航拍图像、行车记录仪图像道路提取项目文档

项目背景

图像分割技术一直是计算机和数字图像处理领域中的重要问题之一。通过计算机自动识别并提取信息可大大减少人工劳动。

在飞机等飞行器的自动导航中最常用到的就是道路提取，在接近机场的上空，准确、快速的定位到跑到对于飞机到平稳降落起到至关重要的作用。

不仅在飞机到自动导航中，现在热门的自动驾驶等相关领域中也大量的应用着道路提取的各种算法，如何提高道路线识别的准确度，对于自动驾驶而言是至关重要的命题。

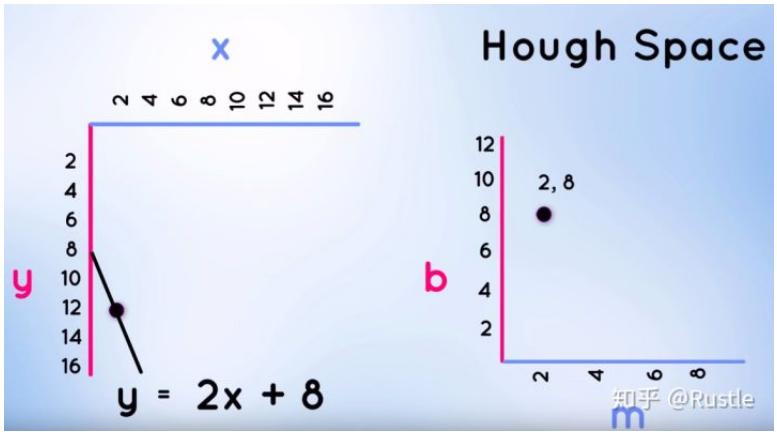
本次项目将围绕着图像分割算法——霍夫变换展开，进行简单的航拍图像道路提取以及行车记录仪道路线提取的相关实验。

霍夫变换

基本思想

将传统的图像从x y轴坐标体系变换到参数空间(m, b)或者**霍夫空间**中，通过在参数空间中计算局部最大值从而确定原始图像直线或圆所在位置

> 在我们从初中就开始使用的**平面直角坐标**中，一条直线的表示通常用$y = m_0x + b0$ 表示，其中m_0表示的是直线的**斜率**，b_0 表示的是直线的**截距**，一条直线上的点所使用的是同一个m_0b_0，因此我们可以设想一下，如果有一个坐标轴体系是以m_0为横轴，b_0为竖轴，形成以(m_0, b_0)为参数的**参数空间**，在平面坐标中同一条直线上的点在参数空间表示为一个点



应用领域

霍夫变换是一种**特征检测(feature)**，被广泛应用在**图像分析（image analysis）**、**计算机视觉(computer vision)**以及**数位影像处理 (digital image processing)**

识别能力

- 直线
- 圆形
- 椭圆形 等

车道线检测实战

1. Gaussian滤波

高斯滤波算法是一种去除**高频噪声**的常用方式，其在**Opencv**中的函数为**cv.GaussianBlur**，其参数大致有:

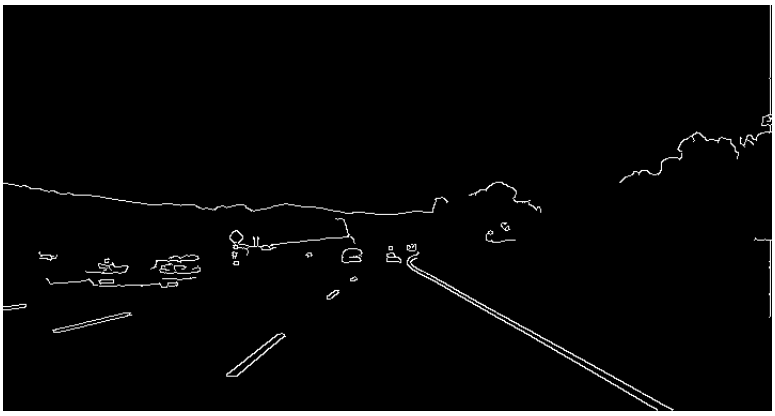
- src输入图像
- Size为**高斯核**大小，即高斯滤波器的尺寸
- 第三个参数即**高斯标准差**σ，一般默认为0

```
# 高斯滤波
def gaussian_blur(image, kernel_size):
    return cv.GaussianBlur(image, (kernel_size, kernel_size), 0)
```

