[**OpenCV3-Python轮廓检测流程介绍**](http://www.yyearth.com/article/18-05/225.html)

admin 于 2018年05月10日 发表在 [计算机视觉](http://www.yyearth.com/index.php?cid=24)

在计算机视觉中，轮廓检测是另外一个比较重要的任务，不单用来检测图像或视频帧中物体的轮廓，而且还有其它操作与轮廓检测有关，如：**计算多边形边界、形状逼近和计算感兴趣区域**。

Numpy中的矩形区域可以使用数组切片(slice)来定义，因此在介绍物体检测(包括人脸)和物体跟踪时，会大量使用这种技术。

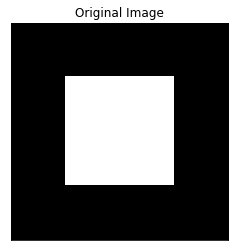
**1. 导入所需模块**

|  |  |
| --- | --- |
|  | import cv2  import numpy as np  from matplotlib import pyplot as plt |

**2. Numpy数组切片**

创建一个200x200的黑色空白图像，接着利用数组切片在图像中放置一个100\*100的白色方块。

|  |  |
| --- | --- |
|  | img = np.zeros((200, 200), dtype=np.uint8)  img[50:150, 50:150] = 255    cv2.imshow("Original Image", img)  cv2.waitKey()  cv2.destroyAllWindows() |



**3. 二值化处理（**[**官网**](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding)**）**

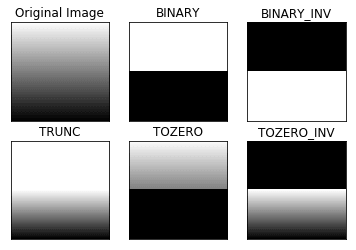
**cv2.threshold(src,thresh,maxval,type[,dst])**

thresh : 源图像，应该是灰度图像；

thresh : 分类像素值的阈值；

maxval : 表示在像素值大于（有时小于）阈值时要给出的值；

type：对应OpenCV不同风格的阈值，有5种不同类型；



|  |
| --- |
| import cv2  import numpy as np  from matplotlib import pyplot as plt  img = cv2.imread('gradient.png',0)  #cv2.THRESH\_BINARY 黑白二值化  ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)  #cv2.THRESH\_BINARY\_INV 黑白二值化反转  ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)  #cv2.THRESH\_TRUNC 多像素值  ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)  #cv2.THRESH\_TOZERO 阈值化为0  ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)  #cv2.THRESH\_TOZERO\_INV 反转阈值化到  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() |

**4. 查找轮廓**

**cv2.findContours(image, mode, method[, contours[, hierarchy[, offset ]]])**

image: 所需寻找轮廓的图像，8位单通道太图像。零像素保持为0，非零像素视为1；

mode: 轮廓的检索模式；

method: 为轮廓的近似办法；

返回值: cv2.findContours()函数返回两个值，一个是轮廓本身，还有一个是每条轮廓对应的属性。

|  |
| --- |
| '''  mode: 轮廓的检索模式，有如下四种：  cv2.RETR\_EXTERNAL: 只检测外轮廓，为所有轮廓设置阶层；  cv2.RETR\_LIST: 检测所有轮廓而不建立层次关系  cv2.RETR\_CCOMP: 检测所有轮廓，建立两个等级的轮廓； 上面一层为外边界，里面的一层为内孔的边界信息。如果内孔内还有一个连通物体，这个物体的边界也在顶层。  cv2.RETR\_TREE: 检测所有轮廓并重建一个等级树结构的轮廓。  method: 为轮廓的近似办法：  cv2.CHAIN\_APPROX\_NONE存储所有的轮廓点，相邻的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，例如一个矩形轮廓只需4个点来保存轮廓信息  cv2.CHAIN\_APPROX\_TC89\_L1，CV\_CHAIN\_APPROX\_TC89\_KCOS使用teh-Chinl chain 近似算法  '''  image, contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  color = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR) |

**5. 绘制图像轮廓**

函数cv2.drawContours()被用来绘制轮廓，它可以根据你提供的边界点绘制任何形状。

**cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]]) → image**

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

image: 目标图像，

contours: 所有输入轮廓。每个轮廓都存储为点矢量，

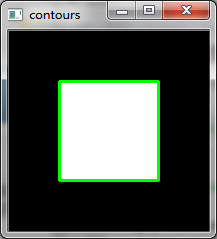
contourIdx: 绘制轮廓的索引（在绘制独立轮廓是很有用，当设置为-1时绘制所有轮廓）。

color: 轮廓的颜色。

|  |  |
| --- | --- |
| 1 | img = cv2.drawContours(color, contours, -1, (0,255,0), 2) |

**6. 显示图像轮廓**

|  |  |
| --- | --- |
|  | cv2.imshow("contours", color)  cv2.waitKey()  cv2.destroyAllWindows() |



**OpenCv学习笔记--轮廓检测,多边形 直线 圆检测**

**此opencv系列博客只是为了记录小编对<<opencv3计算机视觉-pyhton语言实现>>的学习笔记,所有代码在我的github主页**[**https://github.com/RenDong3/OpenCV\_Notes**](https://github.com/RenDong3/OpenCV_Notes)**.**

**欢迎star,不定时更新...**

**一 轮廓检测**

在计算机视觉中，轮廓检测是另一个比较重要的任务，不单是用来检测图像或者视频帧中物体的轮廓，而且还有其他操作与轮廓检测相关。这些操作中，计算多边形边界，形状逼近和计算机感 兴趣区域。这是与图像数据交互时的简单操作，因为numpy中的矩阵中的矩形区域可以使用数组切片(slice)定义。在介绍物体检测(包括人脸)和物体跟踪的概念时会大量使用这种技术。

