智能车学习日记【二】————十字和斜入十字

前言

经过了一个多月的学习，小白的我终于也是能较稳定调出了 2.8m 以上的四轮车，过程中也遇到了很多困难, 其中就有十字补线和斜入十字补线。尤其是斜入十字也是困扰了我快一周，现在和大家分享一下我处理斜入十字的方法(我的方法其实还有很多漏洞需要改进，仅供大家参考理解一下，欢迎各位在评论区指点出可改进之处！)
(后注：源码不开源)

一、普通十字补线

首先，我将十字补线分成了找拐点，判断是否需要补线和补线 3 个步骤。斜入十字也是这样判断，只是多了很多限制条件。防止在直道或弯道也补线导致图像错误。代码里的数组可以在我的第一篇文章里找到定义和含义
智能车学习日记【一】

找拐点

正入十字找拐点比较容易，直接贴代码

```
// 找拐点 L_black 左边线   R_black 右边线
if (l_down_flag == 0)

            {
                if (L_black[0] != 185 && L_black[1] != 185 && L_black[3] != 185)
                {
                    for (i = 1; i < 65; i++)
                    {
                        if (abs(L_black[i] - L_black[i - 1]) < 5 && abs(L_black[i + 1] -
L_black[i]) < 5 && L_black[i + 2] - L_black[i + 1] > 3 && i + 1 >= 2 && L_black[i + 1] < 180
&& i + 1 < 45)
                        {
                            lefty[0] = (byte)(i + 1);
                            SetText("  左下拐点  y x +    " + (i + 1) + "      " + L_black[i +
1]);
                            l_down_flag = 1;
                            break;
                        }
                    }
                }

            }
if (r_down_flag == 0)
            {
                if (R_black[0] != 0 && R_black[1] != 0 && R_black[3] != 0)
                {
                    for (i = 1; i < 65; i++)
                    {
                        if (abs(R_black[i] - R_black[i - 1]) < 5 && abs(R_black[i + 1] -
R_black[i]) < 5 && R_black[i + 2] - R_black[i + 1] < -3 && i + 1 >= 2 && R_black[i + 1] > 2
&& i + 1 < 45)
                        {
                            righty[0] = (byte)(i + 1);
                            SetText("  右下拐点  y x+    " + (i + 1) + "      " + R_black[i +
1]);
```

```
                            r_down_flag = 1;
                            break;
                        }
                    }
                }

            }
if (l_up_flag == 0)
            {
                for (i = 1; i < 65; i++)
                {
                    if (L_black[i] - L_black[i - 1] < -3 && abs(L_black[i + 1] - L_black[i]) <
4 && abs(L_black[i + 2] - L_black[i + 1]) < 4 && L_black[i] < 181 && (i > lefty[0]))
                    {
                        lefty[1] = (byte)(i);
                        SetText("  左上拐点  y x+    " + (i) + "      " + L_black[i]);
                        l_up_flag = 1;
                        break;
                    }
                }

            }
if (r_up_flag == 0)
            {
                for (i = 1; i < 65; i++)
                {
                    if (R_black[i] - R_black[i - 1] > 3 && abs(R_black[i + 1] - R_black[i]) <
4 && abs(R_black[i + 2] - R_black[i + 1]) < 4 && R_black[i] > 2 && (i > righty[0]))
                    {
                        righty[1] = (byte)i;
                        SetText("  右上拐点  y x+    " + (i) + "      " + R_black[i]);
                        r_up_flag = 1;
                        break;
                    }
                }
            }
```
思路其实就是用了差值的方法，以左下拐点为例，第 i 行和第 i-1 行的边界点 x 坐标的差值在 5 之内，第 i+1 行和第 i 行的差值在 5 之内，第 i+2 行和第 i+1 行差值大于 3，则我判断它为左下拐点。其他拐点找到方法类似。

判断是否补线和补线
拐点并不是只有在十字时才有，在弯道时也能找到，但是过弯时我们不应该补线，所以要有判断条件。

如这张正入十字的图，我们是可以通过左右扫线和上面找拐点方式找到 4 个拐点的，找到拐点后我们就需要判断类型和是否需要补线
这张图为入十字前补线。
我对入十字前补线的判断条件是左右两边同时丢线行大于 4，并且 4 个拐点都找到，才执行入十字前补线。补线思路就是左下拐点和左上拐点之间用最小二乘法求斜率和截距拟合出左边线，右边同样方法拟合出右边线，然后再根据新左右边线重新拟合出中线
下面贴代码
```
// 入十字前补线* advanced_regression 是最小二乘法求斜率截距 *run 是拟合直线函数 1
为左 2 为右 0 为中
//abu 是二次扫线的计数位，可先忽略
// a 是左右共同丢线行数
if ((a >= 4&& (lefty[0] > 0 && lefty[1] > 0) || (righty[0] > 0 && righty[1] > 0)) || abu >= 3)
        {
                if (lefty[0] >= 2 && lefty[1] >= 2 && guai == 0)
//找左线            入十字前
                {
                    advanced_regression(1, lefty[0] - 2, lefty[0], lefty[1], lefty[1] + 2);
                    SetText(" 左斜率 + " + parameterB + " 左截距 " + parameterA);
                    run(1, lefty[0], lefty[1], parameterB, parameterA);
                    flagl = 1;
//
                }
                if (righty[0] >= 2 && righty[1] >= 2 && guai == 0)
//找右线
                {
                    advanced_regression(2, righty[0] - 2, righty[0], righty[1], righty[1] + 2);
                    SetText(" 右斜率 + " + parameterB + " 右截距 " + parameterA);
                    run(2, righty[0], righty[1], parameterB, parameterA);
                    flagr = 1;
                }
                if ((r_down_flag == 1 || l_down_flag == 1) && guai == 0)
                {
                    if (flagl == 1 && flagr == 0)
                    {
                        run(0, lefty[0], lefty[1], parameterB, parameterA);
//拟合中线
                        SetText("入十字前拉线 " + a);
                        if (abu >= 3)
                        {
                            run(0, 0, lefty[0], parameterB, parameterA);
                            run(0, lefty[1], 50, parameterB, parameterA);
                        }
                    }
                    else
                    {
                        run(0, righty[0], righty[1], parameterB, parameterA);
//拟合中线
                        SetText("入十字前拉线 " + a);
                        if (abu >= 3)
                        {
                            run(0, 0, righty[0], parameterB, parameterA);
                            run(0, righty[1],50, parameterB, parameterA);
                        }
                    }

                }

            }
```
这里是我参考的最小二乘法求斜率截距函数。
https://blog.csdn.net/qq_42604176/article/details/107719565
最小二乘法拟合
```
//拟合直线函数
void run(int type, int startline, int endline, float parameterB, float parameterA)
        {
            if (type == 1) //左
            {
                for (int i = startline; i < endline; i++)
                {
                    L_black[i] = (byte)(parameterB * i + parameterA);
                    if (L_black[i] < 0)
                    {
                        L_black[i] = 185;

                    }
                    if (L_black[i] > 185)
                    {
                        L_black[i] = 185;

                    }
                }
            }
            else if (type == 2)                    //右
            {
                for (int i = startline; i < endline; i++)
                {
                    R_black[i] = (byte)(parameterB * i + parameterA);
                    if (R_black[i] < 0)
                    {
                        R_black[i] = 0;

                    }
                    if (R_black[i] > 185)
                    {
                        R_black[i] = 0;

                    }
                }
            }
            else if (type == 0)                      //中
            {
                for (int i = startline; i < endline; i++)
                {
                    LCenter[i] = (byte)((L_black[i] / 2 + R_black[i] / 2));
                    if (LCenter[i] < 0)
                    {
                        LCenter[i] = 0;

                    }
                    if (LCenter[i] > 185)
                    {
                        LCenter[i] = 185;

                    }
                }
            }
        }
```
拟合后，就可以的得到入十字前的补线图
当然正入十字分为入十字前和入十字中，入十字中也类似，就是找到左上和右上拐点，然后向下拉线就好了

