

HW02 Line/Circle Extraction

3180101041 杨锐

软件开发说明

- @MacOS
- Python == 3.7.6
- Numpy == 1.8.1
- Opencv-python == 4.2.0.34

算法设计思路

对圆和直线的检测都是基于*hough transform*, 其基本的思想是在参数空间中进行投票, 之后在累加器空间中从候选对象中选择局部最大值, 而累加器空间则由特定形状的检测算法构造。下面具体分析直线和圆的检测。

Hough Line Transform

直线检测的关键在于, 对每一个点, 如何确定它的累加器空间?

我们知道直线的直角坐标系方程:

$$y = kx + b$$

对于每一个点 (x, y) , 我们可以得到一个 (k, b) 的方程确定通过这个点除平行坐标轴的所有直线。因此我们需要将它转换为极坐标下参数方程:

$$y = \left(-\frac{\cos \theta}{\sin \theta} \right) x + \left(\frac{r}{\sin \theta} \right)$$

简单变换:

$$r_\theta = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$$

因此我们便得到了直线检测的累加器空间, 下面是具体实现:

预处理

为了更好的检测效果, 我们需要做三步处理:

- 原图片转为灰度图
- 进行高斯/中值滤波
- 边缘检测

参数初始化

我们需要两个空间(数组),

- 一个二维数组是记录 (r, θ) 投票结果,
- 一个三维数组记录 (r, θ) 空间下的所有点。
- 除此以外还有 (r, θ) 对应的取值范围

投票

遍历, 对于每一个边缘点, 计算所有的 (r, θ) , 放入对应的累加器和点空间, 下面是初始化和投票的具体算法:

```
# Hough Space initialize
height = edges.shape[0]
width = edges.shape[1]
rMax = int(math.hypot(height, width))
thetaMax = 360
points = [[[[] for k in range(thetaMax)] for theta in range(rMax)] for r in range(rMax)]
# vote
for x in range(width):
    for y in range(height):
        if edges[y][x] == 0:
            continue
        for theta in range(0, thetaMax-1):
            r = int(x*math.cos(theta)+y*math.sin(theta))
            if rMin < r < rMax:
                hough_space[r][theta] += 1
                points[r][theta].append((x, y))
```

选择候选对象

对于每一个候选对象, 我们判读是否达到阈值要求, 然后判断该对象是否是 (r, θ) 空间中最边缘的点 (为了更好的绘制效果, 我们需要确定该直线的范围, 而不需要中间的点)。如果是, 将 (r, θ) 转换为 (k, b) , 绘制该直线。

```

# aggregate threshold

def get_threshold(ough_space, width, height, x, y, space):
    threshold = ough_space[x][y]
    if x - space > 0 and x + space < width and y - space > 0 and y + space < height:
        tmp = []
        for i in range(x-space, x+space):
            tmp.append(max(ough_space[i][y-space:y+space]))
        threshold = max(tmp)
    return threshold

# find max

for r in range(rMax):
    for theta in range(thetaMax):
        if ough_space[r][theta] >= threshold:
            if ough_space[r][theta] == get_threshold(ough_space, thetaMax, rMax, r,
theta, 20):
                k = (-math.cos(theta)/math.sin(theta))
                b = (r/math.sin(theta))
                points[r][theta].sort()
                x0 = points[r][theta][0][0]
                x1 = points[r][theta][-1][0]
                y0 = int(x0*k+b)
                y1 = int(x1*k+b)
                cv2.line(cdst, (x0, y0), (x1, y1),
(0, 0, 255), thickness=2)

```

至此，我们便完成了直线的检测。

Hough Circle Transform

圆形检测和直线检测基本思路类似，不过累加器空间需要三个参数确定：

$$C : (x_{center}, y_{center}, r)$$

三维空间意味着内存消耗更多、运算速度更慢。因此我们需要在此基础上引入Hough gradient method.

第一步：估计圆心

1. 遍历 Canny 边缘二值图中的所有非零像素点，沿着梯度方向（切线的垂直方向）画线，将线段经过的所有累加器中的点 $(a,b) \rightarrow 1$ 。
2. 统计排序累加器，得到可能的圆心 $(N(a,b))$ 越大，越有可能是圆心）

第二步：估计半径

0. 对于每一个圆心候选者 (a, b)
1. 计算 Canny 图中所有非 0 点距离圆心的距离。
2. 距离从小到大排序，根据阈值，选取合适的可能半径（
3. 初始化半径空间 $r, N(r)=0$ 。
4. 遍历 Canny 图中的非 0 点， $N(\text{距离}) += 1$ 。
5. 统计得到可能的半径值 $(N(r))$ 越大，说明这个距离值出现次数越多，越有可能是半径值）

