**第十九届全国大学生智能汽车竞赛**

**室外远程驾驶无人车赛项目**

**技 术 报 告**

学 校：

队伍名称：

参赛队员：

带队教师：

**引言**

随着科学技术的发展，智能汽车已经成为未来汽车产业发展的战略方向。无人驾驶技术的应用范围广泛。例如在智慧城市领域，无人清洁车、配送车极大提升了市民的生活质量和城市的管理效率，在环境检测方面，无人车也可以减少大量的人力物力，24小时不间断地搜集环境数据信息。

如今，在人工智能和自动驾驶技术的快速发展之下，无人驾驶汽车技术也受到了广泛关注。然而，要实现真正意义上的全自动驾驶仍面临诸多挑战，如复杂环境下的道路识别和物体检测，定位导航等难题。本报告旨在充分利用现有的5G 的通信技术，实现对无人车的远程驾驶和创新应用，结合对计算机视觉、人工智能、自动控制和远程通信等多种技术综合应用能力探索无人驾驶技术的可能性。通过基于树莓派平台构建智能模型车进行应用，并优化相关算法以实现车辆自主行驶功能。首先，我们将对现有智能车辆领域的关键技术进行全面回顾，涵盖变道算法及道路识别算法等方面的算法。接着详细介绍了本次比赛中各个任务解决时遇到的问题、解决思路以及相关代码实现、测试效果等内容。

尽管近年来无人驾驶领域取得了不小的突破，但是还是有着安全性和可靠性、精确识别与定位等问题。本技术报告聚焦于无人车自主驾驶过程中遇到的计算机视觉和自动控制问题，详细叙述并实现了一套解决方案。同时，对不同方案进行了分析与测试，以期为相关领域的研究提供有价值的参考。

**目 录**

[目 录 4](#_Toc30648)

[第1章 方案概述 5](#_Toc7974)

[第2章 问题描述 7](#_Toc4362)

[1. 斑马线识别 7](#_Toc10107)

[2. 道路识别 8](#_Toc28156)

[3. 变道算法 10](#_Toc21094)

[4. 红色锥桶绕行 10](#_Toc15636)

[5. 停车入库 11](#_Toc13879)

[第3章 技术方案 13](#_Toc23671)

[1. 环境配置部分 13](#_Toc15663)

[2. 视觉处理部分 13](#_Toc18251)

[(1) 蓝色挡板的实现： 13](#_Toc28704)

[(2) 道路检测的实现： 13](#_Toc31830)

[(3) 锥桶避障的实现： 13](#_Toc28422)

[(4) 变道的实现： 14](#_Toc3755)

[(5) 入库的实现： 14](#_Toc20366)

[3. 控制部分 14](#_Toc22690)

[第4章 方案实现 15](#_Toc6302)

[1. 环境配置部分解决方案 15](#_Toc19376)

[2. 视觉处理部分解决方案 15](#_Toc2909)

[(1) 蓝色挡板方案实现： 15](#_Toc27607)

[(2) 道路检测方案实现 16](#_Toc11605)

[(3) 锥桶避障方案实现 19](#_Toc15304)

[(4) 变道方案实现 21](#_Toc25748)

[(5) 入库方案实现 22](#_Toc18162)

[3. 控制部分解决方案 24](#_Toc15874)

[第5章 测试分析 25](#_Toc8456)

[第6章 作品总结 28](#_Toc30534)

[附录A 29](#_Toc11979)

# 方案概述

本次项目中，大部分代码采用C++编写，部分图像检测功能则利用Python实现。所有图像均调整至320x240像素大小。混合代码的使用，充分发挥了C++的高效性能及Python在视觉检测方面的优势。在未采用多线程技术的情况下，系统仍能维持约30帧/秒的处理速度，且图像获取过程十分流畅。

在蓝色挡板识别方面，我们采用了转换颜色空间、截取感兴趣区域对蓝色的面积进行计算，得到了一个蓝色面积大小的阈值，通过蓝色面积大小来找蓝色挡板和检测蓝色挡板是否移开，经过测试效果也很好。