二、斜入十字

不知道有多少人和我一样遇到这种情况，在斜入十字时边线扫描不对，本该是左边线的变成了右边线，导致了中线拟合错误，车子也就跑错道路了，这种情况找到的拐点一般也不对。

1.判断是否为斜入十字和斜入十字类型
我将斜入十字分为左斜入十字和右斜入十字，上图为左斜入十字。判断是否为斜入十字。首

先在一般情况下，左右边界应该是大部分正常分布在中线两端，很少出现上面很多行的右边线大于下边的中线，或上边左边线小于下边中线的情况（我还没写环岛和三岔，所以目前没遇到过），所以当遇到上图这样，上边有大量右线比下边中线大时，则判定为（左）斜入十字

代码如下：

```
for (i = 1; i < 40; i++)
            {
                if (i < 20)
                {
                    if ((R_black[i] == 0 && L_black[i] < 184 && R_black[0] == 0 &&
R_black[1] == 0 && R_black[i+1] > LCenter[0] && R_black[i + 2] > LCenter[1]) || (R_black[0]
== 0 && R_black[1] == 0 && R_black[i] > LCenter[5] && R_black[i] > LCenter[7] &&
R_black[i-1] > LCenter[7]))
                    {
                        if ((R_black[0] == 0 && R_black[1] == 0 && R_black[i] > LCenter[5]
&& R_black[i] > LCenter[7] && R_black[i - 1] > LCenter[7] && L_black[0] != 185 &&
L_black[1] != 185))
                        {
                            cntl = 9;
                        }
                        cntl++;
                    }
                    else if ((L_black[i] == 185 && R_black[i] > 1 && L_black[0] == 185 &&
L_black[1] == 185 && L_black[i+1] < LCenter[0] && L_black[i + 2] < LCenter[1]) || (L_black [0]
== 0 && L_black[1] == 0 && L_black[i] > LCenter[5] && L_black[i] > LCenter[7] && L_black[i-
1] > LCenter[7]))
                    {
                        if (L_black[0] == 0 && L_black[1] == 0 && L_black[i] > LCenter[5]
&& L_black[i] > LCenter[7] && L_black[i - 1] > LCenter[7] && R_black[0] != 0 && R_black[1] !=
0)
                        {
                            cntr = 9;
                        }
                        cntr++;
                    }
                }
                else
                {
                    if ((R_black[0] == 0 && R_black[1] == 0 && R_black[i] > LCenter[5]
&& R_black[i] > LCenter[7] && R_black[i - 1] > LCenter[7] && L_black[0] != 185 &&
L_black[1] != 185))
                    {
                        cntl = 9;
                        break;
                    }
                    else if (L_black[0] == 185 && L_black[1] == 185 && L_black[i] <
LCenter[5] && L_black[i] < LCenter[7] && L_black[i - 1] < LCenter[7] && R_black[0]!=0 &&
R_black[1] != 0)
                    {
                        cntr = 9;
                        break;
                    }
                }

            }
            if (cntl > 8 && cntr < 8)
            {
                cntl = 255;                                         //左斜入十字
                SetText("可能为左斜入十字    " + cntl);
            }
            else if (cntr > 8 && cntl < 8)
            {
                cntr = 255;                                         //右斜入十字
                SetText("可能为右斜入十字    " + cntr);
            }
```

2.找斜入十字拐点

判断完后就是要找拐点。还是以这图为例。其实我们只要找到左上拐点，因为左下拐点可以通过上面的找拐点方式找出来，而且一般不会找错，都是上拐点找错(因为边线错误)
上拐点找的方式也特别简单，我们已经知道了现在是左斜入十字，而且上边的右边线本应该是左边线，所以我们就找上边右边线突变点。图中就是蓝点标记出来的位置，可以看见蓝点之前右边线都是 0，蓝点以后的线本来应该是左边线，却被当作了右边线，所以，这个蓝点就是新的左上拐点。这样我们就找到了左下和左上拐点，进行拟合拉线就可以了

```
guai = 0;       //二次扫线标志位    0 不执行
t = 0;
if (cntl == 255)
            {
                for (i = 5; i < 65; i++)
                {
                    if ((R_black[i] - R_black[i - 1] > 15) && (R_black[i] - R_black[i - 2] > 
15))
                    {

                        L_black[i] = R_black[i];
                        R_black[i] = R_black[i - 1];
                        if (t == 0)
                        {
                            guai = i;
                        }
                        t++;
                        LCenter[i] = (byte)(R_black[i] / 2 + L_black[i] / 2);
                    }
                }
                if (guai != 0)
                {
                    SetText("新左上拐点 y x    " + guai + "    " + L_black[guai]);
                    SetText("左斜入十字    " + cntl);
                }
                else
                {
                    SetText("没找到新左上拐点 y x    " + guai + "    " + L_black[guai]);
                }

            }
            else if (cntr == 255)
            {
                for (i = 5; i < 65; i++)
                {
                    if ((L_black[i] - L_black[i - 1]) < -15 && (L_black[i] - L_black[i - 2]) < -
15)
                    {
                        R_black[i] = L_black[i];
                        L_black[i] = L_black[i - 1];
                        if (t == 0)
                        {
```

```
                        guai = i;
                    }
                    t++;
                    LCenter[i] = (byte)(R_black[i] / 2 + L_black[i] / 2);
                }
            }
            if (guai != 0)
            {
                SetText("新右上拐点 y x    " + guai + "    " + R_black[guai]);
                SetText("右斜入十字    " + cntl);
            }
            else
            {
                SetText("没找到新右上拐点 y x    " + guai + "    " + R_black[guai]);
            }

        }
```
补好以后车子遇到斜入十字也基本不会跑错了，当然情况很多，遇到了还是要具体问题具体分析。下图左斜入的口子已经给补好了，上面的边线其实还可以改的更好看一点的，但是时间比较紧，车子也不会跑错，就没管了 XD。
右斜入也是类似方法。
总结
斜入十字确实是新手入门的一大难点之一，我也困扰也很久，现在其实也只是停留在车子跑对道路，限制了车子稳定突破 2.9m 以上，后面还需要经过各种修改来让图更好看，车跑的更稳。一点点见解希望能帮到大家。如果觉得有点点帮助还希望点赞支持一下新人博主，有问题可以在评论区留言哦~。欢迎大佬指点一下可以改进的地方，一起学习。

