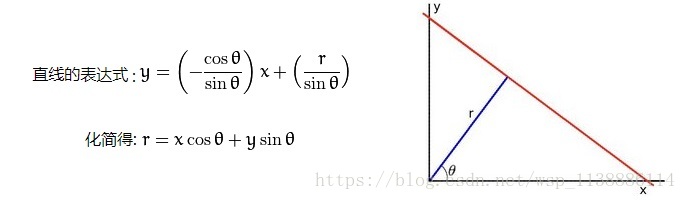
**OpenCV-Python 霍夫变换 检测直线，圆形**

**一、直线表示**

一条直线在图像二维空间可由两个变量表示

* 在 笛卡尔坐标系: 可由参数: (m , b m,b*m*,*b*) 斜率和截距表示.
* 在 极坐标系: 可由参数: (r , θ r,\theta*r*,*θ*) 极径和极角表示
* 对于霍夫变换, 我们将用 极坐标系 来表示直线. 因此, 直线的表达式可为下图：  
  
* 一般来说对于点 (x 0 , y 0 x\_{0}, y\_{0}*x*0​,*y*0​), 我们可以将通过这个点的一族直线统一定义为:  
  r θ = x 0 ⋅ c o s θ + y 0 ⋅ c o s θ r\_{\theta} = x\_0 ·cos\theta + y\_0 ·cos\theta*rθ*​=*x*0​⋅*cosθ*+*y*0​⋅*cosθ*  
  这就意味着每一对 (r θ , θ r\_{\theta},\theta*rθ*​,*θ*) 代表一条通过点 (x 0 , y 0 x\_{0}, y\_{0}*x*0​,*y*0​) 的直线.

**二、霍夫直线检测**

Hough变换是经典的检测直线的算法。  
用来检测图像中的直线，也可以用来检测图像中简单的结构。

OpenCV的中用函数 HoughLines(标准) 和 HoughLinesP(基于统计) 来检测图像中的直线.

* 基本的版本是cv2.HoughLines。其输入一幅含有点集的二值图（由非0像素表示），其中一些点互相联系组成直线。通常这是通过如Canny算子获得的一幅边缘图像。
* cv2.HoughLines()输出的是[float, float]形式的ndarray，  
  检测到的线(ρ , θ)中浮点点值的参数。
* 下面的例子首先使用Canny算子获得图像边缘，然后使用Hough变换检测直线。其中HoughLines函数的参数3和4对应直线搜索的步长。  
  在本例中：函数将通过步长为1的半径和步长为π/180的角来搜索所有可能的直线。最后一个参数是经过某一点曲线的数量的阈值，超过这个阈值，就表示这个交点所代表的参数对(ρ , θ)在原图像中为一条直线。

"""

cv2.HoughLines()

dst: 输出图像. 它应该是个灰度图 (但事实上是个二值化图)

lines: 储存着检测到的直线的参数对 (r,\theta) 的容器

rho : 参数极径 r 以像素值为单位的分辨率. 我们使用 1 像素.

theta: 参数极角 \theta 以弧度为单位的分辨率. 我们使用 1度 (即CV\_PI/180)

threshold: 设置阈值： 一条直线所需最少的的曲线交点

srn and stn: 参数默认为0

cv2.HoughLinesP(dst, lines, 1, CV\_PI/180, 50, 50, 10 )

dst: 输出图像. 它应该是个灰度图 (但事实上是个二值化图)

lines: 储存着检测到的直线的参数对 (x\_{start}, y\_{start}, x\_{end}, y\_{end}) 的容器

rho : 参数极径 r 以像素值为单位的分辨率. 我们使用 1 像素.

theta: 参数极角 \theta 以弧度为单位的分辨率. 我们使用 1度 (即CV\_PI/180)

threshold: 设置阈值： 一条直线所需最少的的曲线交点。超过设定阈值才被检测出线段，值越大，基本上意味着检出的线段越长，检出的线段个数越少。

minLinLength: 能组成一条直线的最少点的数量. 点数量不足的直线将被抛弃.

maxLineGap: 能被认为在一条直线上的两点的最大距离。

"""

import cv2

import numpy as np

original\_img= cv2.imread("jianzhu.png", 0)

img = cv2.resize(original\_img,None,fx=0.8, fy=0.8,

interpolation = cv2.INTER\_CUBIC)

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

edges = cv2.Canny(img, 50, 150, apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,118) #这里对最后一个参数使用了经验型的值

result = img.copy()

for line in lines:

rho = line[0][0] #第一个元素是距离rho

theta= line[0][1] #第二个元素是角度theta

print (rho)

print (theta)

if (theta < (np.pi/4. )) or (theta > (3.\*np.pi/4.0)): #垂直直线

pt1 = (int(rho/np.cos(theta)),0) #该直线与第一行的交点

#该直线与最后一行的焦点

pt2 = (int((rho-result.shape[0]\*np.sin(theta))/np.cos(theta)),result.shape[0])

cv2.line( result, pt1, pt2, (255)) # 绘制一条白线

else: #水平直线

pt1 = (0,int(rho/np.sin(theta))) # 该直线与第一列的交点

#该直线与最后一列的交点

pt2 = (result.shape[1], int((rho-result.shape[1]\*np.cos(theta))/np.sin(theta)))

