浙江大学

本科实验报告

姓名：	沈骏一
学院：	控制科学与工程学院
专业：	自动化（控制）
学号：	3200100259
指导教师：	姜伟

2023年3月19日

DIP 图像傅里叶变换作业

1. 根据本章介绍的轮廓跟踪算法，编写一个标记图像的轮廓跟踪程序。

步骤1 首先从上到下、从左到右顺序扫描 图像，寻找第一个目标点作为边界跟踪的起始点，记为A。A点一定是最左角上的边界点，其相邻的边界点只可能出现在它的左下、下、右下、右四个邻点中。定义一个搜索方向变量dir，用于记录从当前边界点搜索下一个相邻边界点时所用的搜索方向码。

dir初始化为：

1. 对基于4方向的轮廓跟踪，dir=3，即从方向3开始搜索与A相邻的下一个边界点。
2. 对基于8方向的轮廓跟踪，dir=5，即从方向5开始搜索与A相邻的下一个边界点。

如果当前搜索方向dir上的邻点不是边界点，则依次使搜索方向逆时针旋转一个方向，更新dir，直到搜索到一个边界点为止。

如果所有方向都未找到相邻的边界点，则该点是一个孤立点。

dir的更新用公式可表示为：对基于8方向的轮廓跟踪有dir=(dir+1) mod 8，对基于4方向的轮廓跟踪有dir=(dir+1) mod 4。

步骤2 把上一次搜索到的边界点作为当前边界点，在其3×3邻域内按逆时针方向搜索新的边界点，它的起始搜索方向设定如下：

1. 对基于4方向的轮廓跟踪，使dir=(dir + 3) mod 4，即将上一个边界点到当前边界点的搜索方向dir顺时针旋转一个方向；
2. 对基于8方向的轮廓跟踪，若上次搜索到边界点的方向dir为奇数，则使dir=(dir + 6) mod 8，即将上次的搜索方向顺时针旋转两个方向；若dir为偶数，则使dir=(dir + 7) mod 8，即将上次的搜索方向顺时针旋转一个方向。