方法更新：
后续为了使图像更好看，特意加了一种特殊扫线，即在十字状态机中找一条动态中线，以动态中线为基准向左右扫线，这样可以避免中线扫歪的情况出现。
```
//  动态中线扫线
 void hd2_findmiddle()
 {
     byte max_X = 0;
     byte max_Y = 0;
     for (x = R_black[0]; x < L_black[0]; x++)       //找一条中线进行左右扫线
     {
         for (y = 1; y < 70; y++)
         {
             if (y == 69 || (Pixels[y - 1][x] != 0 && Pixels[y][x] == 0))
             {
                 if (y > max_Y)
                 {
                     max_Y = y;
                     max_X = x;
                     break;
                 }
             }
             if (Pixels[y - 1][x] == 0 && Pixels[y][x] == 0)
             {
                 break;
             }
         }
     }
     if (max_X == 0)
         max_X = 1;
     if (max_X == 185)
         max_X = 184;
```

```
        // SetText("max_Y=" + max_Y + " max_X=" + max_X + "  根据中线 x=" + max_X + "拉线
");
         for (y = 1; y < 70; y++)
         {
                if (Pixels[y][max_X] == 0 || y == 69)
                {
                       b = y;
                       break;
                    // SetText("新丢线行 b=" + b);
                }
                for (x = max_X; x < 186; x++)
                {
                       if (x == 185 || (Pixels[y][x] != 0 && Pixels[y][x - 1] != 0 && Pixels[y][x + 1] ==
0))                    //左边界
                       {
                              L_black[y] = x;
                              break;
                       }
                }

                for (x = max_X; x >= 0; x--)
                {
                       if (x == 0 || (Pixels[y][x] != 0 && Pixels[y][x - 1] == 0 && Pixels[y][x + 1] != 0))
//右边界
                       {
                              R_black[y] = x;
                              break;
                       }
                }
                LCenter[y] = (byte)(R_black[y] / 2 + L_black[y] / 2);
         }
  }
```