下面是具体代码实现

预处理

同直线一样

参数初始化

- 计算x方向和y方向的梯度
- 一个二维数组统计圆心(a, b)的投票结果
- 一个三维数组统计对应圆心(a, b)下所有候选者

估计圆心

```
# loop to vote
for x in range(width):
    for y in range(height):
        if edges[y][x] == 0: continue
        if sobel_dx[y][x] != 0:
            # compute theta
            tan_theta = sobel_dy[y][x] / sobel_dx[y][x]
            # circle center
            for a in range(0, width):
                b = int(a * tan_theta - x * tan_theta + y)
                if 0 < b < height:
                    if minRadius < math.hypot(a-x, b-y) < maxRadius:
                        hough[a][b] += 1
                        accumulator[a][b].append((x, y))

        if sobel_dx[y][x] == 0 and sobel_dy[y][x] != 0:
            if sobel_dy[y][x] > 0:
                for b in range(y+minRadius, y+maxRadius):
                    # circle center in y
                    if 0 <= b < height:
                        # vote
                        hough[x][b] += 1
                        accumulator[x][b].append((x, y))

            else:
                for b in range(y-maxRadius, y-minRadius):
                    if 0 <= b < height:
                        # vote
                        hough[x][b] += 1
                        accumulator[x][b].append((x, y))
```

估计半径

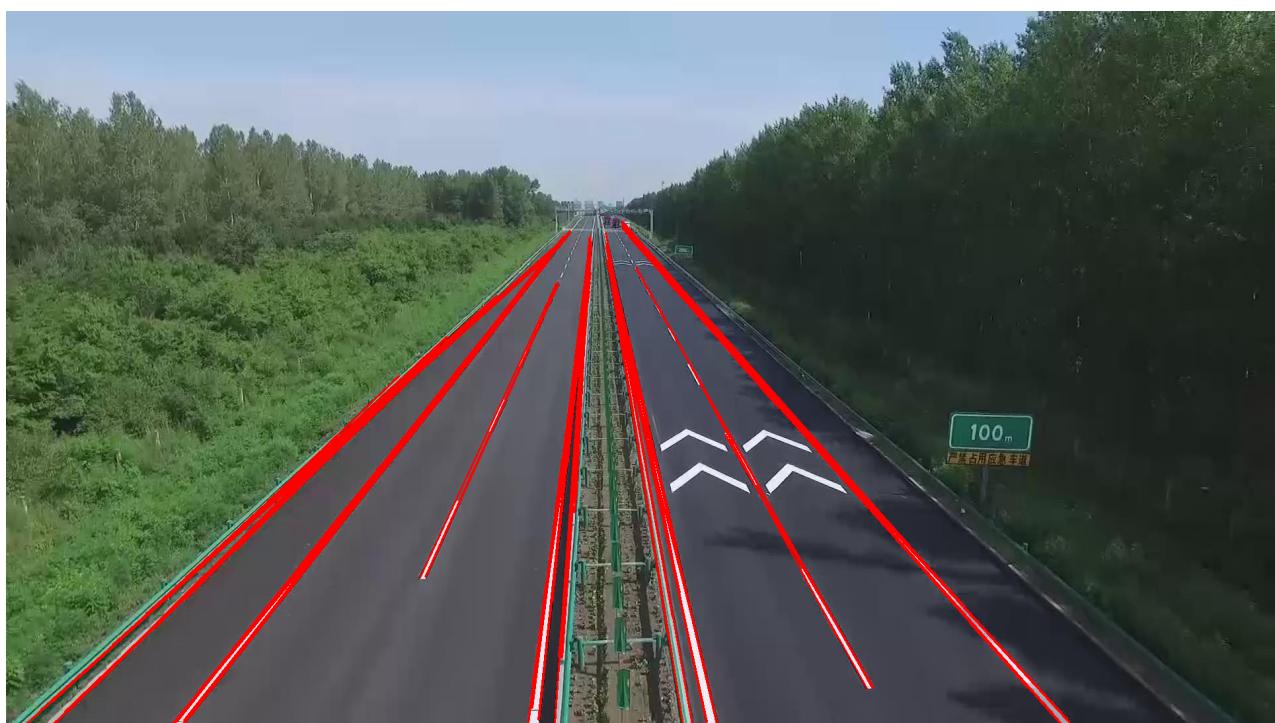
```
for a in range(0, width):
    for b in range(0, height):
        if hough[a][b] > threshold:
            maxth = get_threshold(hough, width, height, a, b, 50)
            if hough[a][b] == maxth:
                list = []
                area_count = [0, 0, 0, 0]
                for i in range(0, len(accumulator[a][b])):
                    dx = accumulator[a][b][i][0] - a
                    dy = accumulator[a][b][i][1] - b
                    radius = int(math.hypot(dx, dy))
                    # get the points of four spaces in one circle
                    if dx >= 0 and dy >= 0:
                        area_count[0] += 1
                    if dx >= 0 and dy < 0:
                        area_count[1] += 1
                    if dx < 0 and dy >= 0:
                        area_count[2] += 1
                    if dx < 0 and dy < 0:
                        area_count[3] += 1
                list.append(radius)

                if(len(list) > 0) and area_count[0] != 0 and area_count[1] != 0 and
area_count[2] != 0 and area_count[3] != 0 and max(area_count)-min(area_count) < 10:
                    radius = list[len(list)//2]
                    if minRadius < radius < maxRadius:
                        cv2.circle(img, center=(a, b), radius=radius,
                                   color=(0, 0, 255), thickness=2)
                    for k in accumulator[a][b]:
                        cv2.circle(img, k, 4, (255, 0, 0), 1, 8, 0)
```