**1、**[**cv2.threshold(src,thresh,maxval,type[,dst])**](https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=cv2.threshold#cv2.threshold)函数用于图像阈值操作。

cv2.threshold(src, thresh, maxval, type[, dst]) → retval, dst

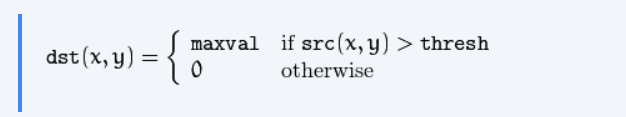
|  |  |
| --- | --- |
| **参数：** | * **src** - 输入数组（单通道，8位或32位浮点）。 * **dst** - 输出与...大小和类型相同的数组src。 * **脱粒** -阈值。 * **maxval** - THRESH\_BINARY与THRESH\_BINARY\_INV阈值和阈值类型一起使用的最大值。 * **type** - thresholding类型（见下面的详细信息）。 |

该功能将固定级阈值应用于单通道阵列。该函数通常用于从灰度图像中获取二值（二值）图像（ [compare()](https://docs.opencv.org/2.4/modules/contrib/doc/openfabmap.html#void%20compare%28const%20Mat&%20queryImgDescriptor,%20vector%3CIMatch%3E&%20matches,%20bool%20addQuery%20,%20const%20Mat&%20mask%29)也可用于此目的）或用于去除噪声，即滤除具有太小或太大值的像素。函数支持几种类型的阈值处理。它们由以下因素决定type：

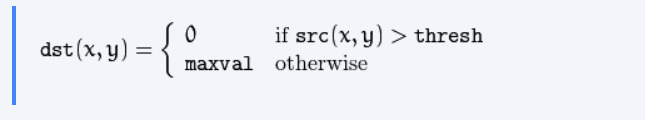
为了从一幅图像中提取我们需要的部分，应该用图像中的每一个像素点的灰度值与选取的阈值进行比较，并作出相应的判断(阈值的选取依赖于具体的问题，物体在不同的图像中可能会有不同的灰度值)。opencv提供了threshold()函数对图像的阈值进行处理，threshold()共支持五中类型的阈值化方式，分别是二进制阈值化、反二进制阈值化、截断阈值化、阈值化为0和反阈值化为0。返回阈值操作后的图像。

* src: 输入图像，图像必须为单通道8位或32位浮点型图像
* thresh: 设定的阈值
* maxval: 使用cv2.THRESH\_BINARY和cv2.THRESH\_BINARY\_INV类型的最大值
* type: 阈值化类型，可以通过ThresholdTypes查看，下面给出opencv中五种阈值化类型及其对应公式：

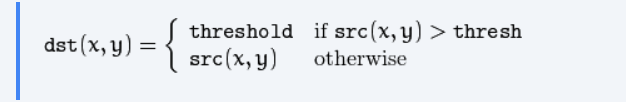
THRESH\_BINARY



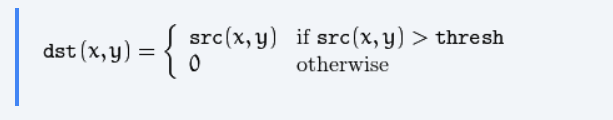
THRESH\_BINARY\_INV



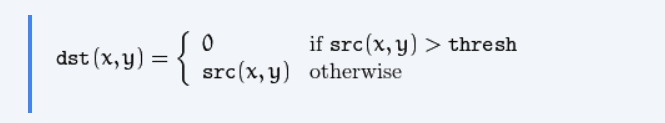
THRESH\_TRUNC



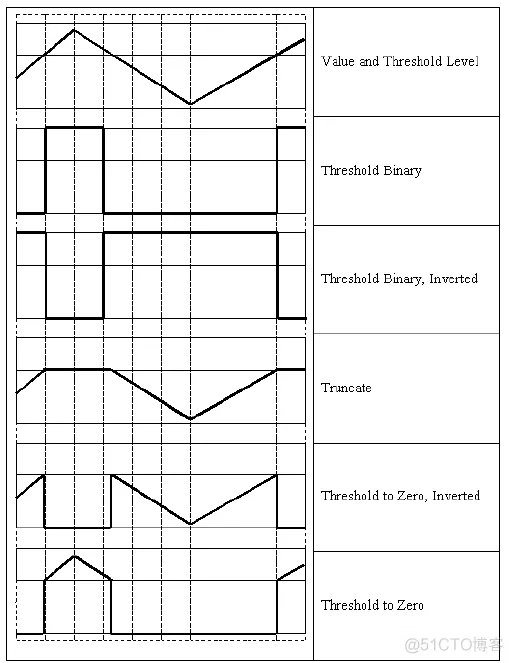
THRESH\_TOZERO



THRESH\_TOZERO\_INV



此外，特殊值THRESH\_OTSU可以与上述值之一组合。在这种情况下，函数使用Otsu算法确定最佳阈值，并使用它而不是指定的thresh。该函数返回计算的阈值。目前，Otsu的方法仅适用于8位图像。



