实验一 几何目标提取

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名：陈志鸿** | **学号：2020281024** | **日期：2022-03-14** | **截止日期：Next Class** |

**一、实验目的**

1. 掌握Opencv图像的几何变换
2. 掌握Opencv进行图像几何目标的检测

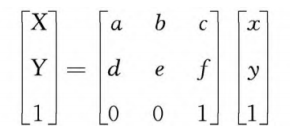
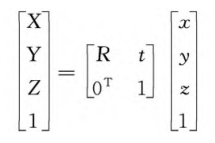
**二、实验报告要求**

1. 实验报告中，**按题目顺序给出实验内容对应的代码及相应结果**，**体现实验步骤和方法**。要求报告整洁，程序清晰，代码简捷，有必要的注释
2. **报告中的图片**要求清晰可见。 如果是屏幕截图，**请不要截取有效结果之外的无关区域**。
3. **报告中的代码**，一般情况下紧随内容。如果代码太长，建议按题目序号，附在报告末尾。

**三、实验内容及要求**

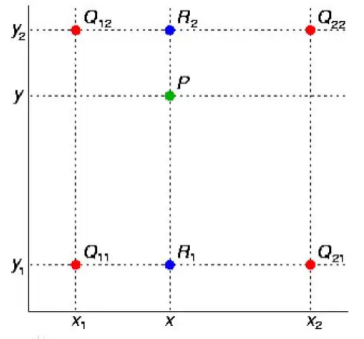
**一、概念题**

1. 用矩阵形式，如何表示仿射变换和透视变换？

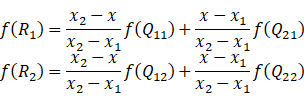
仿射变换：透视变换：

2. 简述前向和后向计算法的区别？

前向计算法存在大量的重复计算，浪费了资源；而后向计算法只需要计算一次，节省资源。

3. 已知函数f 在Q11, Q12, Q21, Q22四个点的值, 如右图所示。

写出双线性插值公式，求得P点的值





1. 解释Hough变换检测直线和圆的算法原理。

将测量空间的一点变换到参量空间的一条曲线或曲面，而具有同一参量特征的点变换后在参量空间中相交，通过判断交点处的积累程度来完成特征曲线的检测。基于参量性质的不同，Hough变换可以检测直线、圆、 椭圆、双曲线等。如直线检测积累的是角度和垂线，圆检测累积的是圆心和半径。

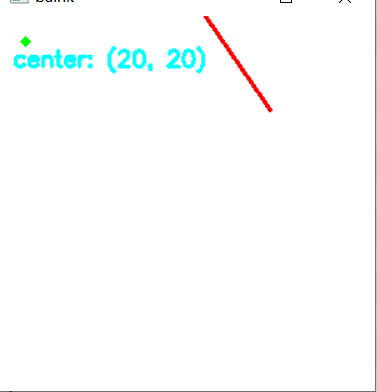
5. 广义Hough变换如何实现任意目标的检测？简述其原理。

广义霍夫变换在霍夫变换的基础上根据模板匹配的原理进行了调整。广义霍夫变换不要求能够给出需要检测的形状的解析式，它可以检测任意给定的形状。也即他通过统计边缘的梯度方向和边缘相对于参考点的偏移量形成R-Table。根据R-Table进行投票。

二、**编程题：请在关键代码处给出注释**

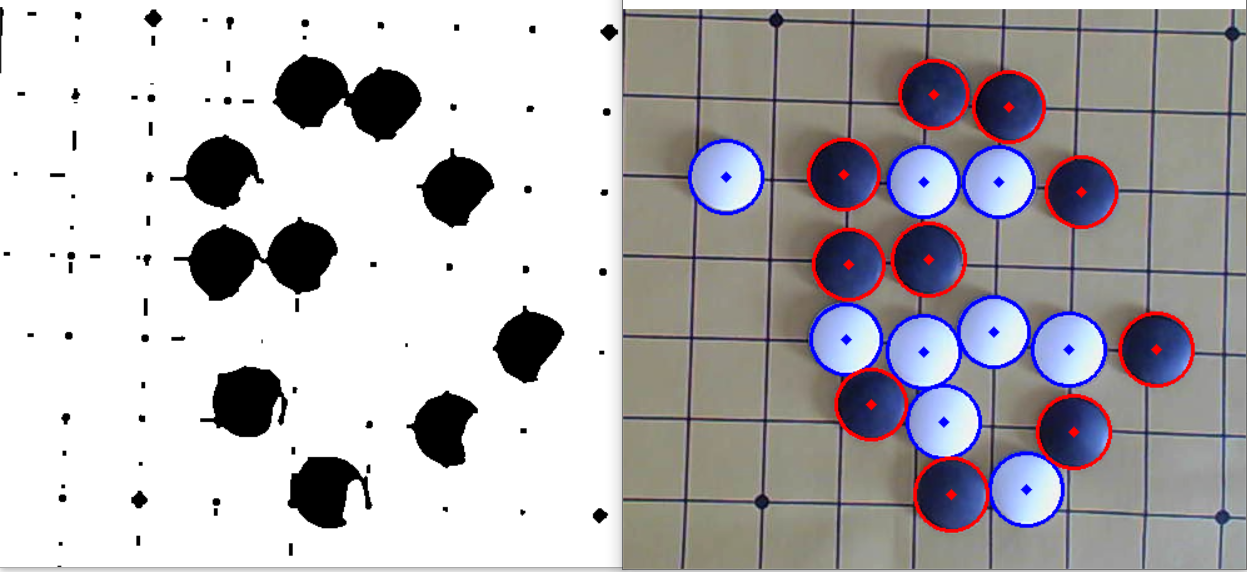
3.1 写出代码，在空白图中画出一条直线，具有如下特征：以(20, 20) 为中心，与水平轴的夹角为60度，长度为100个像素。