2. Canny边缘检测

Canny边缘检测算法具体来说是一种**信息提取算法**，将原本复制的灰度图**最大限度保留信息**的情况下转换为**二值图像**，进而进行其它的操作

```
# Canny边缘检测
def canny(image, low_threshold, high_threshold):
    return cv.Canny(image, low_threshold, high_threshold)
```

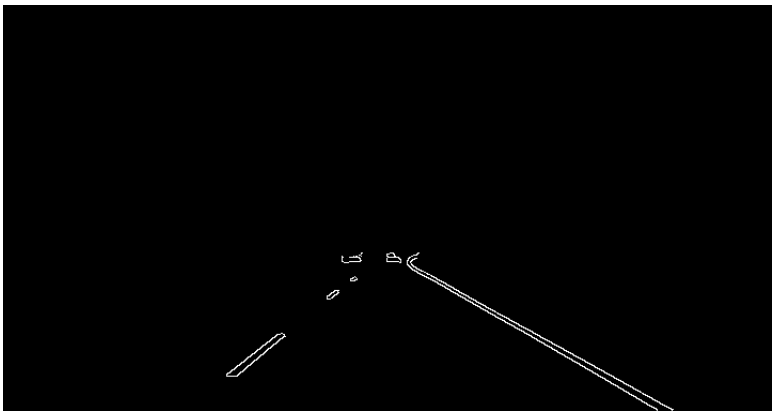


3. 生成Mask掩模

Mask掩模的作用为**降低计算代价**，即只在我们感兴趣部分进行算法的计算，如**霍夫变换**检测车道线

```
# 生成感兴趣区域即Mask掩模
def region_of_interest(image, vertices):

  mask = np.zeros_like(image)  # 生成图像大小一致的zeros矩

  # 填充顶点vertices中间区域
  if len(image.shape) > 2:
    channel_count = image.shape[2]
    ignore_mask_color = (255,) * channel_count
    else:
      ignore_mask_color = 255

      # 填充函数
      cv.fillPoly(mask, vertices, ignore_mask_color)
```

```
        masked_image = cv.bitwise_and(image, mask)
        return masked_image

    # 生成Mask掩模
    vertices = np.array([[(0, imshape[0]), (9 * imshape[1] / 20, 11 *
imshape[0] / 18),
                          (11 * imshape[1] / 20, 11 * imshape[0] / 18),
(imshape[1], imshape[0])]], dtype=np.int32)
    masked_edges = region_of_interest(edge_image, vertices)
```



4. 基于霍夫变换的直线检测

霍夫变换主要参数：

- **image**：输入图像，通常为canny边缘检测处理后的图像
- **rho**：线段以像素为单位的**距离精度**
- **theta**：像素以弧度为单位的**角度精度**(np.pi/180较为合适)
- **threshold**：霍夫平面累加的**阈值**
- **minLineLength**：线段最小长度(像素级)
- **maxLineGap**：最大允许**断裂**长度

