# 任务二 边线搜索

## 🗺️背景介绍

在任务一中，我们已完成图像分割工作，将智能车的可行域转化为白色区域。然而，为了更精准地控制车辆在赛道中的行驶姿态，还需进一步从二值化图像中提取赛道边缘信息。这些边缘特征将为后续的闭环控制（如路径拟合等）提供关键的几何参考，是实现智能车稳定行驶的重要基础。

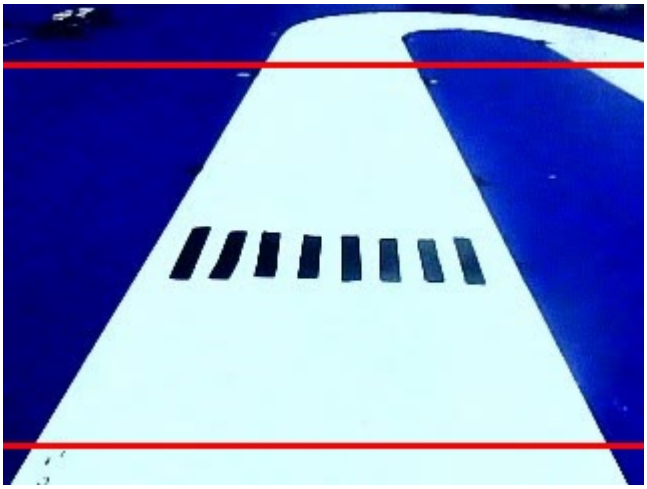
## 🖥️环境配置

IDE：Pycharm
相关依赖：numpy、opencv

## ⚔️实践环节拆解

常见的搜线法有很多，例如**中线继承法、八邻域、迷宫法**，有些算法可以有效降低搜线的时间复杂度，适合算力资源有限的mcu；有些算法可以更好地应对复杂的赛道场景。
这里介绍最简单的扫线方法：**遍历搜线**，具体代码可**参考**“cpp参考代码”
（当然，可以选择其他搜线方法完成该任务，其他方法介绍可见参考资料1）

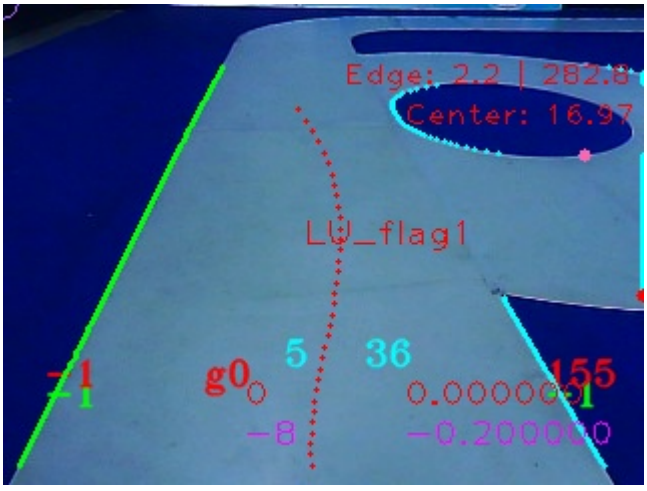
1. **算法实现** 难度：⭐⭐⭐
   1. 设置底部、顶部切行
   图像中并非所有行都具备有效信息：底部部分行可能包含车身区域，顶部区域因距离过远不仅参考价值有限，还易引入无关干扰。因此，通过对图像底部和顶部进行适当裁切，可减少无效区域对边缘检测的影响，提升赛道边缘搜索的稳定性。



   2. 确定起始行
   起始行，即最底部的有效赛道行，其白色色块宽带较大，可以利用此特征来过滤一些底部噪声色块。
   3. 遍历搜索
   从起始行开始一行一行向上搜索，每行从左至右，遍历像素，搜索白色色块（记录色块起点、终点的坐标）。

4. 判断连通性
赛道是连续的，因此行与行之间的色块一定是连通的，通过连通性可以过滤噪声色块

2. **绘制图像** 难度：⭐
可视化，可以更好地帮助我们测试、调试程序。将搜索到的点集绘制到图上。
可参考下图：左边点集绿色，右边点集蓝色




## 任务要求

1. 利用res中的视频，实现左、右赛道点集的搜索
2. 将点集绘制在图上

## 参考资料

1. 常见的边线搜索方法：https://blog.csdn.net/qq_58114029/article/details/132132607
2. 代码设计参考：

- 使用类
- 功能模块化，便于后续维护

```
class Point: # 二维平面坐标
    def __init__(self, row, col):
        self.x = row
        self.y = col
class Tracking:
    def __init__(self):
        # 初始化左边和右边的点集合
        self.LeftPoints = []
        self.RightPoints = []
        # 初始化起始行标志位
        self.start_flag = True
        # 初始化裁剪的范围
        self._top_cut = 30
        self._bottom_cut = 20
        # 存放一行搜索到的白色色块
        self._white_block = []
```

```
def process(self,img_binary):
    # 初始化
    self.start_flag = True
    self.LeftPoints.clear()
    self.RightPoints.clear()
    # 遍历二值图像
    for row in range(220,30):
        for col in range(0,319):
            # 遍历一行的所有像素，提取白色色块
            self.search_white_block()
            self._white_block.append(Point(block_start,block_end))
        if len(self._white_block) > 0:  # 检查white_block是否包含元素
            if self.start_flag:
                # 起始行搜索逻辑
                self.search_start_line()  # 示例方法调用
            else:
                # 非起始行搜索逻辑
                self.search_normal_line()  # 示例方法调用
                self.LeftPoints.append(Point(row,start))
                self.RightPoints.append(Point(row,end))

def search_white_block(self):
    pass
def search_start_line(self):
    pass
def search_normal_line(self):
    pass
```