任务三 中心线拟合

🗺️背景介绍

在任务二中，我们已从二值化图像中成功提取出赛道的左、右边缘点集。智能车的运动轨迹需遵循一条明确的参考线，而这条轨迹线可通过左、右边缘点集拟合得到。考虑到行驶稳定性与路径最优性，通常以赛道中心线作为理想行驶轨迹 —— 即通过拟合左、右边缘的中间对称线，为智能车提供精准的导航基准。

🖥️环境配置

IDE：Pycharm
相关依赖：numpy、opencv

⚔️实践环节拆解

1. **中值法** 难度：⭐
   针对图像中每一行的左、右赛道边缘点集，计算两点的中间对称点，以此构成中心线的点集。这种方法通过直接取每行边缘点的中值位置，快速生成中心线基础数据，实现简单且实时性较好。
2. **贝塞尔拟合** 难度：⭐⭐
   在左、右边缘点集中分别选取三等分点，对所采集的四组对应点（左、右各四点）逐一计算中点，再基于这些中点通过贝塞尔曲线进行拟合，最终得到赛道中心线。该方法通过特征点采样精简数据维度，借助贝塞尔曲线的平滑特性，能够生成连续性更优的中心轨迹；但受限于仅用四点进行拟合，若采样点存在偏差，贝塞尔曲线的拟合结果会进一步放大这一误差，可能对轨迹实际方向产生影响。
   c++程序参考：

```
vector<POINT> Bezier(double dt, vector<POINT> input)//dt控制步长
{
    vector<POINT> output;

    double t = 0;
    while (t <= 1)
    {
        POINT p;
        double x_sum = 0.0;
        double y_sum = 0.0;
        int i = 0;
        int n = input.size() - 1;
        while (i <= n)
        {
            double k =
                factorial(n) / (factorial(i) * factorial(n - i)) * pow(t, i)
```

```
* pow(1 - t, n - i);
            x_sum += k * input[i].x;
            y_sum += k * input[i].y;
            i++;
        }
        p.x = x_sum;
        p.y = y_sum;
        output.push_back(p);
        t += dt;
    }
    return output;
}
```

3. **解耦图像：特征工程** 难度：⭐

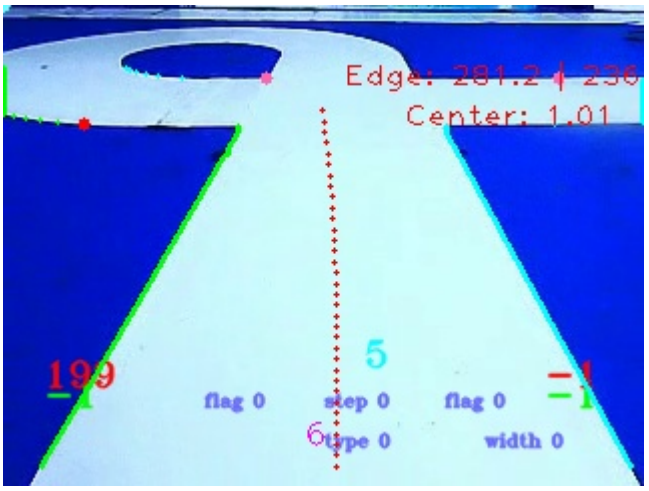
   在前面的任务中，我们已将搜索到的赛道线数据存储在 track 类中。这一设计使得在进行模式识别时，无需重复对图像进行搜索操作，直接从 track 类的点集中提取特征即可，实现了图像搜索与特征分析的解耦。

   此处先实现最基础的统计学特征 —— 方差：通过计算点集的方差，可量化赛道边缘的离散程度，进而判断前方路段是直道（方差较小，点集分布集中）还是弯道（方差较大，点集分布分散）。这种基于特征的判断方式，能为赛道模式识别提供简洁且有效的量化依据。

   c++ 程序参考：

```
double sigma(vector<int> vec)
{
    if (vec.size() < 1)
        return 0;

    double aver = average(vec); // 集合平均值
    double sigma = 0;
    for (int i = 0; i < vec.size(); i++)
    {
        sigma += (vec[i] - aver) * (vec[i] - aver);
    }
    sigma /= (double)vec.size();
    return sigma;
}
```



任务要求

1. 利用res中的视频，使用至少一种方法，实现中心线拟合
2. 将中心线绘制在图上
3. 计算track中的左、右点集的方差，作为成员变量存储；计算center中的中心点集方差，作为成员变量存储。并绘制在图像上。

参考资料

1. 贝塞尔拟合：
   https://blog.csdn.net/weixin_43673156/article/details/128600747
2. 参考 “cpp参考代码”
3. 代码设计参考：

- 使用类

```
class Center:
    def __init__(self):
        self.CenterPoints = []
        self.sigma_center = 0.0
    def process(self,track):
        return None
    def cal_sigma_center(self):
```