**2、**[**cv2.findContours(image,mode,method[,contours,hierarchy[,offset]])**](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.findcontours#cv2.findContours)用于寻找寻找图像轮廓。

cv2.findContours(image, mode, method[, contours[, hierarchy[, offset]]]) → contours, hierarchy

opencv中提供[findContours()](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.find#cv2.findContours)函数来寻找图像中物体的轮廓，并结合[drawContours()](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.find#drawcontours)函数将找到的轮廓绘制出。这个函数会修改输入图像，因此建议使用原始图像的一份拷贝(比如说img.copy()作为输入图像)。**函数返回三个值：返回修改后的图像，图像的轮廓以及它们的层次。**

|  |  |
| --- | --- |
| **参数：** | * **image** - Source，一个8位单通道图像。非零像素被视为1。零像素保持为0，因此图像被视为 binary。您可以使用[compare()](https://docs.opencv.org/2.4/modules/contrib/doc/openfabmap.html#void%20compare%28const%20Mat&%20queryImgDescriptor,%20vector%3CIMatch%3E&%20matches,%20bool%20addQuery%20,%20const%20Mat&%20mask%29)， [inRange()](https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#void%20inRange%28InputArray%20src,%20InputArray%20lowerb,%20InputArray%20upperb,%20OutputArray%20dst%29)， [threshold()](https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#double%20threshold%28InputArray%20src,%20OutputArray%20dst,%20double%20thresh,%20double%20maxval,%20int%20type%29)， [adaptiveThreshold()](https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#void%20adaptiveThreshold%28InputArray%20src,%20OutputArray%20dst,%20double%20maxValue,%20int%20adaptiveMethod,%20int%20thresholdType,%20int%20blockSize,%20double%20C%29)， [Canny()](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#void%20Canny%28InputArray%20image,%20OutputArray%20edges,%20double%20threshold1,%20double%20threshold2,%20int%20apertureSize,%20bool%20L2gradient%29)，和其他人出灰度或彩色之一创建一个二进制图像。该功能修改了image 提取轮廓的 同时。如果mode等于CV\_RETR\_CCOMP或CV\_RETR\_FLOODFILL，则输入也可以是标签（CV\_32SC1）的32位整数图像。 * **轮廓** - 检测到的轮廓。每个轮廓都存储为点矢量。 * **hierarchy** - 可选输出向量，包含有关图像拓扑的信息。它具有与轮廓数量一样多的元素。对于每个第i个轮廓 contours[i]，元件 hierarchy[i][0]， hiearchy[i][1]， hiearchy[i][2]，和 hiearchy[i][3] 被设置为0为基础的指数在 contours 在同一层级中，第一子轮廓和父轮廓分别下一个和先前轮廓的。如果轮廓 i 没有下一个，前一个，父级或嵌套轮廓，则相应的元素hierarchy[i] 将为负数。 |

* **image:**输入图像，**函数接受的参数是二值图，即黑白的(不是灰度图)**，我们同样可以使用cv2.compare,cv2.inRange,cv2.threshold,cv2.adaptiveThreshold,cv2.Canny等函数来创建二值图像，如果第二个参数为cv2.RETR\_CCOMP或cv2.RETR\_FLOODFILL，输入图像可以是32-bit整型图像(cv2.CV\_32SC1)
* **mode**轮廓检索模式，如下

**模式** -

轮廓检索模式（如果使用Python，请参阅下面的注释）。

* **CV\_RETR\_EXTERNAL**仅检索极端外轮廓。它 hierarchy[i][2]=hierarchy[i][3]=-1 适用于所有轮廓。
* **CV\_RETR\_LIST**检索所有轮廓而不建立任何层次关系。
* **CV\_RETR\_CCOMP**检索所有轮廓并将它们组织成两级层次结构。在顶层，组件有外部边界。在第二级，有孔的边界。如果连接组件的孔内有另一个轮廓，它仍然位于顶层。
* **CV\_RETR\_TREE**检索所有轮廓并重建嵌套轮廓的完整层次结构。这个完整的层次结构是在OpenCV contours.c 演示中构建和显示的 。
* **method:**轮廓近似方法
* **方法** -

轮廓近似方法（如果使用Python，请参见下面的注释）。

* + **CV\_CHAIN\_APPROX\_NONE**绝对存储所有轮廓点。也就是说，任何2个后续点(x1,y1)和(x2,y2)轮廓将是水平，垂直或对角线邻居，即max(abs(x1-x2),abs(y2-y1))==1。
  + **CV\_CHAIN\_APPROX\_SIMPLE**压缩水平，垂直和对角线段，仅留下其端点。例如，右上方的矩形轮廓用4个点编码。
  + **CV\_CHAIN\_APPROX\_TC89\_L1，CV\_CHAIN\_APPROX\_TC89\_KCOS**应用了Teh-Chin链式近似算法的一种风格。有关详细信息，请参阅 [[TehChin89]](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.find#tehchin89)。
* **offset** - 每个轮廓点移动的可选偏移量。如果从图像ROI中提取轮廓然后应在整个图像上下文中分析轮廓，则这非常有用。

**3、**[**cv2.drawContours(image,contours,contourIdx,color[,thickness[,lineType[,hierarchy[,maxLevel[,offset]]]]])**](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.findcontours#cv2.findContours)函数轮廓绘制。

cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]]) → None

* 1.

该函数返回绘制有轮廓的图像。

 为什么有了一个精确表示的轮廓却还需要得到一个近似多边形呢？这是因为一个多边形由一组直线构成，能够在一个区域里定义多边形，以便于之后进行操作与处理，这在许多计算机视觉任务中非常重要。ε值对获取有用的轮廓非常重要，所以需要理解它表示什么意思。

ε是为所得到的近似多边形周长与源轮廓周长之间的最大差值，这个值越小，近似多边形与源轮廓就越相似。

在了解了ε值是什么之后，需要得到轮廓的周长信息来作为参考值。这可以通过cv2.arcLength函数来完成：

#arcLength获取轮廓的周长

epsilon = 0.01\*cv2.arcLength(cnt,True)

#计算矩形的多边形框

approx = cv2.approxPolyDP(cnt,epsilon,True)

可以通过OpenCV来有效地计算一个近似多边形。为了计算凸形状，需要利用[cv2.convexHull](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.convexhull#cv2.convexHull)来处理获取的轮廓信息。