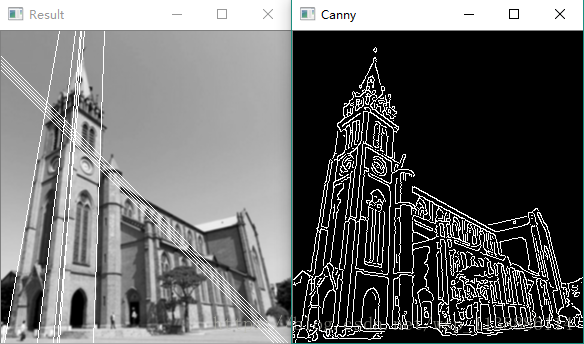
cv2.line(result, pt1, pt2, (255), 1) # 绘制一条直线

cv2.imshow('Canny', edges )

cv2.imshow('Result', result)

cv2.waitKey(0)

cv2.destroyAllWindows()



import cv2

import numpy as np

img = cv2.imread("jianzhu.png")

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

edges = cv2.Canny(img, 50, 150, apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,118)

result = img.copy()

#经验参数

minLineLength = 200

maxLineGap = 15

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

for x1,y1,x2,y2 in lines[0]:

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

cv2.imshow('Result', img)

cv2.waitKey(0)

cv2.destroyAllWindows()



**二、霍夫圆形检测**

**cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]) → circles**

参数说明:

image- 8位，单通道， **灰度**输入图像。  
circles- 找到的圆的输出向量。每个向量被编码为3元素的浮点向量 （x，y，半径）。  
circle\_storage - 在C函数中，这是一个将包含找到的圆的输出序列的内存存储。  
method- 使用检测方法。目前，唯一实现的方法是 CV\_HOUGH\_GRADIENT，基本上是 21HT，在[Yuen90]中有描述 。  
dp - 累加器分辨率与图像分辨率的反比。例如，如果 dp = 1，则累加器具有与输入图像相同的分辨率。如果 dp = 2，则累加器的宽度和高度都是一半。  
minDist -检测到的 **圆的中心之间的最小距离**。如果参数太小，除了真正的参数外，可能会错误地检测到多个邻居圈。如果太大，可能会错过一些圈子。  
param1 - 第一个方法特定的参数。在CV\_HOUGH\_GRADIENT的情况下， 两个传递给Canny（）边缘检测器的阈值较高（较小的两个小于两倍）。  
param2 - 第二种方法参数。在CV\_HOUGH\_GRADIENT的情况下  
，它是检测阶段的圆心的累加器阈值。越小，可能会检测到越多的虚假圈子。首先返回对应于较大累加器值的圈子。  
minRadius -最小圆半径。  
maxRadius - 最大圆半径。

我测试了一些图片，发现此方法只能检测图片背景干净一点的图片，我本来想检测印章的，可是纸张上有许多字，干扰严重，想了许多的办法（使用各种滤波等），但是效果还是不好，以下代码也是借鉴别人的。仅供参考。

import cv2

import numpy as np

def detect\_circle\_demo(image):

# dst = cv2.bilateralFilter(src=image, d=0, sigmaColor=100, sigmaSpace=5) # 高斯双边滤波(慢)

dst = cv2.pyrMeanShiftFiltering(image, 10, 100) # 均值偏移滤波（稍微快）

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

cv2.imshow("adapt\_image", dst)

circle = cv2.HoughCircles(dst, cv2.HOUGH\_GRADIENT, 1, 200, param1=50, param2=30, minRadius=50, maxRadius=300)

if not circle is None:

circle = np.uint16(np.around(circle))

print(circle)

for i in circle[0, :]:

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

cv2.imshow("circle", image)

if \_\_name\_\_ == "\_\_main\_\_":

src = cv2.imread("C:\\Users\\xxxx\\Desktop\\piaoju\\201920100013253001\_30302\_01.jpg")

src = cv2.resize(src, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_CUBIC)

detect\_circle\_demo(src)

cv2.waitKey(0)

cv2.destroyAllWindows()

直接查找圆形

import cv2

src = cv2.imread('C:\\Users\\SongpingWang\\Desktop\\piaoju\\xingqiu.png')

cv2.imshow('src\_img',src)