1. **def** line() :
2. blank = np.full((300, 300, 3), 255, dtype = np.uint8)
3. # cv.line(blank, (150 ,150), (250, 150), (0, 0, 255), thickness = 2)
4. #获得旋转矩阵
5. RotatedMatirx = cv.getRotationMatrix2D((20, 20), -60, 1)
6. pts = np.array([[150, 150, 1], [250, 150, 1]])
7. #获得旋转后对应的点
8. ptsRotated = np.multiply(pts, RotatedMatirx)
9. cv.line(blank, (int(ptsRotated[0, 0]), int(ptsRotated[0, 1])),
10. (int(ptsRotated[1, 0]), int(ptsRotated[1, 1])),
11. (0, 0, 255), thickness = 2)
12. cv.circle(blank, (20, 20), radius = 2,
13. color = (0, 255, 0), thickness = 3)
14. cv.putText(blank, f'center: (20, 20)', (10, 40),
15. fontFace = cv.FONT\_HERSHEY\_SIMPLEX,
16. fontScale = 0.6, color = (255, 255, 0), thickness = 2)
17. cv.imshow('balnk', blank)
18. cv.waitKey(0)



3.2 写出代码，检测出图像weiqi.png中黑白棋子，用不同的颜色标出棋子中心的位置。

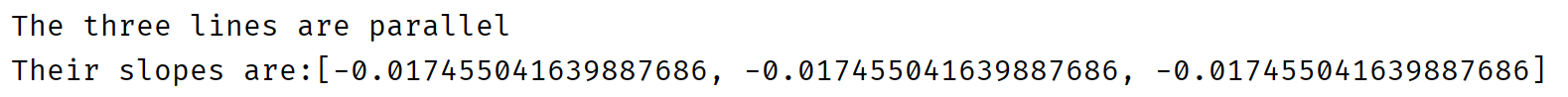
1. **def** circle\_detect() :
2. img = cv.imread('images/weiqi.png')
3. cimg = copy.deepcopy(img)
4. img\_gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)
5. # img\_gray = cv.GaussianBlur(img\_gray, (5, 5), 0)
6. #降噪
7. img\_gray = cv.medianBlur(img\_gray, 5)
8. circles = cv.HoughCircles(img\_gray, cv.HOUGH\_GRADIENT, 1, 10, param1 = 120, param2 = 50).squeeze()
10. **for** circle **in** circles :
11. #转成np.uint16数据类型扩大数据范围
12. circle = np.uint16(np.around(circle))
13. #做二值化处理，白棋为白，黑棋为黑，若圆心的值为255则为白棋，否则为黑棋
14. \_, img\_thresh = cv.threshold(img\_gray, 80, 255, cv.THRESH\_BINARY)
15. #注意x和y与w和h的对应
16. **if** img\_thresh[circle[1], circle[0]] == 255 :
17. **print**(f'chess is black!')
18. # 画出检测到的圆
19. cv.circle(img, center=(circle[0], circle[1]), radius=circle[2],
20. color=(255, 0, 0), thickness=2)
21. # 画出圆心
22. cv.circle(img, center=(circle[0], circle[1]), radius=2,
23. color=(255, 0, 0), thickness=3)
24. **else** :
25. **print**('chess is white!')
26. # 画出检测到的圆
27. cv.circle(img, center=(circle[0], circle[1]), radius=circle[2],
28. color=(0, 0, 255), thickness=2)
29. # 画出圆心
30. cv.circle(img, center=(circle[0], circle[1]), radius=2,
31. color=(0, 0, 255), thickness=3)
32. cv.imshow('canny', img\_thresh)
33. # img\_concatenate = np.concatenate((cimg, img), axis = 1)
34. # cv.imwrite('img\_con.png', img\_concatenate)
35. cv.imshow('img', img)
36. cv.waitKey(0)