在道路识别方面，我们采用边缘检测和霍夫直线进行识别道路，通过调参以及截取区域，使得能够很清晰地画出道路的两侧，再通过一定的算法进行中点的计算，以此来更改舵机的转向。

在斑马线识别方面，我们采用图像二值化的方法，然后通过扫描截取的区域，设置至少应该检测到的斑马线组数以及宽度，达到这些阈值就识别为斑马线，经过在实地的测试，这个方法十分准确，不会误识别斑马线。

在标志识别方面，我们是通过分阶段识别红色图像来判断是否开启识别模块的，在C++中使用系统命令和重定向机制运行标志识别脚本，并把识别的结果重定向到一个文件中，再使用c++对文件进行读取，并返回识别结果。

在变道方面，我们采用的基本思路是，打一个对应方向的转向角，然后再直行，到了另一条跑道再让小车正常巡线。不过不同于简单的设置打角多少时间，我们通过一些算法，使得小车能够自动地判断什么时候应该打角度，什么时候应该直行，增加了变道的稳健性。经过测试，如果角度不是太刁钻，小车就能够自主进行变道。

在红色锥桶绕行方面，我们使用红色识别并返回红色物体坐标，通过这个坐标使得舵机偏转一个角度，再用一个变量来控制是左绕行还是右绕行，同时也使用了一定的算法来保证不会将同一个红色锥桶识别多次，经过测试，能够很稳定地进行绕行，并对锥桶的个数统计也是正确的。

在停车入库方面我们通过二值化来获取黄色，并统计其面积，发现在入库时黄色面积有由大变小再变大的趋势，以此来统计黄色长条的个数，在检测到第二个黄色长条的时候进行停车并结束运行。然后结合上一次的识别结果以及通过巡线获取的中点位置来调节舵机的转向，最终成功入库。经过测试，入库也能稳定地运行，并且能够自主停车。

# 问题描述

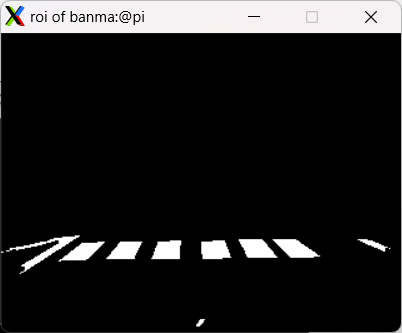
这里对斑马线、道路识别、变道算法、红色锥桶绕行以及停车入库这五个方面在处理过程中遇到的问题进行描述以及提出我们所提供的算法实现。

## 斑马线识别

在斑马线识别方面，我们观察到，斑马线是由一组一组长条状的白线组成的，而且它的宽度大概是道路白线宽度的两倍。抓住这些特点，我们队伍采用了颜色识别，将白色部分保留，其余非白色部分则为黑色，得到了一张检测白色的黑白图，通过保证白色长条的宽度不能低于5个像素和组数不能少于3这个限制（这些参数都是经过测试后给出的），来成功识别斑马线。通过控制识别区域的截取来控制小车能在30cm内停车。可结合下图进行理解：



未找到斑马线时



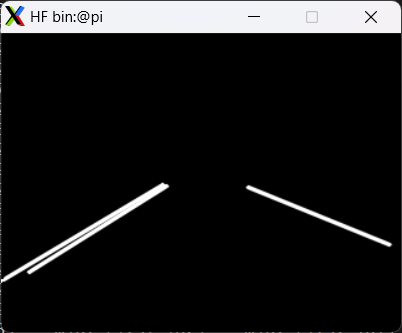
找到斑马线时

## 道路识别

道路识别方面，我们队伍一开始是采用和斑马线一样的思路，就是将图像转化为保留白线的二值化图片。可是经过测试后发现，这种方法受光线影响很大，再加上摄像头拍摄的图片较暗，在光线不好的情况下很难检测出道路，如果要检测出道路就要进行不断地调参，十分麻烦，于是放弃这种方案而采用边缘检测加上霍夫直线来检测道路。