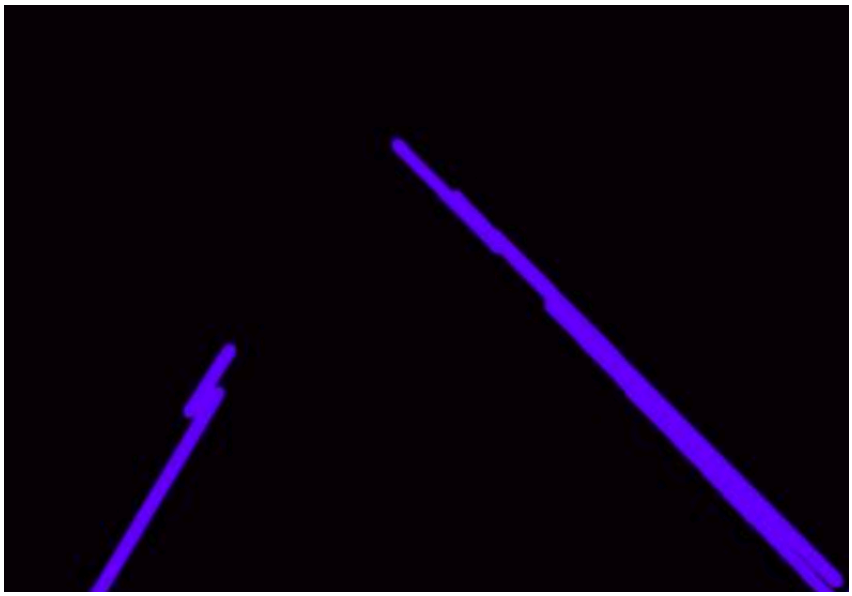
```
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):

  # rho：线段以像素为单位的距离精度
  # theta ： 像素以弧度为单位的角度精度(np.pi/180较为合适)
  # threshold ： 霍夫平面累加的阈值
  # minLineLength ： 线段最小长度(像素级)
  # maxLineGap ： 最大允许断裂长度
  lines = cv.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len, maxLineGap=max_line_gap)
  return lines
```

5. 绘制车道线

将上一步检测到的直线绘制出来

但是考虑到实际现实情况，仅仅绘制出来是不够的

> 上一步检测出的直线结果如下



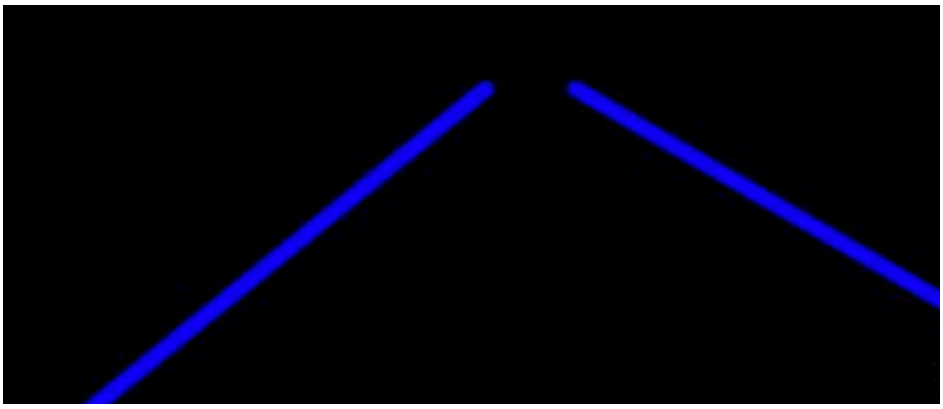
我们可以看到,出现了多条线段相互相邻这一情况,而我们所期望的是**单车道线检测**,便于直观体验与后续处理,因此我们在这一步还需对上一步检测到的直线进行进一步的**预处理**

对直线进行处理

1. 对每条直线求取斜率,分别归为左右列表中
2. 对得到的列表进行斜率平均值m_0的计算和最高点A的计算
3. 根据平均斜率m_0与最高点计算线段与图像下方的交点B坐标
4. 连接最高点A与交点B

```
line_image = np.zeros_like(image)
def draw_lines(image, lines, color=[255,0,0], thickness=2):

  right_y_set = []
  right_x_set = []
  right_slope_set = []

  left_y_set = []
  left_x_set = []
  left_slope_set = []

  slope_min = .35  # 斜率低阈值
  slope_max = .85  # 斜率高阈值
  middle_x = image.shape[1] / 2  # 图像中线x坐标
  max_y = image.shape[0]  # 最大y坐标

  for line in lines:
    for x1, y1, x2, y2 in line:
      fit = np.polyfit((x1, x2), (y1, y2), 1)    # 拟合成直线
      slope = fit[0]  # 斜率

      if slope_min < np.absolute(slope) <= slope_max:

        # 将斜率大于0且线段x坐标在图像中线右边的点存为右边车道线
        if slope > 0 and x1 > middle_x and x2 > middle_x:
```

```
            right_y_set.append(y1)
            right_y_set.append(y2)
            right_x_set.append(x1)
            right_x_set.append(x2)
            right_slope_set.append(slope)

            # 将斜率小于0且线段x坐标在图像中线左边的点存为左边车道线
            elif slope < 0 and x1 < middle_x and x2 < middle_x:
              left_y_set.append(y1)
              left_y_set.append(y2)
              left_x_set.append(x1)
              left_x_set.append(x2)
              left_slope_set.append(slope)

              # 绘制左车道线
              if left_y_set:
                lindex = left_y_set.index(min(left_y_set))  # 最高点
                left_x_top = left_x_set[lindex]
                left_y_top = left_y_set[lindex]
                lslope = np.median(left_slope_set)   # 计算平均值

                # 根据斜率计算车道线与图片下方交点作为起点
                left_x_bottom = int(left_x_top + (max_y - left_y_top) / lslope)

                # 绘制线段
                cv.line(image, (left_x_bottom, max_y), (left_x_top, left_y_top),
color, thickness)

                # 绘制右车道线
                if right_y_set:
                  rindex = right_y_set.index(min(right_y_set))  # 最高点
                  right_x_top = right_x_set[rindex]
                  right_y_top = right_y_set[rindex]
                  rslope = np.median(right_slope_set)

                  # 根据斜率计算车道线与图片下方交点作为起点
                  right_x_bottom = int(right_x_top + (max_y - right_y_top) /
rslope)

                  # 绘制线段
                  cv.line(image, (right_x_top, right_y_top), (right_x_bottom,
max_y), color, thickness)
```