* **image:**输入/输出图像，指明在哪个图像上绘制轮廓。并且该函数会修改源图像image。
* **contours:**使用findContours检测到的轮廓数据，传入一个list。
* **contourIdx:**绘制轮廓的索引变量(表示绘制第几个轮廓)，如果为负值则绘制所有输入轮廓。
* **color:**轮廓颜色。
* **thickness:**绘制轮廓所用线条粗细度，如果值为负值，则在轮廓内部绘制。
* **lineTpye:**线条类型，有默认值LINE\_8，有如下可选类型
* **hierarchy:**可选层次结构信息
* **maxLevel:**用于绘制轮廓的最大等级。
* **offset:**可选轮廓便宜参数，用制定偏移量offset=(dx, dy)给出绘制轮廓的偏移量。
* 完整代码如下:

#-\*\_ coding:utf-8 -\*-

import cv2

import numpy as np

'''

created on Tues jan 08:28:51 2018

@author: ren\_dong

contour detection

cv2.findContours() 寻找轮廓

cv2.drawContours() 绘制轮廓

'''

#加载图像img

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

cv2.imshow('origin', img)

'''

灰度化处理,注意必须调用cv2.cvtColor(),

如果直接使用cv2.imread('1.jpg',0),会提示图像深度不对,不符合cv2.CV\_8U

'''

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

cv2.imshow('gray', gray)

#调用cv2.threshold()进行简单阈值化,由灰度图像得到二值化图像

# 输入图像必须为单通道8位或32位浮点型

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

cv2.imshow('thresh', thresh)

#调用cv2.findContours()寻找轮廓,返回修改后的图像,轮廓以及他们的层次

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

cv2.imshow('image', image)

print('contours[0]:',contours[0])

print('len(contours):',len(contours))

print('hierarchy.shape:',hierarchy.shape)

print('hierarchy:',hierarchy)

#调用cv2.drawContours()在原图上绘制轮廓

img = cv2.drawContours(img, contours, -1, (0, 255, 0), 2)

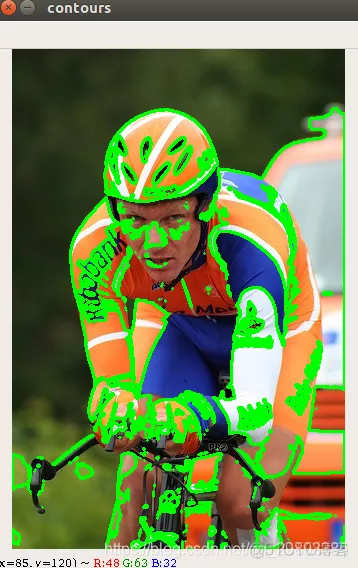
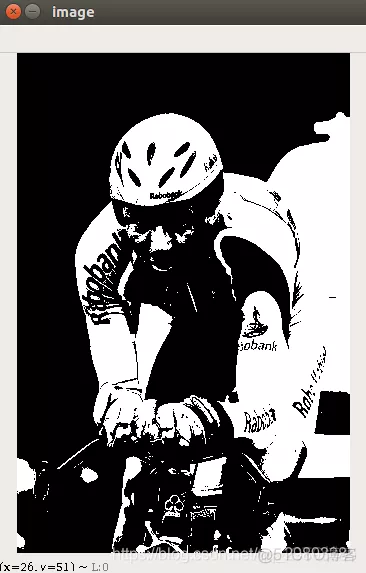
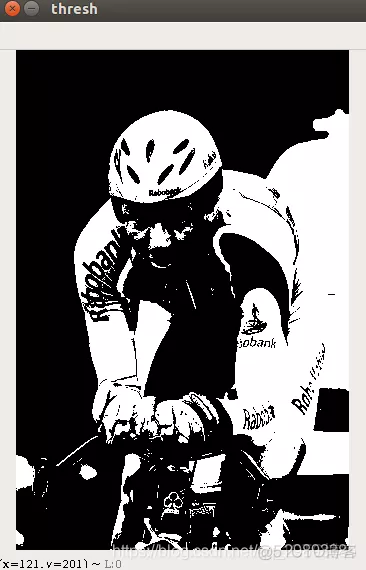
cv2.imshow('contours', img)

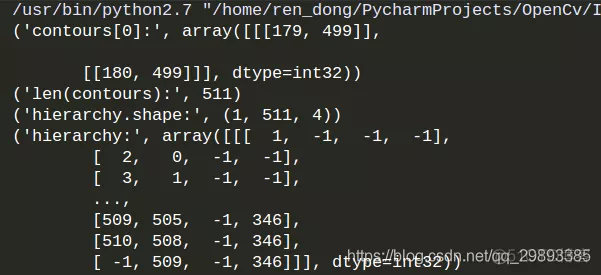
cv2.waitKey()

cv2.destroyAllWindows()

运行后的结果如下:







**二 边界框、最小矩形区域和最小闭圆的轮廓**

 找到一个正方形轮廓很简单，要找到到不规则的，歪斜的以及旋转的形状，可以用Open CV的cv2.findContours()函数，它能得到最好的结果，下面来看一副图：



现实的应用会对目标的边界框，最小矩形面积，最小闭圆特别感兴趣，将cv2.findContours()函数和少量的OpenCV的功能相结合就非常容易实现这些功能：

使用[boundingRect()](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.boundin#cv2.boundingRect)函数计算包围轮廓的矩形框，使用[minEnclosi](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.minenclosi#cv2.minEnclosingCircle)ngCircle()函数计算包围轮廓的最小圆包围。

1、先计算一个简单的边界框(水平矩形)：

x,y,w,h = cv2.boundingRect(c)

* **cv2.boundingRect(img)函数**。函数计算并返回点集最外面的矩形边界(参数一般传入一个轮廓，contours[0])，函数返回四个值，分别是x,y,w,h。x,y是矩阵左上点的左边，w,h是矩阵的宽和高。
* 然后画出这个矩形(在原图img上绘制)：这个操作非常简单，它将轮廓信息转换为(x,y)坐标，并加上矩形的高度和宽度。