以下是其他参考代码

```
#pragma once
/**

******************************************************************************************************
********
 *                                                              示例代码
 *                                                           EXAMPLE    CODE
 *
 *                              (c) Copyright 2024; SaiShu.Lcc.; Leo; https://bjsstech.com
 *                                                 版权所属[SASU-北京赛曙科技有限公司]
 *
 *                 The code is for internal use only, not for commercial transactions(开源学习,
请勿商用).
 *                 The code ADAPTS the corresponding hardware circuit board(代码适配百度
Edgeboard-智能汽车赛事版),
 *                 The specific details consult the professional(欢迎联系我们,代码持续更正，敬
请关注相关开源渠道).

******************************************************************************************************
*********
 * @file tracking.cpp
 * @author your name (you@domain.com)
 * @brief  赛道线识别：提取赛道左右边缘数据（包括岔路信息等）
 * @version 0.1
 * @date 2022-02-18
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <fstream>
#include <iostream>
#include <cmath>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "../../include/common.hpp"


using namespace cv;
using namespace std;

class Tracking
{
public:
#define pixel(row, col) imagePath.at<uchar>(row, col)
    vector<POINT> pointsEdgeLeft;            // 赛道左边缘点集（row，column）
    vector<POINT> pointsEdgeRight;           // 赛道右边缘点集
    vector<POINT> widthBlock;                // 色块宽度=终-起（每行）[0,319]
    vector<POINT> inlines;                   // 记录赛道内的异常点
    vector<POINT> corners;                   // 记录赛道边缘的异常点
    double stdevLeft;                        // 边缘斜率方差（左）
    double stdevRight;                       // 边缘斜率方差（右）
    double curvature_left;                   // 边缘曲率（左）
    double curvature_right;                  // 边缘曲率（右）
    POINT garageEnable = POINT(0, 0);        // 车库识别标志：（x=1/0，y=row）
    uint16_t rowCutUp = 10;                  // 图像顶部切行
    uint16_t rowCutBottom = 10;              // 图像底部切行
    int maxblock_global = 0;                 //一帧最宽色块
    int maxblockrow_global = 0;
    Rect box_roi;                            //inlines[0]的 box
    POINT BottomPoint;
    int block_num =0;
    int blocks_num = 0;
    int block_one = 0;
    vector<uint16_t> pointsBlockFar;
    int FindBottomPoint(int right) {
    int index = 0;
    int col = 0;
    if (right) {
        col = pointsEdgeRight[0].y;
        for (int i = 1; i < pointsEdgeRight.size()*0.75; i++) {
            if (pointsEdgeRight[i].y < col) {
                col = pointsEdgeRight[i].y;
                index = i;
            }
        }
        POINT tmp(pointsEdgeRight[index].x, pointsEdgeRight[index].y);
        BottomPoint = tmp;
    }
    else {
        col = pointsEdgeLeft[0].y;
```

```
            for (int i = 1; i < pointsEdgeLeft.size(); i++) {
                if (pointsEdgeLeft[i].y > col) {
                    col = pointsEdgeLeft[i].y;
                    index = i;
                }
            }
            POINT tmp(pointsEdgeRight[index].x, pointsEdgeRight[index].y);
            BottomPoint = tmp;
        }
        return index;
}
        /**
         * @brief  赛道线识别
         *
         * @param isResearch  是否重复搜索
         * @param rowStart  边缘搜索起始行
         * return  赛道：pointsEdgeLeft，pointsEdgeRight,widthBlock，validRows（有效行数），
边缘斜率/方差
         *              车库：garageEnable
         *              岔路：spurroadEnable，spurroad
         * -------注意---------limitWidthBlock------------取值------------
         */
        void trackRecognition(Mat &imageBinary, Scene scene,int parkstep, int parkside) {
          vector<POINT> mblock;
          bool flagStartBlock = true; //  搜索到色块起始行的标志（行）
          bool spurroadEnable = false;
          bool cornerEnable = false;
          int gap = -1;
          int rowStart = ROWSIMAGE - rowCutBottom;

          init_param();
          imagePath = imageBinary.clone();
          FilterbyDE(imagePath);
          switch (scene) {// corner 的 gap 取值
          case Scene::NormalScene:
                gap = 30;
                break;
          case Scene::CateringScene:
                gap = 20;
                break;
          case Scene::ObstacleScene:
                gap = 20;
                break;
          default:
                gap = -1;
          }
          for (int row = rowStart; row > rowCutUp; row--) //  提取 row 范围
          {
            max_width = 0;
            indexWidestBlock = 0;
            block.clear();
            mblock.clear();
            FindBlockinRow(row);
            FindMaxBlockinRow();
            if (flagStartBlock) //  起始行做特殊处理
            {
              if (block.size() == 0 || max_width < COLSIMAGE * 0.25)
                continue;
```

```
                    if (row < ROWSIMAGE / 2) //首行行数限制
                        return;
                    float white_ratio = WhiteRatioinRow(block);
                    //提取赛道边缘
                    if (white_ratio > 0.5 && scene != Scene::ParkingScene) { //比率满足时，障碍物在
赛道内
                        flagStartBlock = false;
                        PushBlock(row, block[0].x, block[block.size() - 1].y);
                    } else { //障碍物在赛道外
                        maxblock_global = max_width;
                        maxblockrow_global = row;
                        flagStartBlock = false;
                        PushBlock(row, block[indexWidestBlock].x,block[indexWidestBlock].y);
                    }
                    spurroadEnable = false;
                }
                else // 其它行色块坐标处理
                {
                    if (block.size() == 0)
                        return;
                    if (maxblock_global < max_width) {
                        maxblock_global = max_width;
                        maxblockrow_global = row;
                    }
#if 1
                    //------------------------------------------------<车库标识识别>---
-----------------------------------------------------------
                //      LOGLN(block.size())
                    if (block.size() > 5 && !garageEnable.x) {
                        int widthThis = 0;              // 色块的宽度
                        int widthVer = 0;               // 当前行色块的平均值
                        vector<int> widthGarage;   // 当前行色块宽度集合
                        vector<int> centerGarage; // 当前行色块质心集合
                        vector<int> indexGarage;   // 当前行有效色块的序号

                        for (int i = 0; i < block.size(); i++) //去噪
                        {
                            widthThis = block[i].y - block[i].x;              // 色块的宽度
                            int center = (block[i].x + block[i].y) / 2; // 色块的质心
                            if (widthThis > 5 &&
                                    widthThis < 50) // 过滤无效色块区域：噪点，只保存 5-50 的色块
                            {
                                centerGarage.push_back(center);
                                widthGarage.push_back(widthThis);
                            }
                        }

                        int widthMiddle = getMiddleValue(widthGarage); // 斑马线色块宽度中值

                        for (int i = 0; i < widthGarage.size(); i++) //滤波
                        {
                            if (abs(widthGarage[i] - widthMiddle) < widthMiddle / 3) {
                                indexGarage.push_back(i);
                            }
                        }
                        // LOGLN(indexGarage.size() );