6. 图像融合

将**原始图像**与我们刚绘制的**车道线图像**进行比例的**融合**,这里需要介绍一个**函数cv.addWeighted**,参数src1和alpha表示图像或矩阵和它对应的权重(Weight),src2和beta表示的则是第二副图像或矩阵和它对应的权重,第五个参数gamma表示整体添加到数值,默认为0即可

这里我们将原图权重设为0.8,车道线图像设为1,则最后呈现的效果为车道线较为明显,**可视化程度提高**

> 本实验包提供的是黑白图片，同学们也可以自己寻找彩色图片进行道路提取

航拍图像机场跑道检测实战

标准霍夫变换

算法说明

cv2.HoughLines()

- **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。

- **输出：**[float, float]形式的ndarray，其中每个值表示检测到的线(ρ , θ)中浮点点值的参数

算法实现

1. 使用Canny算子获得图像边缘

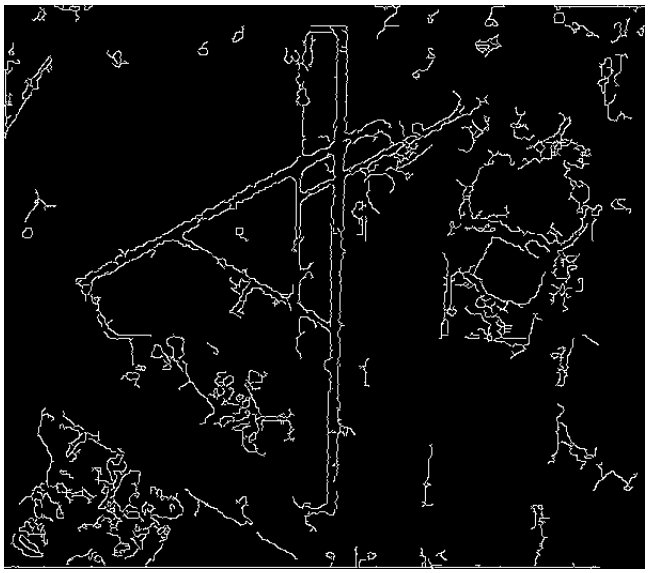
2. 使用Hough变换检测直线（其中HoughLines函数的参数3和4对应直线搜索的步长）