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

下面来将如何找到一个旋转的矩阵和圆形轮廓。 首先加载图片，然后在源图像的灰度图像上面执行一个二值化操作。这样之后，可在这个灰度图像上执行所有计算轮廓的操作，但在源图像上可利用色彩信息来画这些轮廓。

2、计算包含出包围目标的最小矩形区域(旋转矩形)：

#找到最小区域

rect = cv2.minAreaRect(c)

#计算最小矩形的坐标

box = cv2.boxPoints(rect)

#坐标转换为整数

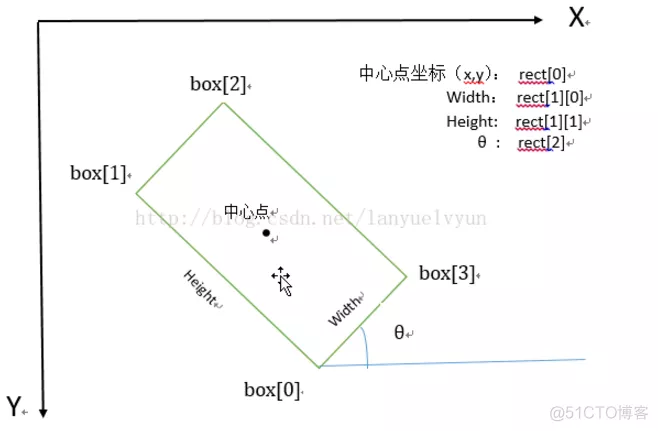
box = np.int0(box)

这里用到一个非常有趣的机制：Open CV没有函数能直接从轮廓信息中计算出最小矩形顶点的坐标。所以需要计算最小矩形区域，然后计算这个矩形的顶点。注意计算出来的顶点左边是浮点型，但是所得像素的坐标值是整数，所以需要做一个转换。

函数 [cv2.minAreaRect()](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.minarea#cv2.minAreaRect)返回一个tuple：（最小外接矩形的中心（x，y），（宽度，高度），旋转角度）。

但是要绘制这个矩形，我们需要矩形的4个顶点坐标box, 通过函数 cv2.cv.BoxPoints() 获得，box：[ [x0,y0], [x1,y1], [x2,y2], [x3,y3] ]

最小外接矩形的4个顶点顺序、中心坐标、宽度、高度、旋转角度（是度数形式，不是弧度数）的对应关系如下：



注意：旋转角度θ是水平轴（x轴）逆时针旋转，与碰到的矩形的第一条边的夹角。并且这个边的边长是width，另一条边边长是height。也就是说，在这里，width与height不是按照长短来定义的。

在opencv中，坐标系原点在左上角，相对于x轴，逆时针旋转角度为负，顺时针旋转角度为正。在这里，θ∈（-90度，0]。

然后画出这个矩形(在原图img上绘制)：

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

首先，该函数与所有绘图函数一样，它会修改源，其次该函数的第二个参数接收一个保存着轮廓的数组，从而可以在一次操作中绘制一系列的轮廓。因此如果只有一组点来表示多边形轮廓，可以把这组点放到一个list中，就像前面例子里处理方框(box)那样。这个函数第三个参数是绘制的轮廓数组的索引，-1表示绘制所有的轮廓，否则只绘制轮廓数组里指定的轮廓。

大多数绘图函数把绘图的颜色和线宽放在最后两个参数里。

3、最后检查的边界轮廓为最小闭圆。

#计算闭圆中心店和和半径

(x,y),radius = cv2.minEnclosingCircle(c)

#转换为整型

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

radius = int(radius)

#绘制闭圆(在原图img上绘制)

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

* **cv2.minEnclosingCircle(points)函数** 利用迭代算法，对给定的二维点集寻找计算可包围点集的最小圆形。
* points:输入的二维点集，一般传入一个轮廓 contours[0]

cv2.minEnclosingCircle()函数会返回一个元组，第一个元素为圆心的坐标组成的元素，第二个元素为圆的半径值。把这些值转换为整数后就能很容易地绘制出圆来。

完整代码如下：

#-\*- coding:utf-8 -\*-

import cv2

import numpy as np

'''

created on Tues jan 09:36:30 2018

@author:ren\_dong

cv2.boundingRect() 边界框即直边界矩形

cv2.minAreaRect() 最小矩形区域即旋转的边界矩形

cv2.minEnclosingCircle() 最小闭圆

'''

#载入图像img

img = cv2.pyrDown(cv2.imread('star.jpg', cv2.IMREAD\_UNCHANGED))

#灰度化

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

#二值化

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

#寻找轮廓

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

print('hierarchy[0]:', hierarchy[0])

print('len(contours):', len(contours))

print('hierarchy.shape:', hierarchy.shape)

#遍历每一个轮廓

for C in contours:

#计算边界框坐标

x, y, w, h = cv2.boundingRect(C)

##在Img图像上绘制矩形框,颜色为green, 线宽为2

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

#计算包围目标的最小矩形区域

rect = cv2.minAreaRect(C)

#计算最小矩形的坐标

box = cv2.boxPoints(rect)

#坐标变为整数

box = np.int0(box)

#绘制矩形框轮廓 颜色为red 线宽为3

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

#最小闭圆的圆心和半径

(x,y),radius = cv2.minEnclosingCircle(C)

#转换为整型

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

radius = int(radius)

#绘制最小闭圆

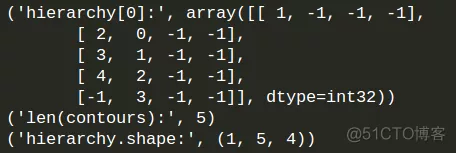
img = cv2.circle(img,center, radius, (255, 0, 0), 2)

cv2.drawContours(img,contours, -1, (0, 0, 255), 1)

cv2.imshow('contours',img)

cv2.waitKey()

cv2.destroyAllWindows()运行后的结果:

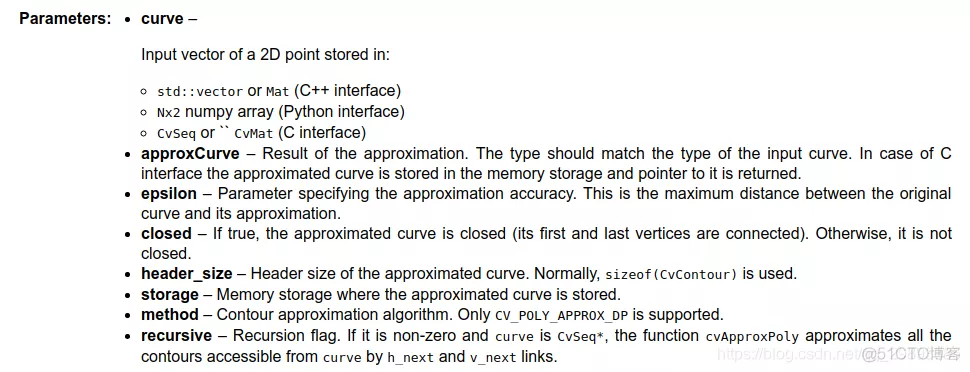


**三 凸轮廓与Douglas-Peucker算法**

 大多数处理轮廓的时候，图的形状(包括凸形状)都是变化多样的。凸形状内部的任意两点的连线都在该形状内部。

[cv2.approxPloyDP](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cv2.approx#cv2.approxPolyDP)函数，它用来计算近似的多边形框。该函数有三个参数：

* 第一个参数为轮廓
* 第二个参数为ε值，它表示源轮廓与近似多边形的最大差值(这个值越小，近似多边形与源轮廓越接近)
* 第三个参数为布尔标记，它表示这个多边形是否闭合



* #从轮廓信息中计算得到凸形状
* hull = cv2.convexHull(cnt)

 为了理解源轮廓、近似多边形和凸包的不同之处，可以把他们放在一副图片中进行观察：

#-\*- coding:utf-8 -\*-

import cv2

import numpy as np

'''

created on Tues jan 10:49:30 2018

@author:ren\_dong

凸轮廓和Douglas-Peucker算法

cv2.approxPloyDP()

CV2.arcLength()

cv2.convexHull()

'''

#读入图像img

img = cv2.pyrDown(cv2.imread('arc.jpg', cv2.IMREAD\_COLOR))

#resize

#img = cv2.resize(img, None, fx=0.6, fy=0.6, interpolation=cv2.INTER\_CUBIC)

#创建空白图像,用来绘制多边形轮廓

curve = np.zeros(img.shape,np.uint8)

#灰度变换

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

#使用定制kernel 进行中值滤波,去除一些噪声

kernel = np.ones((3,3),np.float32) / 9

#这里的-1表示目标图像和原图像具有同样的深度,比如cv2.CV\_8U

gray = cv2.filter2D(gray,-1,kernel)

#阈值化 gray image --> binary image

# 输入图像必须为单通道8位或32位浮点型

# 这里使用cv2.THRESH\_BINARY\_INV 实现效果像素>125 设置为0(黑) 否则设置为255(白)

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

#寻找轮廓 返回修改后的图像, 图像轮廓 以及他们的层次

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

##假设contours[0]为最大轮廓

cnt = contours[0]

max\_area = cv2.contourArea(cnt)

##遍历contours, 和原始设置比较,获得最大轮廓区域

for i in contours:

if cv2.contourArea(i) > max\_area:

cnt = i

max\_area = cv2.contourArea(cnt)

print('max\_area:',max\_area)

##获得轮廓周长

epsilon = 0.01 \* cv2.arcLength(cnt, True)

#计算得到近似的多边形框

approx = cv2.approxPolyDP(cnt, epsilon, True)

#得到凸包

hull = cv2.convexHull(cnt)

print('contours', len(contours), type(contours))

print('cnt.shape', cnt.shape, type(cnt))

print('approx.shape', approx.shape, type(approx))

print('hull.shape', hull.shape, type(hull))

#在原图像得到原始的轮廓

cv2.drawContours(img, contours, -1, (255, 0 , 0),2)

#在空白图像中得到最大轮廓, 多边形轮廓, 凸包轮廓

cv2.drawContours(curve, [cnt], -1, (0, 0, 255), 1) #

cv2.drawContours(curve, [hull], -1, (0, 255, 0), 2) ##绿色多边形轮廓 线宽2

cv2.drawContours(curve, [approx], -1, (255, 0, 0), 3) ##蓝色凸包轮廓 线宽3

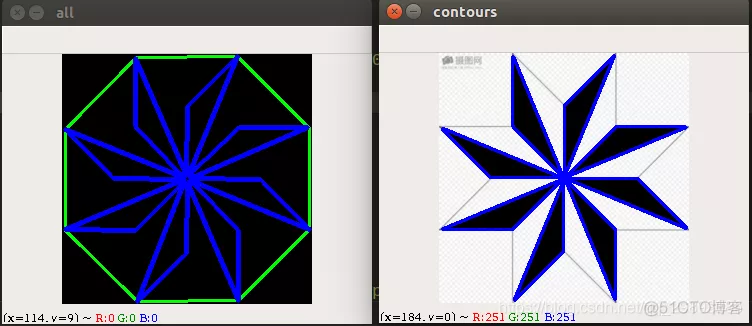
cv2.imshow('contours',img)

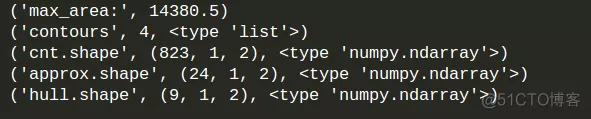
cv2.imshow('all',curve)

cv2.waitKey()

cv2.destroyAllWindows()

 运行结果如下:





如上图所示，凸包是由绿色表示，然后里面是近似多边形,使用黄色表示，在两者之间的是源图片中一个最大的轮廓，它主要由弧线构成.

