个显示相似性的度量。结果越小，匹配就越好。它是根据hu-moment值计算的。

### 实例：

#### 测试用图片

|  |  |
| --- | --- |
|  |  |

#### 代码

|  |
| --- |
|  |

#### 注意：这里有一个坑，你必须慢慢调试找到正确的轮廓，不能一上来就认定第0个，因为我用第0个根本就得不到正确的效果，这里第一个轮廓组的第三个和第二个轮廓组的第八个(下标位7)

#### 效果：

|  |
| --- |
|  |

### 实例2.

#### 测试图片

|  |
| --- |
|  |

#### 代码

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

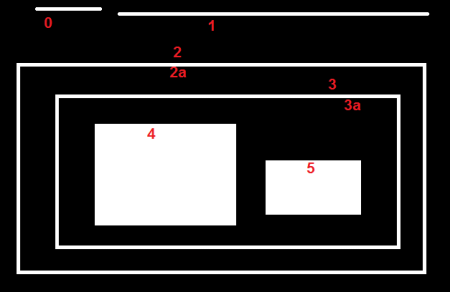
#### 从结果可以知道，自己和自己比对，符号0.0说明是完全一样，小星形和大星形对比，形状是很接近的，虽然大星形旋转了一个角度，但是不影响效果，所以这个返回值也是很小的，另外两个结果比较大，因为测试的两个形状不一样

## 5.层次结构

## 什么是层次结构？

通常我们使用cv.findContours()函数来检测图像中的对象，对吧？有时对象位于不同的位置。但在某些情况下，某些形状在其他形状内。就像嵌套的数字一样。在这种情况下，我们将外部一个称为父项，将内部项称为子项。这样，图像中的轮廓彼此之间存在某种关系。我们可以指定一个轮廓如何相互连接，例如，它是某个其他轮廓的子项，还是父项等。这种关系的表示称为层次结构。

考虑下面的示例图片：

[](https://camo.githubusercontent.com/531dcfee51e27316727bfc44163b6ec4fa5799323fb9c7ad93796f648b7c75db/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f6869657261726368792e706e67)

在这张图片中，有一些形状，我从0-5编号。图2和2a表示最外侧盒子的外部和内部轮廓。

这里，轮廓0,1,2是外部或最外部的。我们可以说，它们在层次结构0中，或者只是它们处于相同的层次结构级别。

接下来是轮廓-2a。它可以被认为是轮廓-2的子节点（或者相反，轮廓-2是轮廓-2a的父节点）。所以让它在层次结构-1中。类似地，轮廓-3是轮廓-2的子，它进入下一层次。最后，轮廓4,5是轮廓-3a的子节点，它们位于最后的层次结构级别。从我编号框的方式，我会说轮廓-4是轮廓-3a的第一个孩子（它也可以是轮廓-5）。

我提到这些东西来理解相同的层次结构，外部轮廓，子轮廓，父轮廓，第一个孩子等术语。现在让我们进入OpenCV。

## OpenCV中的层次结构表示

因此每个轮廓都有自己的信息，关于它是什么层次结构，谁是它的子，谁是它的父等.OpenCV将它表示为四个值的数组：[Next，Previous，First\_Child，Parent]

**“下一个表示同一层级的下一个轮廓。”**

例如，在我们的图片中取出contour-0。谁是同一水平的下一个轮廓？它是轮廓-1。所以简单地说Next = 1.类似地，对于Contour-1，next是contour-2。所以Next = 2。

轮廓-2怎么样？同一级别没有下一个轮廓。所以简单地说，将Next = -1。轮廓-4怎么样？它与contour-5处于同一水平。所以它的下一个轮廓是轮廓-5，所以Next = 5。

**“上一个表示同一层级的先前轮廓。”**

与上述相同。轮廓-1的先前轮廓在同一水平面上为轮廓-0。类似地，对于轮廓-2，它是轮廓-1。而对于contour-0，没有先前的，所以把它作为-1。

**“First\_Child表示其第一个子轮廓。”**

无需任何解释。对于轮廓-2，孩子是轮廓-2a。因此它获得了contour-2a的相应索引值。轮廓-3a怎么样？它有两个孩子。但我们只带第一个孩子。它是轮廓-4。因此，对于轮廓-3a，First\_Child = 4。

**“父表示其父轮廓的索引。”**

它与First\_Child相反。对于轮廓-4和轮廓-5，父轮廓都是轮廓-3a。对于轮廓-3a，它是轮廓-3，依此类推。

**注意：如果没有子项或父项，则该字段将被视为-1**

所以现在我们知道OpenCV中使用的层次结构样式，我们可以在上面给出的相同图像的帮助下检查OpenCV中的Contour Retrieval Modes。即cv.RETR\_LIST，cv.RETR\_TREE，cv.RETR\_CCOMP，cv.RETR\_EXTERNAL等标志是什么意思？

## 轮廓检索模式

### RETR\_LIST

这是四个标志中最简单的（从解释的角度来看）。它只是检索所有轮廓，但不创建任何父子关系。根据这条规则，父和子是平等的，他们只是轮廓。即它们都属于同一层次结构。

所以这里，层次结构数组中的第3和第4项始终为-1。但显然，Next和Previous术语将具有相应的值。请自行检查并验证。

下面是我得到的结果，每行是相应轮廓的层次结构细节。例如，第一行对应于轮廓0.下一个轮廓是轮廓1.所以Next = 1.没有先前的轮廓，所以Previous = -1。如前所述，剩下的两个是-1。

>>> hierarchy

array([[[ 1, -1, -1, -1],

[ 2, 0, -1, -1],

[ 3, 1, -1, -1],

[ 4, 2, -1, -1],

[ 5, 3, -1, -1],

[ 6, 4, -1, -1],

[ 7, 5, -1, -1],

[-1, 6, -1, -1]]])