                        if (indexGarage.size() >= 4) // 验证有效斑马线色块个数
                        {
```

```
                        vector<int> distance;
                        for (int i = 1; i < indexGarage.size(); i++) // 质心间距的方差校验
                        {
                            distance.push_back(widthGarage[indexGarage[i]] -
                                                                widthGarage[indexGarage[i - 1]]);
                        }
                        double var = sigma(distance);
                    //      LOGLN(var);
                        if (var < 5.0) // 经验参数，检验均匀分布
                        {
                            garageEnable.x = 1;                                    // 车库标志使能
                            garageEnable.y = pointsEdgeRight.size(); // 斑马线行序号
                            cornerEnable = true;
                            spurroadEnable = true;
                        }
                    }
                }
                //------------------------------------------------------------------
-----------------------------------------------------
                int last_l = pointsEdgeLeft[pointsEdgeLeft.size() - 1].y;
                int last_r = pointsEdgeRight[pointsEdgeRight.size() - 1].y;
                // 上下行色块的连通性判断
                // if (scene == Scene::CateringScene) {
                //      int start_l = pointsEdgeLeft[0].y;
                //      int start_r = pointsEdgeRight[0].y;
                //      //LOGLN(start_l << start_r);
                //      for (int i = 0; i < block.size(); i++) {
                //          if (block[i].y > start_l && block[i].x < start_r)
                //              mblock.push_back(block[i]);
                //      }
                // }
                for (int i = 0; i < block.size(); i++) {
                    if (block[i].y > last_l && block[i].x < last_r)
                        mblock.push_back(block[i]);
                }


                if (mblock.size() == 0) { // 如果没有发现联通色块，则图像搜索完成，结束任务
                    return;
                } else if (mblock.size() == 1) { // 只存在单个色块，正常情况，提取边缘信息
                    if (mblock[0].y - mblock[0].x < COLSIMAGE / 10) //排除异常短色块
                        continue;
                    PushBlock(row, mblock[0].x, mblock[0].y);
                    // 边缘斜率计算
                    slopeCal(pointsEdgeLeft, pointsEdgeLeft.size() - 1);
                    slopeCal(pointsEdgeRight, pointsEdgeRight.size() - 1);

                    FindCorner(gap);
                    cornerEnable = false;
                    spurroadEnable = false;
                } else if (mblock.size() >1) { // 存在多个色块，则需要择优处理：选取与上一行最
近的色块
                    //方案一：只使用最大的色块
                    //int max_index = 0;
                    //for (int i = 1; i < mblock.size(); i++) {
                    //            if    (mblock[i].y    -    mblock[i].x    >    mblock[max_index].y    -
mblock[max_index].x)
                    //            max_index = i;
```

```
            //}
            //PushBlock(row, mblock[max_index].x, mblock[max_index].y);

            //方案二：  计算比率,满足则合并，否则用 max
            if(scene == Scene::ParkingScene && parkstep == 3){
                if(parkside == 1)
                    PushBlock(row, mblock[mblock.size()-1].x, mblock[mblock.size()-1].y);
                else
                    PushBlock(row, mblock[0].x, mblock[0].y);
            }else if(WhiteRatioinRow(mblock) > 0.8 || garageEnable.x)
                PushBlock(row, mblock[0].x, mblock[mblock.size()-1].y);
            else {
                int max_index = 0;
                for (int i = 1; i < mblock.size(); i++) {
                    if  (mblock[i].y  -  mblock[i].x  >  mblock[max_index].y  -
mblock[max_index].x)
                        max_index = i;
                }
                PushBlock(row, mblock[max_index].x, mblock[max_index].y);
            }
            slopeCal(pointsEdgeLeft, pointsEdgeLeft.size() - 1);
            slopeCal(pointsEdgeRight, pointsEdgeRight.size() - 1);
            // if (!cornerEnable) {
            //      FindCorner(gap);
            //      cornerEnable = true;
            // }
            //------------------------------<岔路信息提取>-------------------
--------------------
            if (!spurroadEnable) {
              for (int i = 1; i < mblock.size(); i++) {
                inlines.push_back(POINT(row, mblock[i].x));
              }
              spurroadEnable = true;
            }
            //------------------------------------------------------------------
------------------
          }
#endif
          stdevLeft = stdevEdgeCal(pointsEdgeLeft, ROWSIMAGE); // 计算边缘方差
          stdevRight = stdevEdgeCal(pointsEdgeRight, ROWSIMAGE);
        }
        GetBlocksNum();
      }
      if(garageEnable.x){
        inlines.clear();
        corners.clear();
      }
    }
    //计算在赛道中的白色占比（当赛道中出现深色障碍物）
    float WhiteRatioinRow(vector<POINT>blocks) {
      int white_width = 0;
      for (int i = 0; i < blocks.size(); i++)
          white_width += blocks[i].y - blocks[i].x;
      int sum_width = blocks[blocks.size() - 1].y - blocks[0].x;
      return (float)white_width / sum_width;
    }
    void PushBlock(int row,int start,int end){
      POINT pointTmp(row, start);
```

```
        pointsEdgeLeft.push_back(pointTmp);
        pointTmp.y = end;
        pointsEdgeRight.push_back(pointTmp);
        widthBlock.emplace_back(row,end-start);
    }
    void init_param(){
        pointsEdgeLeft.clear();                    //  初始化边缘结果
        pointsEdgeRight.clear();                   //  初始化边缘结果
        widthBlock.clear();                        //  初始化色块数据
        corners.clear();                           //  岔路信息
        inlines.clear();
        maxblock_global = 0;
        maxblockrow_global = 0;
        garageEnable = POINT(0, 0);                //  车库识别标志初始化

        block_num =0;
        blocks_num = 0;
        block_one    = 0;
        pointsBlockFar.clear();

        //切行
        if (rowCutUp > ROWSIMAGE / 4)
        rowCutUp = ROWSIMAGE / 4;
        if (rowCutBottom > ROWSIMAGE / 4)
        rowCutBottom = ROWSIMAGE / 4;
    }
    void FindMaxBlockinRow(){
        for (int i = 0; i < block.size(); i++){
                int tmp_width = block[i].y - block[i].x;
                if (tmp_width > max_width){
                        max_width = tmp_width;
                        indexWidestBlock = i;
                }
        }
    }
    //判断最后两个点斜率，如果相反且两点的距离满足条件则提取
    void FindCorner(int width) {
        if (pointsEdgeLeft.size() < 5)return;
        int last_y = pointsEdgeLeft[pointsEdgeLeft.size() - 1].y;
        int current_y = pointsEdgeLeft[pointsEdgeLeft.size() - 2].y;
        float tmp1_slope = pointsEdgeLeft[pointsEdgeLeft.size() - 1].slope;
        float tmp2_slope = pointsEdgeLeft[pointsEdgeLeft.size() - 2].slope;
        if (tmp1_slope * tmp2_slope < 0 && abs(last_y-current_y) > width) {
                corners.push_back(pointsEdgeLeft[pointsEdgeLeft.size() - 1]);
        }
        last_y = pointsEdgeRight[pointsEdgeRight.size() - 1].y;
        current_y = pointsEdgeRight[pointsEdgeRight.size() - 2].y;
        tmp1_slope = pointsEdgeRight[pointsEdgeRight.size() - 1].slope;
        tmp2_slope = pointsEdgeRight[pointsEdgeRight.size() - 2].slope;
        if (tmp1_slope * tmp2_slope < 0 && abs(last_y - current_y) > width) {