* **四 直线和圆检测**
* 检测边缘和轮廓不仅重要，还经常用到，它们也是构成其他复杂操作的基础。直线和形状检查与边缘和轮廓检测有密切的关系。

Hough变换是直线和形状检测背后的理论基础，它由Richard Duda和Peter Hart发明，他们是对Paul Hough在20世纪60年代早期所做工作的扩展。

1、直线检测

首先介绍直线检测，这可通过[HoughLines](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines#cv2.HoughLines)和[HoughLinesP](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines#houghlinesp)函数来完成，它们仅有的差别是:第一个函数使用标准的Hough变换，第二个函数使用概率Hough变换(因此名称里有一个P)。

[HoughLinesP](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines#houghlinesp)函数之所以称为概率版本的Hough变换是因为它只通过分析点的子集并估计这些点都属于一条直线的概率，这是标准Hogh变换的优化版本。该函数的计算代价会少一些，执行会变得更快。

* 完整代码如下:

#-\*- coding:utf-8 -\*-

import cv2

import numpy as np

'''

created on Tues jan 13:57:30 2018

@author:ren\_dong

Hough变换检测直线

cv2.HoughLine()

cv2.HoughLines()

'''

#载入图片img

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

#灰度化 --> gray

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

##中值滤波去除噪声 图像平滑

gray = cv2.medianBlur(gray, ksize=3)

#Canny边缘检测

edges = cv2.Canny(gray, 50, 120)

#最小直线长度, 更短的直线会被消除

minLineLength = 10

#最大线段间隙,一条线段的间隙长度大于这个值会被认为是两条分开的线段

maxLineGap = 5

##Hough 变换检测直线

lines = cv2.HoughLinesP(edges, 1, np.pi/180, 80, minLineLength, maxLineGap)

for i in range(len(lines)):

for x1, y1, x2, y2 in lines[i]:

#给定两点 在原始图片绘制线段

cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

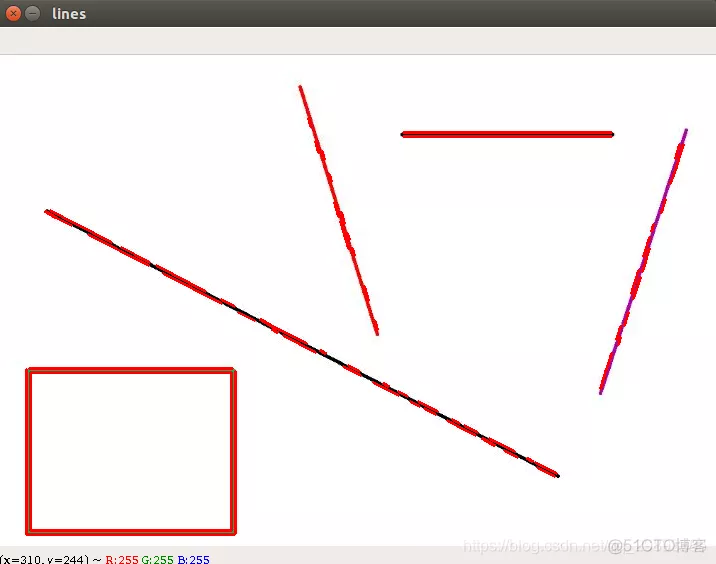
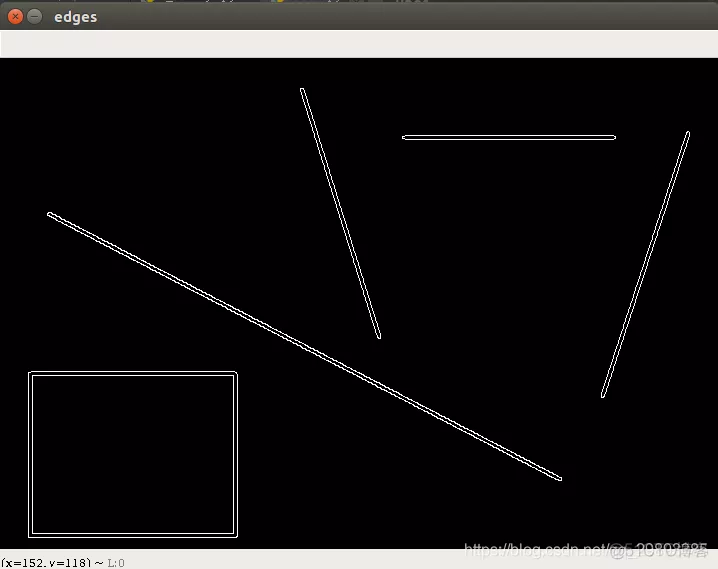
cv2.imshow('edges', edges)

cv2.imshow('lines', img)

cv2.waitKey()

cv2.destroyAllWindows()

实现结果如下:



* 除了HoughLinesP函数调用是这段代码的关键点以外，设置最小直线长度(更短的直线会被消除)和最大线段间隙也很重要，一条线段长度大于这个值会被视为两条分开的线段。

注意：HoughLinesP函数会接收一个由Candy边缘检测滤波器处理过的单通道二值图像。**不一定需要Candy滤波器，但是一个经过去噪并且只有边缘的图像当中Hough变换的输入会很不错**，因此使用Candy滤波器是一个普遍的惯例。