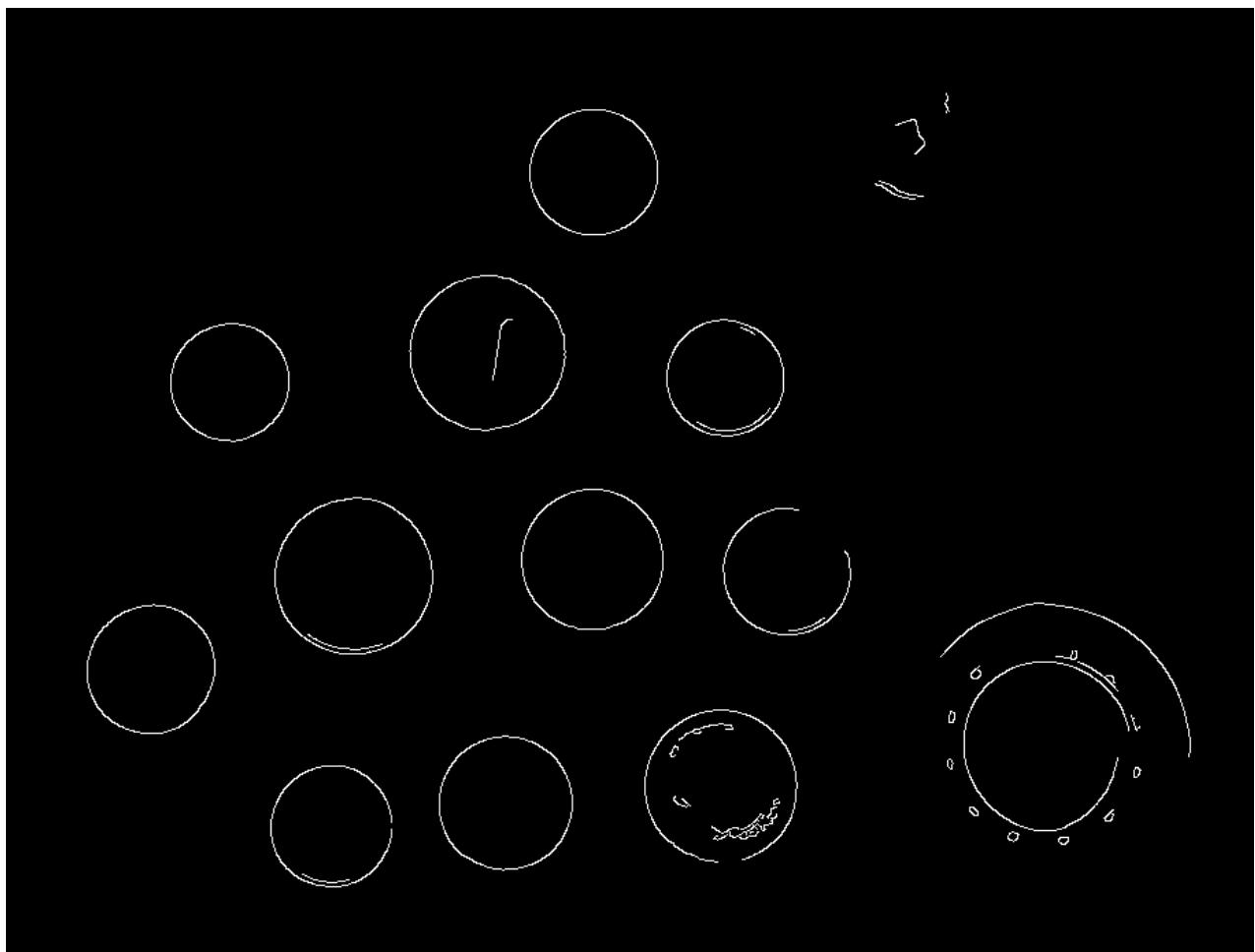
至此，我们便完成了圆形检测。

实验结果分析

直线检测



圆检测







编程体会

- 理解了*hough line transform*算法实现步骤
- 理解了*hough circle transform*算法实现步骤

个人照片