                POINT          tmp(pointsEdgeRight[pointsEdgeRight.size()          -          1].x,
pointsEdgeRight[pointsEdgeRight.size() - 1].y);
                corners.push_back(pointsEdgeRight[pointsEdgeRight.size() - 1]);
        }
    }
    void FindBlockinRow(int row) {
        int black_limit = 5;
        int white_limit = 7;
    int start = 0;
    int tmp_end = 0;
    bool flag = false;
    for (int col = 1; col < COLSIMAGE; col++) {
        if (!flag) {
            if (pixel(row, col) > 127 && pixel(row, col - 1) <= 127) {
                start = col;
            }
            else if (pixel(row, col) <= 127 && pixel(row, col - 1) > 127) {
                if (col - start > white_limit) {
                    tmp_end = col;
                    flag = true;
                }
            }
        }
        else {
            if (pixel(row, col) > 127 && pixel(row, col - 1) <= 127) {
                if (col - tmp_end >= black_limit) {//滤波
                    POINT tmp(start, tmp_end);
                    block.push_back(tmp);
                    start = col;
                    flag = false;
                }
            }
            else if (pixel(row, col) <= 127 && pixel(row, col - 1) > 127) {
                tmp_end = col;
            }
        }
    }
    if (pixel(row, COLSIMAGE - 1) > 127 && COLSIMAGE - 1 - start > white_limit) {
            POINT tmp(start, COLSIMAGE - 1);
            block.push_back(tmp);

    }
    else if(pixel(row, COLSIMAGE - 1) < 127 && tmp_end - start > white_limit){
            POINT tmp(start, tmp_end);
            block.push_back(tmp);
    }
}

/**
 * @brief 显示赛道线识别结果
 *
 * @param trackImage 需要叠加显示的图像
 */
void drawImage(Mat& trackImage)
{
    for (int i = 0; i < pointsEdgeLeft.size(); i++)
    {
        circle(trackImage, Point(pointsEdgeLeft[i].y, pointsEdgeLeft[i].x), 1,
            Scalar(0, 255, 0), -1); // 绿色点
    }
    for (int i = 0; i < pointsEdgeRight.size(); i++)
    {
        circle(trackImage, Point(pointsEdgeRight[i].y, pointsEdgeRight[i].x), 1,
            Scalar(0, 255, 255), -1); // 黄色点
    }
```

```
		for (int i = 0; i < inlines.size(); i++)
		{
			circle(trackImage, Point(inlines[i].y, inlines[i].x), 3,
				Scalar(0, 0, 255), -1); //  红色点
		}
		for (int i = 0; i < corners.size(); i++)
		{
			circle(trackImage, Point(corners[i].y, corners[i].x), 3,
				Scalar(178, 102, 255), -1); //粉色点
		}
	circle(trackImage, Point(BottomPoint.y, BottomPoint.x), 7, Scalar(255, 102, 178), 1);
	int blocks = GetSlope0Bottom(0);
	putText(trackImage, to_string(blocks), Point(140, 180), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
	int blocks_r = GetSlope0Bottom(1);
	putText(trackImage, to_string(blocks_r), Point(180, 180), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
	int index = GetEndIndex(0);
	putText(trackImage, to_string(index), Point(20, 200), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(0, 255, 0), 1, cv::LINE_AA);
	index = GetEndIndex(1);
	putText(trackImage, to_string(index), Point(270, 200), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(0, 255, 0), 1, cv::LINE_AA);
	index = GetMiddleIndex(0);
	putText(trackImage, to_string(index), Point(20, 190), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(0, 0, 255), 1, cv::LINE_AA);
	index = GetMiddleIndex(1);
	putText(trackImage, to_string(index), Point(270, 190), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(0, 0, 255), 1, cv::LINE_AA);
	putText(trackImage,	'g'+to_string(garageEnable.x),	Point(100,	190),
cv::FONT_HERSHEY_TRIPLEX, 0.6, cv::Scalar(0, 0, 255), 1, cv::LINE_AA);

	}



	/**
	 * @brief  边缘方差计算
	 *
	 * @param v_edge
	 * @param img_height
	 * @return double
	 */
	double stdevEdgeCal(vector<POINT> &v_edge, int img_height)
	{
		if (v_edge.size() < img_height / 4)
		{
			return 1000;
		}
		vector<int> v_slope;
		int step = 10; // v_edge.size()/10;
		for (int i = step; i < v_edge.size(); i += step)
		{
			if (v_edge[i].x - v_edge[i - step].x)
				v_slope.push_back((v_edge[i].y - v_edge[i - step].y) * 100 / (v_edge[i].x -
v_edge[i - step].x));
		}
		if (v_slope.size() > 1)
		{
			double sum = accumulate(begin(v_slope), end(v_slope), 0.0);
			double mean = sum / v_slope.size(); //  均值
			double accum = 0.0;
			for_each(begin(v_slope), end(v_slope), [&](const double d)
				{ accum += (d - mean) * (d - mean); });

			return sqrt(accum / (v_slope.size() - 1)); //  方差
		}
		else
			return 0;
	}

	double stdevEdgeCal50(vector<POINT> &v_edge, int img_height)
	{
		vector<int> v_slope;
		int step = 10; // v_edge.size()/10;
		for (int i = step; i < v_edge.size(); i += step)
		{
			if (v_edge[i].x - v_edge[i - step].x)
				v_slope.push_back((v_edge[i].y - v_edge[i - step].y) * 100 / (v_edge[i].x -
v_edge[i - step].x));
		}
		if (v_slope.size() > 1)
		{
			double sum = accumulate(begin(v_slope), end(v_slope), 0.0);
			double mean = sum / v_slope.size(); //  均值
			double accum = 0.0;
			for_each(begin(v_slope), end(v_slope), [&](const double d)
				{ accum += (d - mean) * (d - mean); });

			return sqrt(accum / (v_slope.size() - 1)); //  方差
		}
		else
			return 0;
	}
	// -2 全部点集为边际 -1 前半部分无边际 返回正数前半部分为边际   适合做判断算
子
	int GetEndIndex(int right) {
		int index = -1;
		if (right != 0) {
			for (int i = 0; i < pointsEdgeRight.size(); i++) {
				if (pointsEdgeRight[i].y > 317) {
					index = i;

				}
				else if (pointsEdgeRight[i].y < 280) {
					return index;
				}
			}
		}
		else {
			for (int i = 0; i < pointsEdgeLeft.size(); i++) {
				if (pointsEdgeLeft[i].y < 2) {
					index = i;
				}
				else if (pointsEdgeLeft[i].y > 30) {
					return index;
```

```
                        }
                }
        }
        if (index == -1) {
                return -1;
        }
        else
                return -2;
}
//-1  全集无边际  会提取大于 30 宽度的最近边际  适合用作补线算子
int GetMiddleIndex(int right) {
        int index = -1;
        int counter = 0;
        int threshold = 30;

        if (right != 0) {
                for (int i = 0; i < pointsEdgeRight.size(); i++) {
                        if (pointsEdgeRight[i].y > 317) {
                                index = i;
                                counter++;
                        }
                        else if (counter != 0) {
                                if (counter > threshold)return index;
                                counter = 0;
                                index = -1;
                        }
                }
        }
        else {
                for (int i = 0; i < pointsEdgeLeft.size(); i++) {
                        if (pointsEdgeLeft[i].y < 2) {
                                index = i;
                                counter++;
                        }
                        else if (counter != 0) {
                                if (counter > threshold)return index;
                                counter == 0;
                        }
                }
        }
        if (threshold > 10)
                return index;
        else
                return -1;
}


int GetSlope0Bottom(int right) {
        int counter = 0;
        int rows = 120;
        if (right) {
                if (pointsEdgeRight.size() < rows)return -1;
                for (int i = 0; i < rows; i++) {
                        if (pointsEdgeRight[i].slope == 0)
                                counter++;
                }
        }
        else {
```

```
			if (pointsEdgeLeft.size() < rows)return -1;
			for (int i = 0; i < rows; i++) {
				if (pointsEdgeLeft[i].slope == 0)