HoughLinesP函数参数如下：

* 需要处理的图像，需要是灰度图。
* 线段的几何表示rho和theta，一般分别取1和np.pi/180。
* 该函数返回一个numpy.array类型，形状为[num,1,4]，每一行对应一条直线，每条直线形状为(1,4)，这4个数值表示起始点和终止点坐标。
* 阈值。低于该阈值的直线会被忽略。Hough变换可以理解为投票箱和投票数之间的关系，每一个投票箱代表一个直线，投票数达到阈值的直线会被保留，其他的会被删除。
* 最小直线长度。
* 最大线段间隙。
* 2、圆检测

OpenCV的[HoughCircles](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines#houghcircles)函数可用来检测圆，其主要是利用霍尔变换在图像中寻找圆。我们知道，一个圆形的表达式为(x-x\_center)2+(y-y\_center)2=r2，一个圆环的确定需要三个参数，那么霍尔变换的累加器必须是三维的，但是这样的计算效率很低，而opencv采用了霍夫梯度的方法，这里利用了边界的梯度信息。

首先对图像进行Candy边缘检测，对边缘中的每一个非0点，通过sobel算子进行计算局部梯度。那么计算得到的梯度方向，实际上就是圆切线的法线。三条法线即可确定一个圆心，同理在累加器中对圆心通过的法线进行累加，就得到可圆环的判定。

[cv2.HoughCircles(img,method,dp,minDist,circles,param1,param2,minRadius,maxRadius)](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghlines#houghcircles)函数的参数如下：

* img为输入图像，需要是灰度图。
* method为检测方法，常用cv2.HOUGH\_GRADIENT
* 下面是一个例子：
* dp为检测内侧圆心的累加器图像的分辨率于输入图像之比的倒数，如dp=1，累加器和输入图像具有相同的分辨率，如果dp=2.累加器便有输入图像一半那么大的宽度和高度。
* minDist表示两个圆之间圆心的最小距离。
* param1有默认值100，它是method设置的检测方法对应的参数，对当前唯一的方法霍夫梯度法cv2.HOUGH\_GRADIENT，它表示传递给Candy边缘检测算子的高阈值，而低阈值为高阈值的一半。
* param2有默认值100，它是method设置的检测方法对应的参数，对当前唯一的方法霍夫梯度法cv2.HOUGH\_GRADIENT,它表示在检测阶段圆心的累加器阈值，它越小，就越可以检测到更多根本不存在的圆，而它越大的话，能通过检测的圆就更接近完美的圆形了。
* minRadius有默认值0，圆半径的最小值。
* maxRadius有默认值0，圆半径的最大值。
* 下面是一个检测例子:

#-\*- coding:utf-8 -\*-

import cv2

import numpy as np

'''

created on Tues jan 14:37:10 2018

@author:ren\_dong

Hough变换检测圆

cv2.HoughCricles()

'''

#载入图片img

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

#灰度化处理 --> gray image

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

##中值滤波 图像平滑

gray = cv2.medianBlur(gray, 7)

##Hough变换检测圆

circles = cv2.HoughCircles(gray, cv2.HOUGH\_GRADIENT, 1, 200,

param1=200, param2=20, minRadius= 0, maxRadius=0 )

circles = np.uint16(np.around(circles))

##绘制圆

for i in circles[0,:]:

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

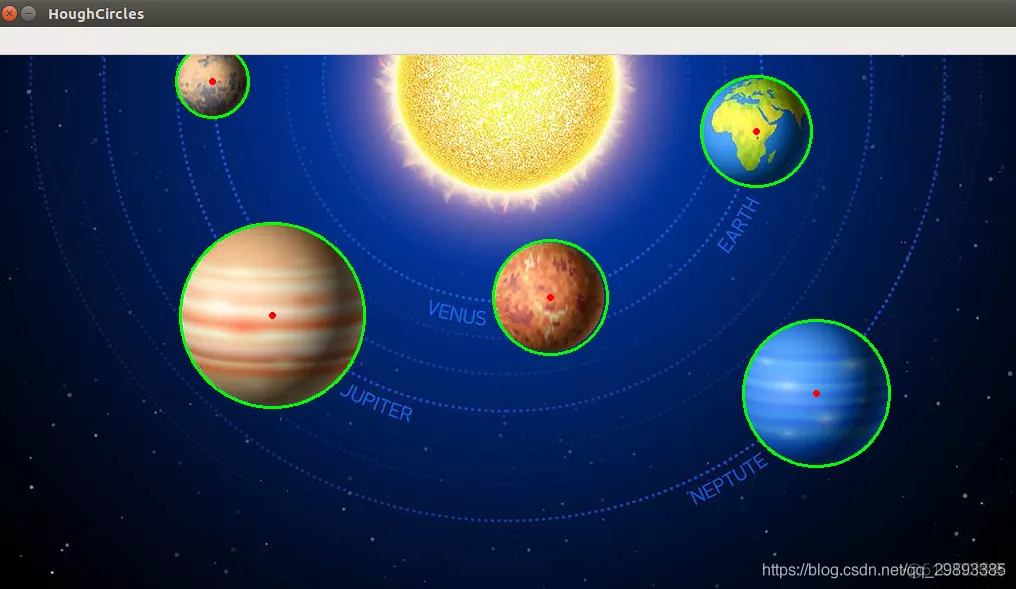
cv2.circle(img, (i[0], i[1]), 2, (0, 0, 255), 2)

cv2.imshow('HoughCircles',img)

cv2.waitKey()

cv2.destroyAllWindows()

实现效果如下:



* 3、检测其他形状

Hough变换能检测的形状仅限于圆，但是前面曾提到过检测任何形状的方法，特别是用approxPloyDP函数来检测。该函数提供多边形的近似，所以如果你的图像有多边形，再结合cv2.findContous函数和cv2.approxPloyDP函数，就可以相当准确的检测出来。

参考文章: javascript:void(0)

个人github主页:<https://github.com/RenDong3/OpenCV_Notes>

opencv 官方文档:<https://docs.opencv.org/2.4/index.html>