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

# 输出图像大小，方便根据图像大小调节minRadius和maxRadius

# 演示 minDist 的值对检测效果的影响

minDists = [100,125,150]

imgcopy = [src.copy(),src.copy(),src.copy()]

for minDist,imgcopy in zip(minDists,imgcopy):

circles= cv2.HoughCircles(gray,cv2.HOUGH\_GRADIENT,dp=1,minDist=minDist,param1=100,param2=30,minRadius=20,maxRadius=300)

print('circles',circles) # 查看返回值

print('len(circles[0])',len(circles[0])) # 输出检测到圆的个数

print('-------------------------------------')

for circle in circles[0]:

x=int(circle[0]) # 坐标行列

y=int(circle[1])

r=int(circle[2]) # 半径

img=cv2.circle(imgcopy,(x,y),r,(0,0,255),2) # 在原图用指定颜色标记出圆的位置

cv2.imshow('circle\_img\_'+str(minDist),img) # 显示新图像

cv2.waitKey(0)

cv2.destroyAllWindows()

'''

circles输出如下：

circles [[[ 89.5 279.5 84.5]

[328.5 277.5 76.5]

[305.5 100.5 86.2]

[101.5 95.5 87.7]

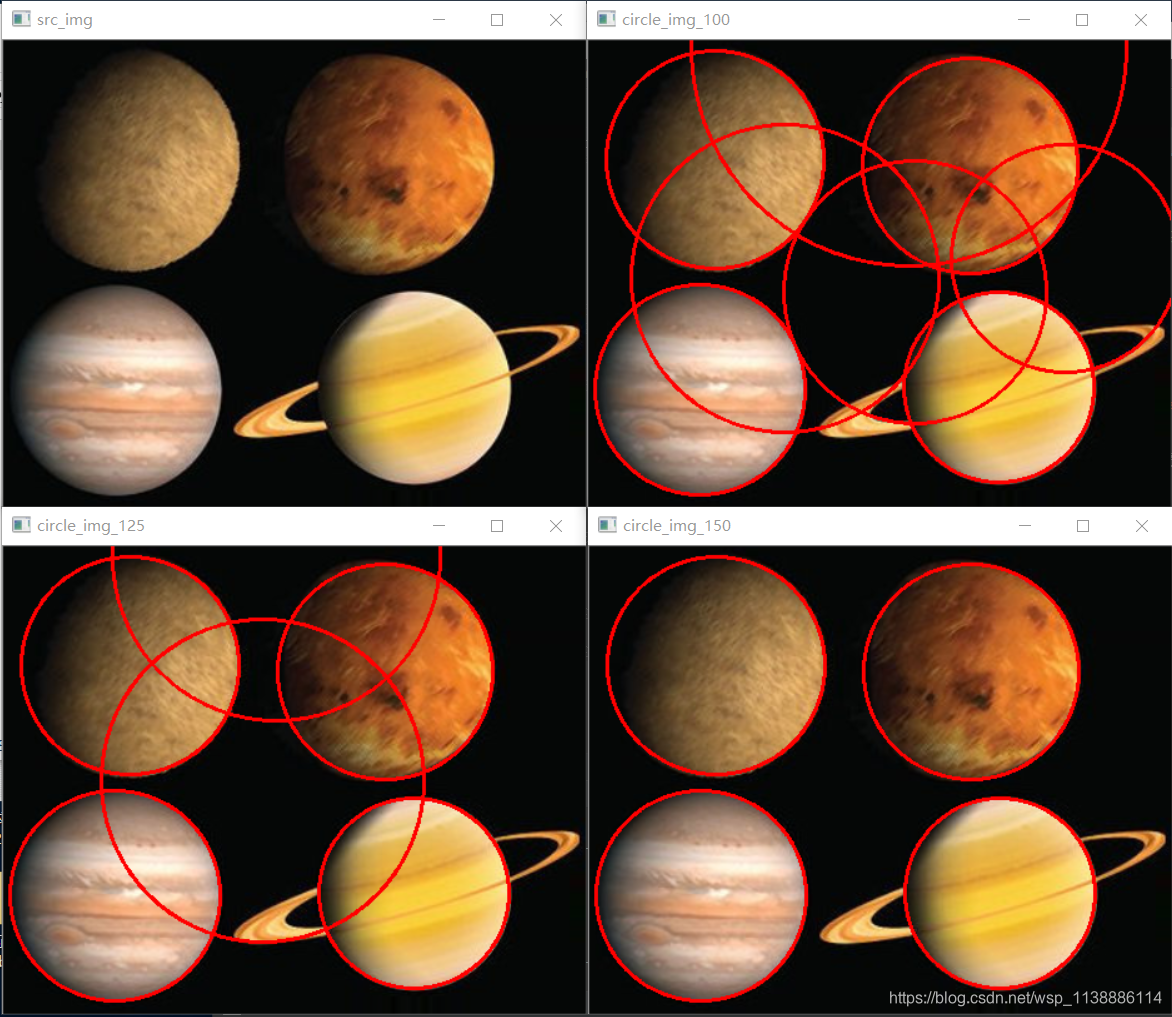
[157.5 190.5 123.4]

[261.5 201.5 105.8]

[256.5 6.5 174.8]

[381.5 174.5 91. ]]]

'''

  
从上图也能得出，此图的背景还算干净，若是参数稍微有所差异，检测效果天壤之别，我这里还只是调节了一个参数，最小半径与最大半径还是人为指定的。