					counter++;
			}
		}
		return counter;

	}
	void AdjustbyParking(int right){
		if (right && !pointsEdgeRight.empty()) {   // 增加非空检查
			vector<POINT> tmp_pointsEdgeRight;
			tmp_pointsEdgeRight.push_back(pointsEdgeRight[0]);   // 先保存第一个点
			int tmp_y = pointsEdgeRight[0].y;

			// 遍历所有元素（修正循环条件）
			for (size_t i = 1; i < pointsEdgeRight.size(); i++) {
				if (pointsEdgeRight[i].y <= tmp_y) {   // 保留 y 值更小的点
					tmp_pointsEdgeRight.push_back(pointsEdgeRight[i]);
					tmp_y = pointsEdgeRight[i].y;
				}
			}
			pointsEdgeRight = tmp_pointsEdgeRight;   // 更新原始数组
		}else if(!right && !pointsEdgeLeft.empty()){
			vector<POINT> tmp_pointsEdgeleft;
			tmp_pointsEdgeleft.push_back(pointsEdgeLeft[0]);   // 先保存第一个点
			int tmp_y = pointsEdgeLeft[0].y;

			// 遍历所有元素（修正循环条件）
			for (size_t i = 1; i < pointsEdgeLeft.size(); i++) {
				if (pointsEdgeLeft[i].y >= tmp_y) {   // 保留 y 值更小的点
					tmp_pointsEdgeleft.push_back(pointsEdgeLeft[i]);
					tmp_y = pointsEdgeLeft[i].y;
				}
			}
			pointsEdgeLeft = tmp_pointsEdgeleft;   // 更新原始数组
		}
	}
	int CalculateRoiAverageGray(const POINT& roiPoint, int width=30 , int hight=80 ) {
		// roiPoint  为下边的中点
		mean = 0;
		hight = roiPoint.x / 3;
		int col = roiPoint.y - width/2;
		int row = roiPoint.x - hight;

		// 创建 ROI 矩形（确保不超出图像边界）
		cv::Rect tmp(
			std::max(0, col),                        // x 坐标
			std::max(0, row),                        // y 坐标
			std::min(width, 319 - col),              // 宽度
			std::min(hight, 239 - row)               // 高度（确保不超出图像底部）
		);
		box_roi = tmp;
		// 提取 ROI 并计算平均灰度值
		cv::Mat roiImage = imagePath(box_roi);
		cv::Scalar meanValue = cv::mean(roiImage);
		mean = meanValue[0];
		return meanValue[0];   // 返回单通道图像的平均灰度值
```

```
        }

        void DrawRect(Mat& Image) {
                putText(Image, to_string(mean), Point(160, 200), cv::FONT_HERSHEY_TRIPLEX, 0.6,
cv::Scalar(0, 0, 255), 1, cv::LINE_AA);
                cv::rectangle(Image, box_roi, cv::Scalar(0, 0, 255), 2, cv::LINE_AA);
        }

void GetBlocksNum()
{
        if(block.size() > 1)
        {
                block_num ++;
        }
        else
        {
                block_one ++;
                if(block_one >= 2)
                {
                        if(block_num >= 4)
                        {
                        //block_num = 0;
                                blocks_num ++;
                                pointsBlockFar.push_back((pointsEdgeLeft[pointsEdgeLeft.size()  -  1].x  +
pointsEdgeRight[pointsEdgeRight.size() - 1].x) / 2);
                        }
                        block_num = 0;
                        block_one = 0;
                }
                else
                {
                        block_num ++;
                }
        }
}

private:
        Mat imagePath; // 赛道搜索图像
        vector<POINT>block;
        int max_width; // 一行中最大色块
        int indexWidestBlock; // 最宽色块索引
        Mat k_e = getStructuringElement(MORPH_RECT, Size(3, 3)); //腐蚀核
        Mat k_d = getStructuringElement(MORPH_RECT, Size(3, 3)); //膨胀核
        int    mean = 0;
        void FilterbyDE(Mat& img) {
            cv::dilate(img, img, k_d, cv::Point(-1, -1), 1); //膨胀
            cv::erode(img, img, k_e, cv::Point(-1, -1), 1);    //腐蚀
        }
        /**
          * @brief  边缘斜率计算
          *
          * @param edge
          * @param index
          */
        void slopeCal(vector<POINT> &edge, int index)
        {
                if (index <= 4)
                {
```

```
        return;
    }
    float temp_slop1 = 0.0, temp_slop2 = 0.0;
    if (edge[index].x - edge[index - 2].x != 0)
    {
        temp_slop1 = (float)(edge[index].y - edge[index - 2].y) * 1.0f /
                     ((edge[index].x - edge[index - 2].x) * 1.0f);
    }
    else
    {
        temp_slop1 = edge[index].y > edge[index - 2].y ? 255 : -255;
    }
    if (edge[index].x - edge[index - 4].x != 0)
    {
        temp_slop2 = (float)(edge[index].y - edge[index - 4].y) * 1.0f /
                     ((edge[index].x - edge[index - 4].x) * 1.0f);
    }
    else
    {
        edge[index].slope = edge[index].y > edge[index - 4].y ? 255 : -255;
    }
    if (abs(temp_slop1) != 255 && abs(temp_slop2) != 255)
    {
        edge[index].slope = (temp_slop1 + temp_slop2) * 1.0 / 2;
    }
    else if (abs(temp_slop1) != 255)
    {
        edge[index].slope = temp_slop1;
    }
    else
    {
        edge[index].slope = temp_slop2;
    }
}


/**
 * @brief 冒泡法求取集合中值
 *
 * @param vec 输入集合
 * @return int 中值
 */
int getMiddleValue(vector<int> vec)
{
    if (vec.size() < 1)
        return -1;
    if (vec.size() == 1)
        return vec[0];

    int len = vec.size();
    while (len > 0)
    {
        bool sort = true; // 是否进行排序操作标志
        for (int i = 0; i < len - 1; ++i)
        {
            if (vec[i] > vec[i + 1])
            {
                swap(vec[i], vec[i + 1]);
```

```
                    sort = false;
                }
            }
            if (sort) //  排序完成
                break;

            --len;
        }

        return vec[(int)vec.size() / 2];
    }
};



#pragma once


#include <fstream>
#include <iostream>
#include <cmath>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "../../include/common.hpp"
#include "tracking.cpp"


using namespace cv;
using namespace std;

class Crossroad
{
public:
    enum CrossStep{
        None=0,
        Fix,
    };
    enum CrossType {
        Left = 0,
        Right,
        Mid
    };
    CrossStep step = None;
    CrossType type = Left;
    POINT TopPoint;
    POINT LeftBottomPoint;
    POINT RightBottomPoint;
    bool process(Tracking& track);
    void draw_img(Mat& img);
    int FindBottomPoint(Tracking& track, int right);
    int FindBottomPoint2(Tracking& track, int right);
    int WholeBlock(Tracking track);
private:
    POINT startpoint = POINT(0, 0);
    POINT midpoint = POINT(0, 0);
    POINT endpoint = POINT(0, 0);
    int width_counter = 0;
    int memory = 0;
```

```
	vector<POINT> bezier_tmp;
	bool right_flag = false;
	bool left_flag = false;
};


bool Crossroad::process(Tracking& track) {
	switch (step) {
		case CrossStep::None: {
			if (!track.inlines.empty()) {
				int bottom_right_index = FindBottomPoint(track, 1);
				int bottom_left_index = FindBottomPoint(track, 0);
				if (right_flag || left_flag) {// CASE 1
					int left_endindex = 0;
					int right_endindex = 0;
					int left_mid = 0;
					int right_mid = 0;
					left_endindex = track.GetEndIndex(0);
					right_endindex = track.GetEndIndex(1);
					left_mid = track.GetMiddleIndex(0);
					right_mid = track.GetMiddleIndex(1);
					if (right_flag && (left_endindex > 100 || left_mid > 100) &&
right_endindex < 60 && track.inlines[0].y < 100 && track.inlines[0].x < 120) {//右斜入
						LOGLN("RIGHT CASE 1");
						memory = track.inlines[0].y;
						type = CrossType::Right;
						step = CrossStep::Fix;
					}
					else if (left_flag && (right_endindex > 100 || right_mid > 100) &&
left_endindex < 60 && track.inlines[0].y > 220 && track.inlines[0].x < 120) {//左斜入
						LOGLN("LEFT CASE 1");
						memory = track.inlines[0].y;
						type = CrossType::Left;
						step = CrossStep::Fix;
					}
					else if (left_mid > 140 && right_mid > 140 && abs(track.stdevLeft -
track.stdevRight) < 100) {//直入
						LOGLN("MID CASE 1");
						type = CrossType::Mid;
						step = CrossStep::Fix;
					}
				}

			}
			break;
		}
		case CrossStep::Fix: {
			int blockline = WholeBlock(track);
