实验设计文档

一. 设计概述

本程序是一个钱币定位系统，能够检测出输入图像中各个钱币的边缘，并给出各个钱币的圆心坐标与半径。

在`usePackage.py`中，我调用了opencv库中的`cv.Canny`与`cv.HoughCircles`实现了钱币定位，在`canny.py`与`houghCircle.py`中，我自主实现了Canny与Hough算法，在实验测试图片上同样较准确地完成了定位功能。

1.1 算法流程

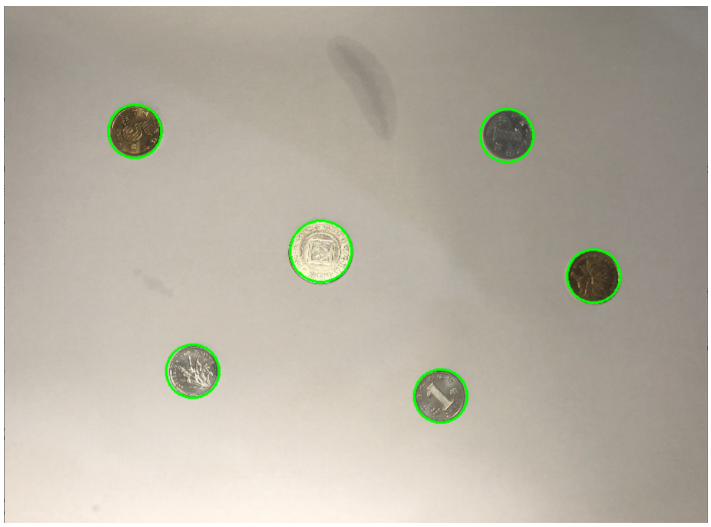
首先使用`opencv`读入图像，并转换为灰度图，然后将图片缩放为预设大小。

然后调用`Canny`类。首先将图片进行高斯模糊，然后求得图像的梯度幅值与梯度方向，之后对图像进行非最大化抑制将边缘细化，最后使用启发式双阈值算法连接弱边和强边，输出包含边缘的图像。

最后进行霍夫变换。输入canny算法处理后的边缘图片及梯度方向，在3维投票矩阵中，每一个边缘上的点都沿着梯度方向进行投票。选用合适的格子大小对投票进行计数，然后筛选出高于阈值的圆并进行非最大化抑制，只选出圆心相近投票数最多圆，记录下圆心及半径，最后画出拟合的圆并输出圆心与半径信息。

1.2 最终拟合结果

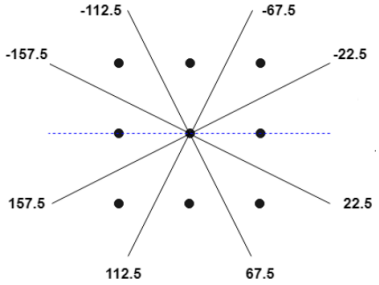
```
Circle center :(157, 152), Radius: 32
Circle center :(607, 157), Radius: 32
Circle center :(712, 327), Radius: 32
Circle center :(227, 442), Radius: 32
Circle center :(527, 472), Radius: 32
Circle center :(382, 297), Radius: 37
```



二. 核心函数说明

2.1 Canny算法

`Non_maximum_suppression`函数将图像进行非最大化抑制。它使用3*3的窗口在图像上滑动，并将梯度方向分成四个区间，将每一个梯度方向四舍五入到这四个区间之一，比较区间两端的幅值与中心点的幅值，若中心点的幅值最大则保留，否则移除。



这里，将梯度方向分为四个区间的做法存在一定误差，使用双线性插值的方法会更加精确。代码如下：

```
def Non_maximum_suppression(self):
    self.orientation[self.orientation >= 180] -= 180
    a, b = 0, 0

    for i in range(1, self.height - 1):
        for j in range(1, self.width - 1):
            if self.orientation[i][j] < 22.5 or self.orientation[i][j] >= 157.5:
                a, b = self.image[i][j + 1], self.image[i][j - 1]
            elif 22.5 <= self.orientation[i][j] < 67.5:
                a, b = self.image[i + 1][j + 1], self.image[i - 1][j - 1]
            elif 67.5 <= self.orientation[i][j] < 112.5:
                a, b = self.image[i + 1][j], self.image[i - 1][j]
            elif 112.5 <= self.orientation[i][j] < 157.5:
                a, b = self.image[i - 1][j + 1], self.image[i + 1][j - 1]

            if self.image[i][j] < a or self.image[i][j] < b:
                self.image[i][j] = 0
```