如果起始搜索方向没有找到边界点，则依次使搜索方向逆时针旋转一个方向，更新dir，直到 搜索到一个 新的边界点为止。

步骤3 如果搜索到的边界点就是第一个边界点A，则停止搜索，结束跟踪，否则重复步骤2继续搜索。

由依次搜索到的边界点系列就构成了被跟踪的边界。

步骤1中所采用的准则称为“探测准则”，其作用是找出第一个边界点；步骤2中所采用的准则称为“跟踪准则”，其作用是找出所有边界点。

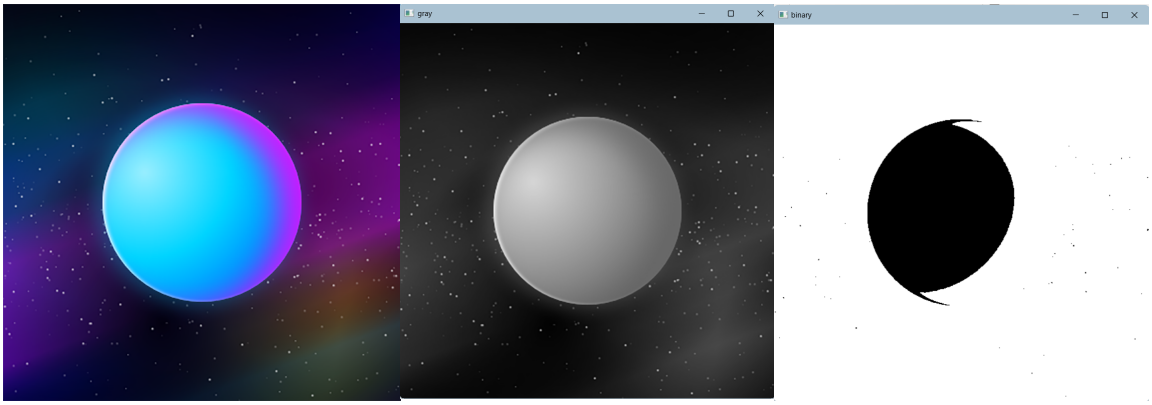
```
import numpy as np
import cv2
def ouline_track(img_gray):
    # 方向矩阵
    direct = np.array([[1,-1],[1,0],[1,1],[0,1],[-1,1],[-1,0],
[-1,-1],[0,-1]])
    h,w = img_gray.shape
    for row in range(h):
        for col in range(w):
            # 寻找起始点
            if img_gray[row][col] == 1 :
                graph = np.zeros_like(img_gray)
                startP = [row,col]
                currentP = startP
```

```
                currentD = 0
                while True:
                    #开始轨迹追踪
                    for i in range(8):
                        newP = currentP + direct[(currentD+i)%8]
                        if img_gray[newP[0]][newP[1]] == 1:
                            graph[currentP[0]][currentP[1]] = 1
                            currentP = newP
                            currentD = (currentD+i+7)%8 if
currentD%2 else (currentD+i+6)%8
                            break

                    if currentP[0] == startP[0] and currentP[1] ==
startP[1]:
                        return graph
```

```
import cv2
#读取测试图像
img = cv2.imread('testpic.png',flags = cv2.IMREAD_GRAYSCALE) #灰度值
读取
cv2.imshow('gray',img)
cv2.waitKey()
img_g = cv2.GaussianBlur(img,(27,27),sigmaX=1)  #高斯滤波
img_b = cv2.threshold(img_g,thresh= 130, maxval=255,type=
cv2.THRESH_BINARY_INV)    #二值化，type可调
cv2.imshow('binary',img_b[1])
cv2.waitKey()
# 由于图像非理想，采用建立矩阵的方式验证算法
# 示例用法
img_gray = np.array([[0, 0, 0, 0, 0, 0, 0],
                     [0, 0, 1, 1, 1, 0, 0],
                     [0, 1, 0, 0, 1, 0, 0],
                     [0, 1, 0, 1, 1, 1, 1],
                     [0, 1, 1, 0, 0, 0, 1],
                     [0, 1, 1, 1, 1, 1, 1],
                     [0, 0, 0, 0, 0, 0, 0]])

contour = outline_track(img_gray)
```