3.3 写出代码，检测并判断下图中箭头所示的三条直线是否平行？请自行设定合理的误差范围， 并给出实验结果。



1. **def** line\_detect() :
2. img = cv.imread('images/chemical\_tube.png')
3. img\_gray = cv.cvtColor(img, cv.COLOR\_RGB2GRAY)
4. img\_gaussion = cv.GaussianBlur(img\_gray, (5, 5,), 0)
5. img\_canny = cv.Canny(img\_gaussion, 70, 120)
7. x1\_, x2\_, y1\_, y2\_= [], [], [], []
8. lines = cv.HoughLines(img\_canny, 1, np.pi / 180, 300)
9. **for** line **in** lines:
10. rho, theta = line[0][0], line[0][1]
11. a = np.cos(theta)
12. b = np.sin(theta)
13. x = rho \* a
14. y = rho \* b
15. x1, y1 = x + 1000 \* b, y - 1000 \* a
16. x2, y2 = x - 1000 \* b, y + 1000 \* a
17. cv.line(img, (int(x1), int(y1)), (int(x2), int(y2)),
18. (255, 0, 0), thickness=2)
19. #收集每条直线的两个点
20. x1\_.append(x1)
21. x2\_.append(x2)
22. y1\_.append(y1)
23. y2\_.append(y2)
24. #因为目标的三条直线自上而下的第二到第四条，也就是y在第二到第四的位置
25. y\_arr = np.array(y1\_)
26. #获得下标
27. y\_target\_index = np.argsort(y\_arr)[1 : 4].tolist()
28. k = []
29. #计算三条直线的斜率
30. **for** index **in** y\_target\_index :
31. k.append((y2\_[index] - y1\_[index]) / (x2\_[index] - x1\_[index]))
32. # print(k)
33. #当他们的斜率之间差值小于1e-3时，则认为他们相互平行
34. **if** abs(k[0] - k[1]) < 1e-3 **and** abs(k[0] - k[2]) < 1e-3 **and** abs(k[1] - k[2]) < 1e-3:
35. **print**(f'The three lines are parallel\nTheir slopes are:{k}')
37. cv.imshow('gray', img)
38. cv.waitKey(0)



3.4 把图像中的圆环拉直，如图circle\_band.bmp所示。

1. 简述算法的原理和步骤

首先是对圆环进行圆检测，检测出外圆和内圆的圆心和半径，后以外圆的周长，圆环的宽度也即内圆和外圆的半径之差作为宽创建一个矩形，若无法检测内圆，可以粗略使用外圆半径的二分之一作为宽。从圆环的最外圈开始遍历，最外圈即对应着矩形中的第一行，宽度为多少，则矩形中就有多少行以圆心为中心构成极坐标系，则圆环上任意一点可以用rho和theta来表示  
再根据公式计算出该点在图上的真是坐标，把像素值赋给矩形框中对应的位置。最后要将矩形的数据类型转回np.uint8类型。

2. 写出Python代码，实现上述方法，并展示代码效果

1. **def** circle\_flatten() :
2. img = cv.imread('images/circle\_band.bmp')
3. img\_gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)
4. # img\_gray = cv.medianBlur(img\_gray, 3)
6. circles = cv.HoughCircles(img\_gray, cv.HOUGH\_GRADIENT, 1, 50, param1 = 170, param2 = 100).squeeze()
7. #获得检测到的所有圆的半径
8. circle\_radius = circles[ : , 2]
9. #获得最大半径的下标
10. radius\_biggest\_index = np.argsort(circle\_radius)[-1]
11. **print**(radius\_biggest\_index)
12. #做出最大圆
13. circle = np.uint16(np.around(circles[radius\_biggest\_index]))
14. cv.circle(img, (circle[0], circle[1]), radius = circle[2], color = (0, 0, 255), thickness = 5)
15. cv.circle(img, (circle[0], circle[1]), radius = 2, color = (255, 0, 0), thickness = 2)
17. #取展平后条形圆环的宽为最大半径的一半，而长取最大圆的周长
18. height = int(circle\_radius[radius\_biggest\_index] \* np.pi \* 2)
19. width = int(circle\_radius[radius\_biggest\_index] / 3)
20. rectangle = np.zeros([width, height])
21. **print**(rectangle.shape)
22. **print**(img\_gray.shape)
23. **for** row **in** range(width) :
24. **for** col **in** range(height) :
25. #转成极坐标系
26. theta = np.pi \* 2.0 / height \* (col + 1)
27. rho = circle\_radius[radius\_biggest\_index] - row - 1
28. #以圆心为原点，求得原来圆环对应的坐标
29. position\_x = int(circle[0] + rho \* np.cos(theta) + 0.5)
30. position\_y = int(circle[1] - rho \* np.sin(theta) + 0.5)
31. rectangle[row, col] = img\_gray[position\_y, position\_x]
32. #要转回np.uint8型数据，否则显示有问题
33. rectangle = np.uint8(rectangle)
34. cv.imshow('1', rectangle)
35. cv.imshow('img', img)
36. cv.waitKey(0)

效果：