`Hysteresis_thresholding`函数是双阈值算法，它设定了两个阈值，将图像中的点分为三类：高于高阈值，低于低阈值，介于二者之间。然后使用3*3窗口在图像上进行滑动，若中心点是中等强度的点且周围8个像素存在高阈值点则保留，否则移除。这样就实现了使用低阈值点连接高阈值点的效果。

如果连接效果不好，可以调用多次双阈值算法，来实现更好的连接效果。

```
def Hysteresis_thresholding(self):
    thresholded = np.zeros((self.height, self.width))
    index = self.magnitude >= self.threshold1
    thresholded[index] = 1
    index = (self.threshold2 < self.magnitude) & (self.magnitude <
self.threshold1)
    thresholded[index] = -1

    for i in range(1, self.height - 1):
        for j in range(1, self.width - 1):
            if thresholded[i][j] == -1:
```

```
                if sum([thresholded[x][y] for x in range(i - 1, i + 2) for y in
range(j - 1, j + 2)]) > -1:
                    thresholded[i][j] = 1
                else:
                    thresholded[i][j] = 0

    self.image = np.uint8(thresholded) * 255
```

参数分析：

canny算法接收3个超参：kernal_size, threshold1, threshold2。kernal_size是高斯模糊的卷积核大小，作用是对图像进行平滑处理以便求得图像梯度，其取值越小图像的细节保留越多，取值越大图像越模糊。threshold1是双阈值算法中的高阈值，其取值越大丢失的边缘信息越多，反之保留信息越多且会保留更多杂边。threshold2是双阈值算法中的低阈值，其值越大连接效果越差，反之连接效果越好但是会保留更多杂边。

2.2 houghCircle算法

在霍夫变换算法中，我并将许多中间数据矩阵化，并使用了大量`numpy`库函数进行**并行计算**，极大提高了计算效率。

在三维投票矩阵中，对于每一个半径r，计算出每个边缘点相对梯度方向上的延伸点，并投票。

```
# 初始化3D投票矩阵
vote_matrix = np.zeros((maxr, h, w), dtype=np.int32)
# 每一个edge point下标及方向
edge_sub = np.argwhere(image)
edge_orient = np.array([orientation[i, j] for i, j in edge_sub])

# 并行计算加速
for r in tqdm(range(minr, maxr)):
    x = np.array([])
    y = np.array([])

    # 数组坐标[i,j]，opencv坐标为[j,i]
    x = np.append(x, edge_sub[:, 0] + r * np.sin(edge_orient))
    x = np.append(x, edge_sub[:, 0] - r * np.sin(edge_orient))
    y = np.append(y, edge_sub[:, 1] + r * np.cos(edge_orient))
    y = np.append(y, edge_sub[:, 1] - r * np.cos(edge_orient))

    # 将坐标拼接起来，以便投票
    hough_coor = np.dstack((x, y))
    hough_coor = hough_coor.reshape(-1, 2).astype(np.uint32)

    for i, j in hough_coor:
        if 0 <= i < h and 0 <= j < w:
            vote_matrix[r, i, j] += 1
```

然后使用一个小正方体在投票矩阵上滑动计票，在参数空间中计算累计结果的局部最大值，选出高于预设阈值的点。之后对这些点进一步筛选（**非最大化抑制**），舍弃掉那些圆心相近但是投票数不是最大的点，最后返回圆心与半径信息。

参数分析：

hough算法接收4个超参：threshold, minr, maxr, region。threshold越大提取出的符合条件的圆越少。minr、maxr则是限定的圆半径，他们之间相差越小计算时间花费越少。region是选取的计票格子大小，取值越大计算越快，但是相对定位不准确，但是取值越小会导致较难求焦（找到投票数最集中的区域），会增加算法的误判。

三. 遇到的问题

3.1 统一图片大小

上述分析的参数设置都必须参考图片的分辨率，若是在不同分辨率的图像上使用相同参数，效果不太好。因此为了得到更好的泛化效果，将图像在输入后先缩放到统一的预设大小，然后在进行之后的操作。

3.2 Debug问题

在编写霍夫变换的投票算法时，出现了很多bug，使用IDE调试很难找出错误根源，我将三维矩阵沿着r轴进行投影，并将结果输出为图像，可以更加直观地看到投票的分布。

例如利用下图，我找到了我的错误所在：opencv的坐标与numpy数组的坐标系是不一致的，计算角度、投票都应该是基于opencv的坐标，只是使用数组记录。由于opencv的坐标系特殊，并且很绕，导致了代码编写过程中许多bug，以后编程中还需更加注意。