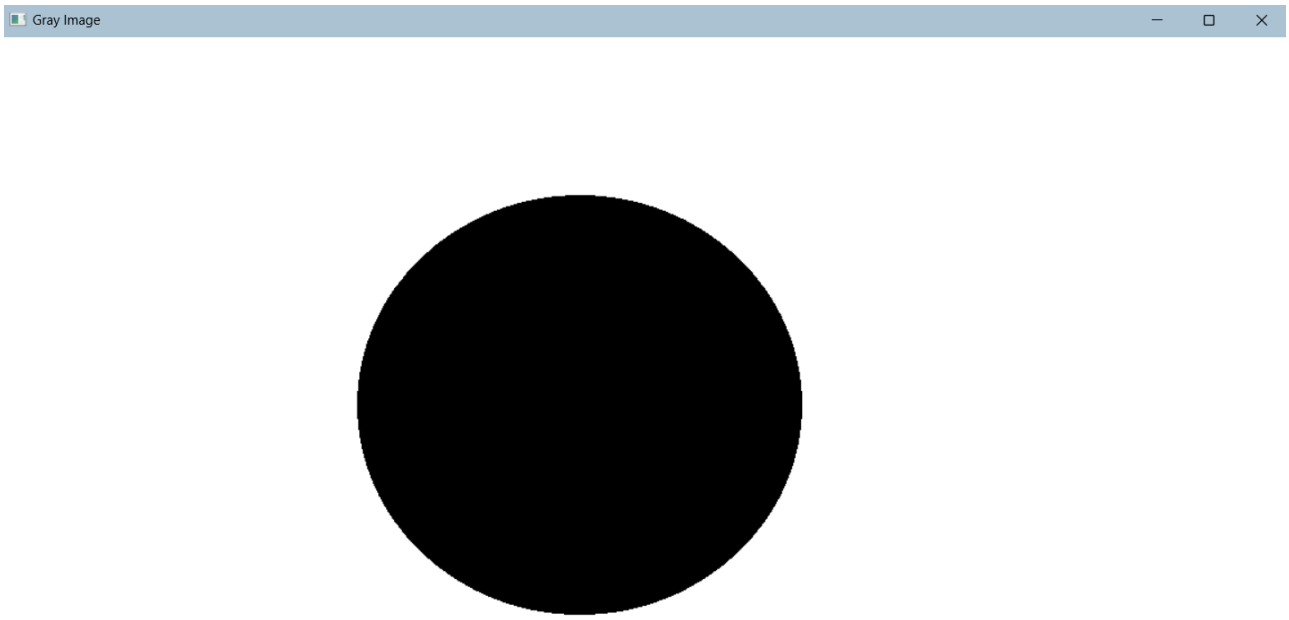
```
print(contour)
```

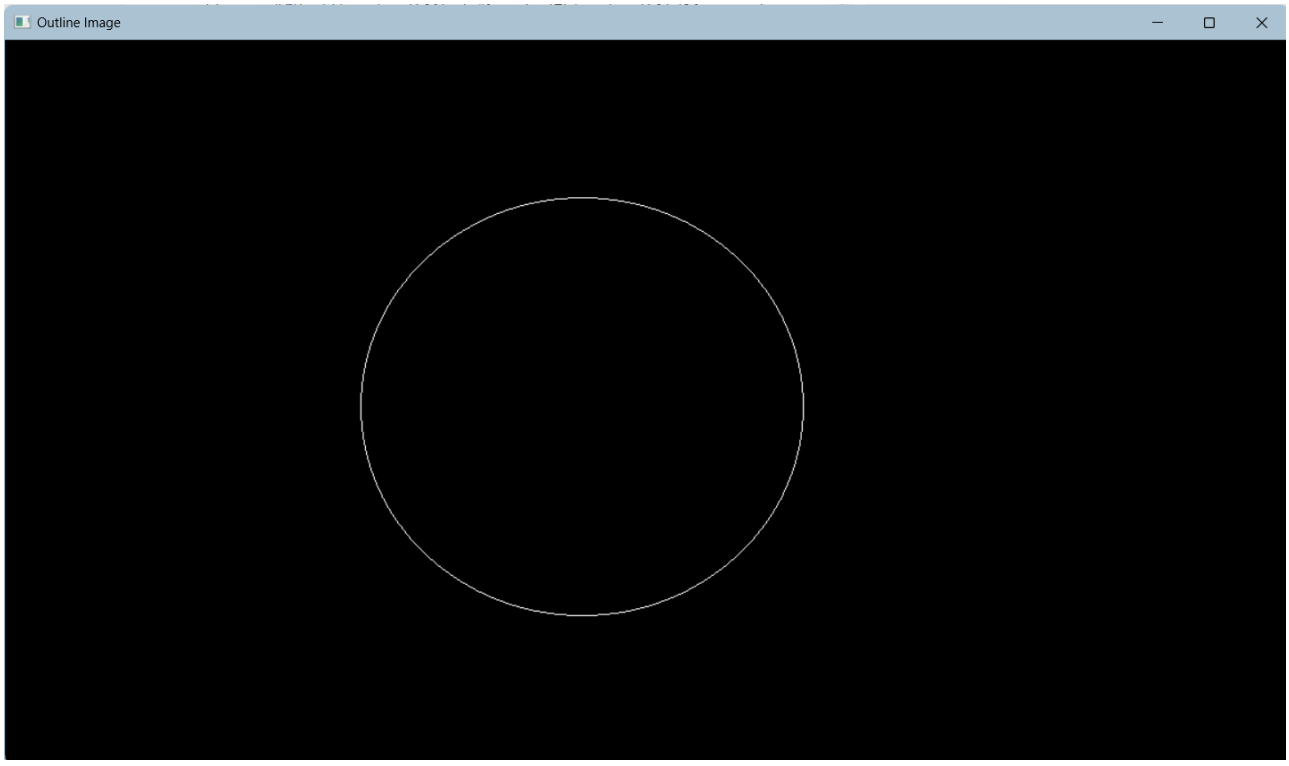
```
[[0 0 0 0 0 0 0]
 [0 0 1 1 1 0 0]
 [0 1 0 0 1 0 0]
 [0 1 0 0 0 1 1]
 [0 1 0 0 0 0 1]
 [0 1 1 1 1 1 1]
 [0 0 0 0 0 0 0]]
```




```
import cv2
import numpy as np

# 读取灰度图像
img_gray = cv2.imread('test.png', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Gray Image', img_gray)

# 转换为二值图像
img_gray = (img_gray/255).astype(int)

def outline_track(img_binary):
    assert len(img_binary.shape) == 2, "Input image must be a 2D grayscale image"
    assert np.unique(img_binary).size <= 2, "Input image must be a binary image"

    direct = np.array([[0, -1], [1, -1], [1, 0], [1, 1], [0, 1], [-1, 1], [-1, 0], [-1, -1]])
    h, w = img_binary.shape
```

```
    for row in range(h):
        for col in range(w):
            if img_binary[row][col] == 0:
                graph = np.zeros_like(img_binary)
                startP = [row, col]
                currentP = startP
                currentD = 0
                while True:
                    for i in range(8):
                        newP = currentP + direct[(currentD + i) %
8]
                        # 检查新的像素点是否位于图像边界内
                        if 0 <= newP[0] < h and 0 <= newP[1] < w
and img_binary[newP[0]][newP[1]] == 0:
                            graph[currentP[0]][currentP[1]] = 1
                            currentP = newP
                            currentD = (currentD + i + 7) % 8 if
currentD % 2 else (currentD + i + 6) % 8
                            break

                    if currentP[0] == startP[0] and currentP[1] ==
startP[1]:
                        return graph

img_l = outline_track(img_gray)
img_l = (img_l*255).astype(np.uint8)
cv2.imshow('Outline Image', img_l)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





为什么算法中每次的起始搜索方向要在上次搜索方向的基础上旋转1~2个方向？如果不这样做，对于有毛刺的区域，轮廓跟踪时会出现什么问题？

对于有毛刺的图像，即待跟踪图像为非凸图形，会存在尖刺区域在轨迹跟踪不被包括的问题。换言之，在跟踪后的图像中，尖刺被削平了。

当边界存在毛刺或细小的凹凸时，如果每次都只选择一个固定的搜索方向，可能会导致轮廓跟踪算法进入无限循环或跳过边界点的情况。

2. 试编写区域标记程序，并用一幅有多个对象的二值图像进行检验。

8连通区域的序贯标记算法如下：

1. 从左到右、从上到下扫描图像，寻找未标记的目标点P。
2. 如果P点的左、左上、上、右上4个邻点都是背景点，则赋予像素P一个新的标记；如果4个邻点中有1个已标记的目标像素，则把该像素的标记赋给当前像素P；如果4个邻点中有2个不同的标记，则把其中的1个标记赋给当前像素P，并把这两个标记记入一个等价表，表明它们等价。
3. 第二次扫描图像，将每个标记修改为它在等价表中的最小标记。

4连通区域的序贯标记算法与8连通区域的相同，只是在步骤(2)中仅判断左邻点和上邻点。