> 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(rho, theta)在原图像中为一条直线

```
def Standard_Hough(img):
  '''
    :description: 标准霍夫变换

    :param : 原始图像

    :return : 显示用标准霍夫变换提取的直线边缘图像
    '''
  house = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 获取灰度图
  edges = cv2.Canny(house, 50, 200)
  lines = cv2.HoughLines(edges, 1, np.pi / 180, 260)  # 霍夫变换返回的就是极坐标系中的两个参数  rho和theta
  print(np.shape(lines))
  lines = lines[:, 0, :]  # 将数据转换到二维
  for rho, theta in lines:
    a = np.cos(theta)
    b = np.sin(theta)
    # 从图b中可以看出x0 = rho x cos(theta)
    #               y0 = rho x sin(theta)
    x0 = a * rho
    y0 = b * rho
    # 由参数空间向实际坐标点转换
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * a)
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * a)
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 1)
    cv2.imshow('img', img)
    cv2.imshow('edges', edges)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

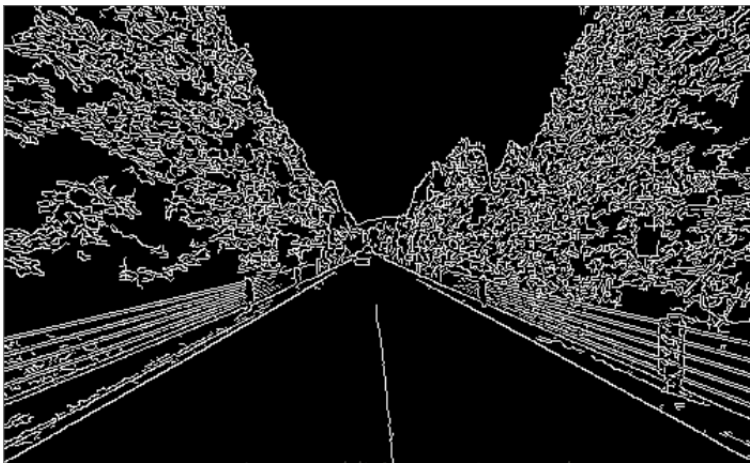
实验效果

图像边缘

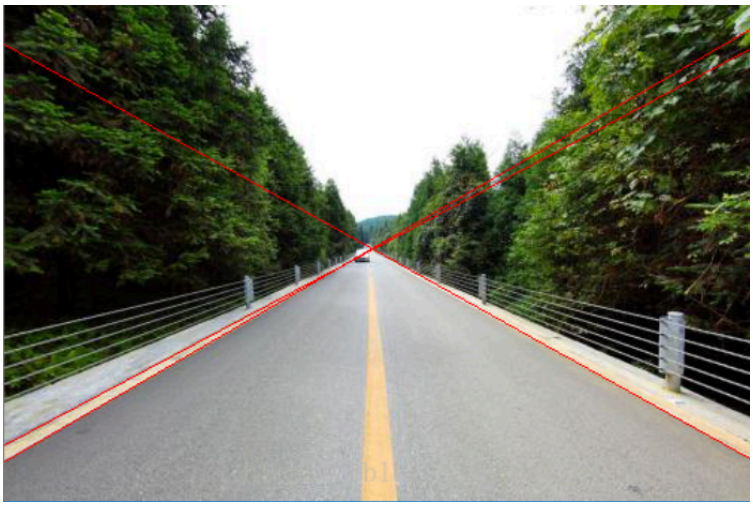


机场跑道

边缘效果



提取的道路直线



> 可以看到标准的霍夫变换并不适用各种情况，因此我们引出了进一步的算法改进

渐进概率式霍夫变换

算法说明

cv2.HoughLinesP()

- **minLIneLength:** 线的最短长度，比这个线段短的都忽略掉
- **maxLineGap:** 两条直线之间的最大间隔，小于此值，就认为是一条直线

> 从原理上讲hough变换是一个耗时耗力的算法，尤其是对每一个点的计算，即便经过了canny转换，但有的时候点的数量依然很庞大，这时候采取一种概率挑选机制，不是所有的点都进行计算，而是随机的选取一些点来进行计算，这样的话在阈值设置上也需要降低一些

算法实现

```
def ProgressiveProbability_Hough(img):
  '''
    :description: 渐进概率式霍夫变换

    :param : 原始图像

    :return : 显示用渐进概率式霍夫变换提取的直线边缘图像
    '''
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  edges = cv2.Canny(gray, 50, 250)
  lines = cv2.HoughLinesP(edges, 1, np.pi/180, 30, minLineLength=60,
maxLineGap=10)
  lines = lines[:, 0, :]
  for x1, y1, x2, y2 in lines:
    cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 1)
    cv2.imshow('img', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

实验效果