如果你没有使用任何层次结构功能，这是在代码中使用的不错选择。

### RETR\_EXTERNAL

如果使用此标志，则仅返回极端外部标志。所有儿童轮廓都被遗忘。我们可以说，根据这项法律，只有每个家庭中最年长的人才能得到照顾。它并不关心其他家庭成员:)。

那么，在我们的图像中，有多少极端外轮廓？即在等级0级？只有3个，即轮廓0,1,2，对吧？现在尝试使用此标志查找轮廓。这里，给予每个元素的值与上面相同。将其与上述结果进行比较。以下是我得到的：

>>> hierarchy

array([[[ 1, -1, -1, -1],

[ 2, 0, -1, -1],

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

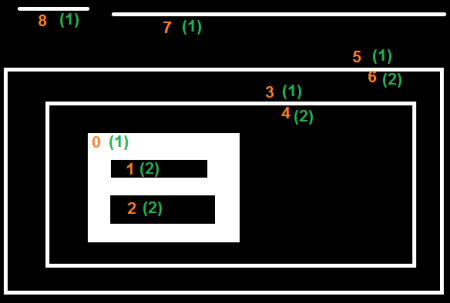
如果只想提取外轮廓，可以使用此标志。在某些情况下它可能有用。

### RETR\_CCOMP

此标志检索所有轮廓并将它们排列为2级层次结构。即对象的外部轮廓（即其边界）放置在层次结构-1中。对象内部的孔的轮廓（如果有的话）放在层次结构-2中。如果其中有任何对象，则其轮廓仅再次放置在层次结构-1中。它在层次结构-2中的漏洞等等。

只需考虑黑色背景上的“大白零”图像。零的外圆属于第一层次，零的内圈属于第二层次。

我们可以用简单的图像来解释它。在这里，我用红色（1或2）标记了红色轮廓的顺序和它们所属的层次结构。顺序与OpenCV检测轮廓的顺序相同。

[](https://camo.githubusercontent.com/ae45265f993ff88d8a8eae3f5bb065c477a1611a29266523d7d04796167d3427/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f63636f6d705f6869657261726368792e706e67)

因此，考虑第一个轮廓，即轮廓-0。它是层次结构-1。它有两个孔，轮廓1和2，它们属于层次结构-2。因此对于轮廓-0，相同层级中的下一轮廓是轮廓-3。并且之前没有。它的第一个是子级是层次结构-2中的轮廓-1。它没有父级，因为它位于层次结构-1中。所以它的层次结构数组是[3，-1,1，-1]

现在采取轮廓-1。它在层次结构-2中。同一层次中的下一个（在轮廓-1的父下面）是轮廓-2。没有前一个。没有子，但父是轮廓-0。所以数组是[2，-1，-1,0]。

类似于contour-2：它在层次结构-2中。在contour-0下，同一层次中没有下一个轮廓。所以没有下一个。以前是轮廓-1。没有子，父是轮廓-0。所以数组是[-1,1，-1,0]。

轮廓-3：层次结构-1中的下一个是轮廓-5。上一个是轮廓-0。子是轮廓4而没有父。所以数组是[5,0,4，-1]。

轮廓 - 4：它在等高线3中的等级2中，并且没有兄弟。所以没有下一个，没有先前，没有子，父是轮廓-3。所以数组是[-1，-1，-1,3]。

剩下的你可以填写。这是我得到的最终答案：

>>> hierarchy

array([[[ 3, -1, 1, -1],

[ 2, -1, -1, 0],

[-1, 1, -1, 0],

[ 5, 0, 4, -1],

[-1, -1, -1, 3],

[ 7, 3, 6, -1],

[-1, -1, -1, 5],

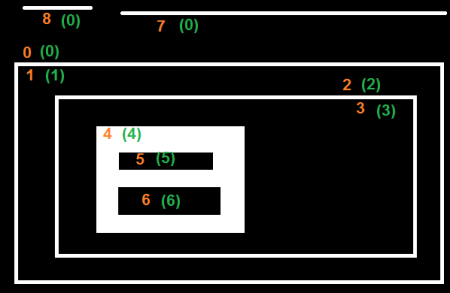
[ 8, 5, -1, -1],

[-1, 7, -1, -1]]])

### RETR\_TREE

这是最后一个人，Mr.Perfect。它检索所有轮廓并创建完整的族层次结构列表。它甚至告诉，谁是爷爷，父，子，孙子，甚至超越...... :)。

例如，我拍摄了上面的图像，重写了cv.RETR\_TREE的代码，根据OpenCV给出的结果重新排序轮廓并进行分析。同样，红色字母给出轮廓编号，绿色字母给出层次结构顺序。

[](https://camo.githubusercontent.com/ac6a285a97e46f0ee4e0eafd6ea754aa05e097b70c22b47cbf4d301576c8123a/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f747265655f6869657261726368792e706e67)

取contour-0：它在层次结构-0中。 同一层次中的下一个轮廓是轮廓-7。没有以前的轮廓。子是轮廓-1。 没有父。 所以数组是[7，-1,1，-1]。

取等高线2：它在层次结构-1中。同一级别没有轮廓。没有前一个。子是轮廓-3。父是轮廓-1。所以数组是[-1，-1,3,1]。

剩下的，试试吧。 以下是完整的答案：

>>> hierarchy

array([[[ 7, -1, 1, -1],

[-1, -1, 2, 0],

[-1, -1, 3, 1],

[-1, -1, 4, 2],

[-1, -1, 5, 3],

[ 6, -1, -1, 4],

[-1, 5, -1, 4],

[ 8, 0, -1, -1],

[-1, 7, -1, -1]]])