```
def sequential_labeling(image):
    rows, cols = image.shape
    label = 0
    labels = {}
    equivalent_labels = {}

    # 第一次扫描
    for i in range(rows):
        for j in range(cols):
            if image[i][j] != 0:
                left = image[i][j-1] if j > 0 else 0
                up = image[i-1][j] if i > 0 else 0

                if left == 0 and up == 0:
                    label += 1
                    image[i][j] = label
                    labels[label] = label
                elif left != 0 and up == 0:
                    image[i][j] = left
                elif left == 0 and up != 0:
                    image[i][j] = up
                else:
                    image[i][j] = min(left, up)
                    if left != up:
```

```
                        equivalent_labels[max(left, up)] =
min(left, up)

    # 更新等价表
    for k in range(label, 0, -1):
        if k in equivalent_labels:
            labels[k] = labels[equivalent_labels[k]]

    # 第二次扫描
    for i in range(rows):
        for j in range(cols):
            if image[i][j] != 0:
                image[i][j] = labels[image[i][j]]

    return image
```

```
img_t = np.array([
    [0,0,0,0,0],
    [0,1,1,0,0],
    [0,1,1,0,0],
    [0,0,0,1,1],
    [1,1,0,0,0]
    ])
img_a = sequential_labeling(img_t)
print(img_a)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