			if(type == CrossType::Right){
				if (!track.inlines.empty()) {
					int bottom_right_index = FindBottomPoint2(track, 1);
					if (bottom_right_index == -1) {
						track.pointsEdgeRight.clear();
						track.pointsEdgeRight = bezier_tmp;
						track.pointsEdgeLeft.resize(track.pointsEdgeRight.size());
					}
					else {
						if (memory - track.inlines[0].y > 50)
```

```
                            track.inlines[0].y = COLSIMAGE - track.inlines[0].y;
                        startpoint = track.pointsEdgeRight[0];
                        endpoint = track.inlines[0];
                        midpoint = track.pointsEdgeRight[bottom_right_index];
                        vector<POINT> input = { startpoint, midpoint, endpoint };
                        bezier_tmp.clear();
                        bezier_tmp = Bezier(0.01, input);
                        track.pointsEdgeRight.clear();
                        track.pointsEdgeRight = bezier_tmp;
                        track.pointsEdgeLeft.resize(track.pointsEdgeRight.size());
                    }
                    memory = track.inlines[0].y;
                    if (track.inlines[0].y > 220 || width_counter > 40) {
                        LOGLN("RIGHT EXIT:"<<track.inlines[0].y<<' '<<width_counter);
                        step = CrossStep::None;
                    }
                }
            }
            else if(type == CrossType::Left){
                if (!track.inlines.empty()) {
                    int bottom_left_index = FindBottomPoint2(track, 0);
                    if (bottom_left_index == -1) {
                        track.pointsEdgeLeft.clear();
                        track.pointsEdgeLeft = bezier_tmp;
                        track.pointsEdgeRight.resize(track.pointsEdgeLeft.size());
                    }
                    else {
                        if (memory - track.inlines[0].y < -50)
                            track.inlines[0].y = COLSIMAGE - track.inlines[0].y;
                        startpoint = track.pointsEdgeLeft[0];
                        endpoint = track.inlines[0];
                        midpoint = track.pointsEdgeLeft[bottom_left_index];
                        LOGLN(bottom_left_index);
                        vector<POINT> input = { startpoint, midpoint, endpoint };
                        bezier_tmp.clear();
                        bezier_tmp = Bezier(0.01, input);
                        track.pointsEdgeLeft.clear();
                        track.pointsEdgeLeft = bezier_tmp;
                        track.pointsEdgeRight.resize(track.pointsEdgeLeft.size());
                    }
                    memory = track.inlines[0].y;
                    if (track.inlines[0].y < 100 || width_counter > 40) {
                        LOGLN("LEFT EXIT:"<<track.inlines[0].y<<' '<<width_counter);
                        step = CrossStep::None;
                    }
                }
            }
            else {
                if (blockline == -1)return;
                int left_mid = 0;
                int right_mid = 0;
                left_mid = track.GetMiddleIndex(0);
                right_mid = track.GetMiddleIndex(1);
                if ((left_mid < 60 && right_mid < 60) || track.widthBlock[blockline].x > 160)
{
                    LOGLN("MID EXIT:"<<' '<<track.widthBlock[blockline].x);
                    step = CrossStep::None;
                }
            }
            break;
        }
    }

}
void Crossroad::draw_img(Mat& img) {
    circle(img, Point(RightBottomPoint.y, RightBottomPoint.x), 3, Scalar(255, 106, 106), -1);
//红色点
    circle(img, Point(LeftBottomPoint.y, LeftBottomPoint.x), 3, Scalar(255, 255, 106), -1); //红
色点
    putText(img,    "width    "    +    to_string(width_counter),    Point(240,    220),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(255, 106, 106), 1, cv::LINE_AA);
    putText(img, "flag " + to_string(right_flag), Point(220, 200), cv::FONT_HERSHEY_TRIPLEX,
0.3, cv::Scalar(255, 106, 106), 1, cv::LINE_AA);
    putText(img, "flag " + to_string(left_flag), Point(100, 200), cv::FONT_HERSHEY_TRIPLEX,
0.3, cv::Scalar(255, 106, 106), 1, cv::LINE_AA);
    putText(img, "step "+to_string(step), Point(160, 200), cv::FONT_HERSHEY_TRIPLEX, 0.3,
cv::Scalar(255, 106, 106), 1, cv::LINE_AA);
    putText(img, "type " + to_string(type), Point(160, 220), cv::FONT_HERSHEY_TRIPLEX, 0.3,
cv::Scalar(255, 106, 106), 1, cv::LINE_AA);
}
int Crossroad::FindBottomPoint(Tracking& track,int right) {
    int index = 0;
    int col = 0;
    int counter = 0;
    if (right) {
        RightBottomPoint = POINT(0, 0);
        col = track.pointsEdgeRight[0].y;
        for (int i = 1; i < track.pointsEdgeRight.size(); i++) {
            if (track.pointsEdgeRight[i].y < col) {
                col = track.pointsEdgeRight[i].y;
                index = i;
            }
            else if (track.pointsEdgeRight[i].y > col + 5 && track.pointsEdgeRight[i].y < col
+ 50) {
                RightBottomPoint = track.pointsEdgeRight[index];
                col = track.pointsEdgeRight[i].y;
                index = i;
                break;
            }
            else if (track.pointsEdgeRight[i].y >= col + 50) {
                right_flag = false;
                return -1;
            }
        }
        for (int i = index; i < track.pointsEdgeRight.size(); i++) {
            if (track.pointsEdgeRight[i].y > col) {
                col = track.pointsEdgeRight[i].y;
                counter++;
                if (counter > 15) {
                    right_flag = true;
                    return index;
                }
            }
        }
        return -1;
    }
```

```
	else {
		LeftBottomPoint = POINT(0, 0);
		col = track.pointsEdgeLeft[0].y;
		for (int i = 1; i < track.pointsEdgeLeft.size(); i++) {
			if (track.pointsEdgeLeft[i].y > col) {
				col = track.pointsEdgeLeft[i].y;
				index = i;
			}
			else if (track.pointsEdgeLeft[i].y < col - 5 && track.pointsEdgeLeft[i].y > col - 
50) {
				LeftBottomPoint = track.pointsEdgeLeft[index];
				col = track.pointsEdgeLeft[i].y;
				index = i;
				break;
			}
			else if (track.pointsEdgeLeft[i].y <= col - 50) {
				left_flag = false;
				return -1;
			}
		}
		for (int i = index; i < track.pointsEdgeLeft.size(); i++) {
			if (track.pointsEdgeLeft[i].y < col) {
				col = track.pointsEdgeLeft[i].y;
				counter++;
				if (counter > 15) {
					left_flag = true;
					return index;
				}
			}
		}
		return -1;
	}
	return index;
}
int Crossroad::FindBottomPoint2(Tracking& track, int right) {
	int index = 0;
	int col = 0;
	if (right) {
		col = track.pointsEdgeRight[0].y;
		for (int i = 1; i < track.pointsEdgeRight.size(); i++) {
			if (track.pointsEdgeRight[i].y <= col) {
				col = track.pointsEdgeRight[i].y;
				index = i;
			}
			else {
				return i;
			}
		}
	}
	else {
		col = track.pointsEdgeLeft[0].y;
		for (int i = 1; i < track.pointsEdgeLeft.size(); i++) {
			if (track.pointsEdgeLeft[i].y >= col) {
				col = track.pointsEdgeLeft[i].y;
				index = i;
			}
			else {
				return i;
				}
			}
		}
		return -1;
	}
	int Crossroad::WholeBlock(Tracking track) {//  输出最远一行的 index
		int index = 0;
		width_counter = 0;
		for (int i = 0; i < track.widthBlock.size(); i++) {
			if (track.widthBlock[i].y == 319) {
				width_counter++;
				index = i;
			}else if (width_counter != 0) {
				if (width_counter > 40) {
					return index;
				}
				else
					width_counter = 0;
			}
		}
		return -1;
	}
```