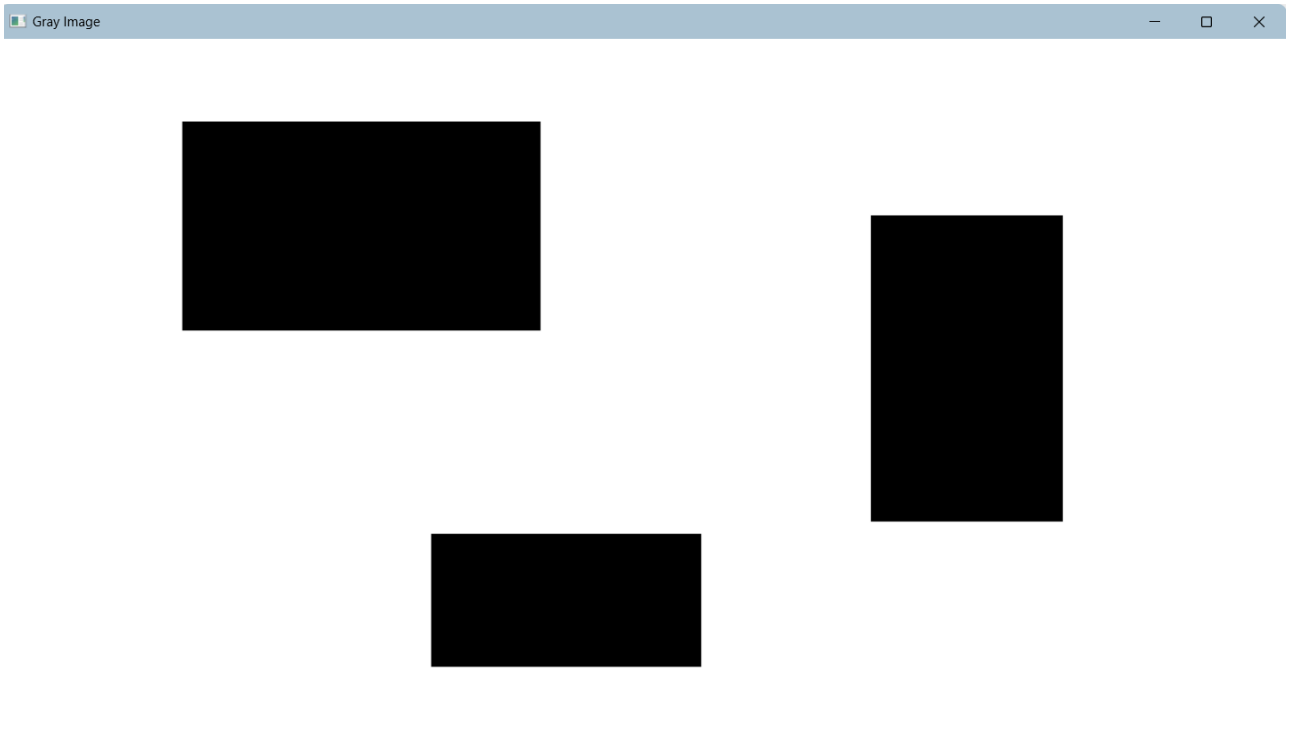
```
[[0 0 0 0 0]
 [0 1 1 0 0]
 [0 1 1 0 0]
 [0 0 0 2 2]
 [3 3 0 0 0]]
```

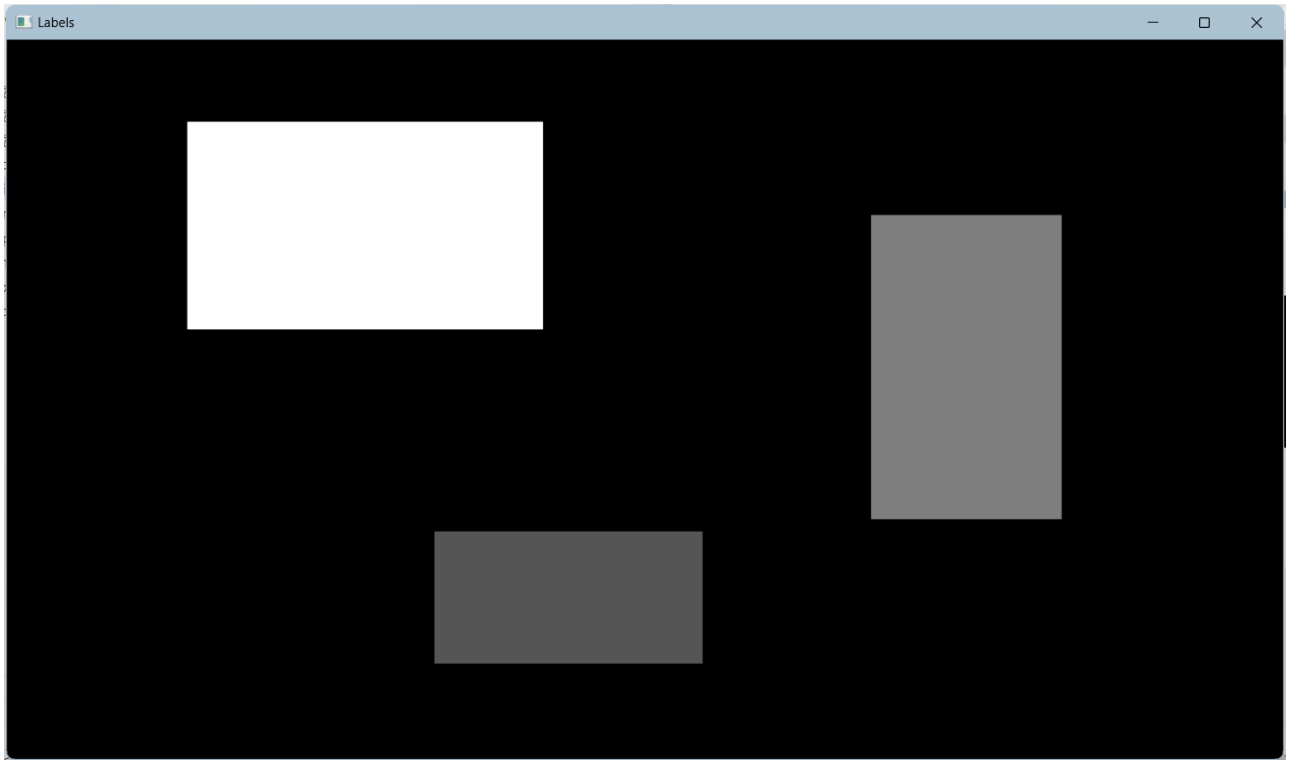
```
import cv2
import numpy as np
```

```
def sequential_labeling(image):
    rows, cols = image.shape
    label = 0
    labels = {}
    equivalent_labels = {}

    # 第一次扫描
    for i in range(rows):
        for j in range(cols):
            if image[i][j] != 0:
                left = image[i][j-1] if j > 0 else 0
                up = image[i-1][j] if i > 0 else 0

                if left == 0 and up == 0:
                    label += 1
                    image[i][j] = label
                    labels[label] = label
                elif left != 0 and up == 0:
                    image[i][j] = left
                elif left == 0 and up != 0:
                    image[i][j] = up
                else:
                    image[i][j] = min(left, up)
                    if left != up:
                        equivalent_labels[max(left, up)] =
min(left, up)

    # 更新等价表
    for k in range(label, 0, -1):
        if k in equivalent_labels:
            labels[k] = labels[equivalent_labels[k]]

    # 第二次扫描
    for i in range(rows):
        for j in range(cols):
            if image[i][j] != 0:
                image[i][j] = 255//labels[image[i][j]]

    return image.astype(np.uint8)
```

```python
# 读取灰度图像
img_gray = cv2.imread('test2.png', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Gray Image', img_gray)

# 转换为二值图像
img_binary = (1-img_gray/255).astype('uint')

img_a = sequential_labeling(img_binary)

# 显示标记图像
print(np.unique(img_a))
cv2.imshow('Labels', img_a)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
[  0  85 127 255]
```





注：颜色代表不同label。

3. 编写利用哈夫变换实现直线检测的程序。

在已知区域形状的条件下，利用哈夫变换(Hough Transform)可以方便地检测到边界曲线。哈夫变换的主要优点是受噪声和曲线间断的影响小，但计算量较大，通常用于检测已知形状的目标，如直线、圆等。

哈夫变换的原理是通过投票程序在参数空间中寻找不完美的对象实例。这种投票程序是在参数空间中进行的，从中可以获得作为算法显式构建的所谓累加器空间中局部最大值的对象候选项。

```
import cv2
import numpy as np

# 读取图像
image = cv2.imread('road.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 边缘检测
edges = cv2.Canny(gray, 50, 150)

# 哈夫变换直线检测
```

```
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 100,
minLineLength=100, maxLineGap=10)

# 绘制直线
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

# 显示结果
cv2.imshow('Result', image)
cv2.waitKey(0)
```