为了避免除了道路以外的其它区域的干扰，我们队伍截取了一个类似于梯形的区域，只对这个区域进行边缘检测等处理，这大大减少了外部的影响。在进行霍夫检测的时候，虽然经过参数的调节使得检测出来的直线大幅减少了，不过还是有部分不必要的直线存在，比如一些比较垂直的线和比较平的直线，因此，我们使用计算直线的角度来进一步过滤直线，使得最终检测出来的直线质量会比较高，很少出现杂线的干扰。

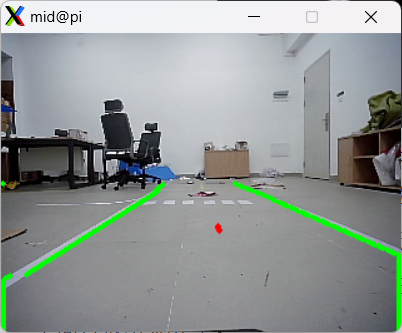
在获得上一步过滤的直线之后，我们在一个纯黑的图像上将这些直线以白线的方式画出，再将画出的图像传入中值处理函数进行处理并获取中间值。如下图：



经过霍夫直线检测的道路图

在中值处理函数中，使用了三个数组，分别存储了左边的点、中间的点和右边的点。从图像底部的行往上进行遍历，再从中间往两边找左边的点和右边的点，每一行结束之后，根据这一行找到的左点和右点计算出这一行的中间点，并存入中间点数组mid中。

由于中间点的数组的大小比较大，全部统计算均值不仅慢而且没有太大必要，因此我们打算截取部分的点来计算中点。经过测试，截取数组中间的点较为接近道路的真实中点，因此，我们截取了中点数组中间的五个点，对他们求均值后进行返回，得到道路的中点值，通过获得的中点值结合pid就可以改变舵机的转向，从而使小车可以稳定地行驶在道路中央。下面是道路检测的效果图：



绿色的线为两边的道路，红色的点代表中点

## 变道算法

在处理变道的时候的基本思路是，先让小车往左或者往右偏一个角度，然后再直行一段时间，等到小车差不多进入另一个道路的时候再恢复正常巡线的方法，恢复正常巡线之后，小车就会自己在这个道路上调整回来，从而实现成功变道。这里有两个问题，小车要往左往右开多久转为直行状态呢？小车要直行多久之后恢复正常巡线呢？针对这两个问题，我们队提出来一种基于函数连续性的变道算法，在这个算法下，小车能够自动控制什么时候转为直行状态还有什么时候恢复正常巡线。

基于函数连续性的变道算法的思路和上面差不多，不过能够使小车自动调节转向和直行的时间，这个算法的详细思路将会在第4章技术方案中的“基于函数连续性的变道算法”部分中给出。

## 红色锥桶绕行

红色锥桶的绕行有几个难点：

1. 如何控制小车的连续绕行时的方向不同
2. 如何通过红色锥桶的位置来控制小车的偏向
3. 如何不把一个锥桶识别为多个锥桶，也即识别锥桶个数准确性如何实现？

第一个难点我们采用了一个变量来控制即可解决。第二个难点我们采用红色识别并且返回红色物体的纵坐标，通过160与这个坐标做一定的差值来保证小车的偏向准确性。第三个问题是锥桶绕行中最难的部分，也就是不让小车将一个锥桶识别多次，如果识别多次，会对小车绕行的方向有所影响。

针对第三个难点，我们原本打算的是如果小车下一次没有识别到锥桶，就当做这个锥桶已经绕行结束。不过经过我们测试发现，小车在绕行的过程中，可能第一秒识别到锥桶进行转向，由于转向的原因，下一秒摄像头可能就没拍到锥桶，而下一秒转向回来又可能拍到锥桶，这种情况下小车会把同一个锥桶识别多次，从而影响下一次绕行方向的判断。

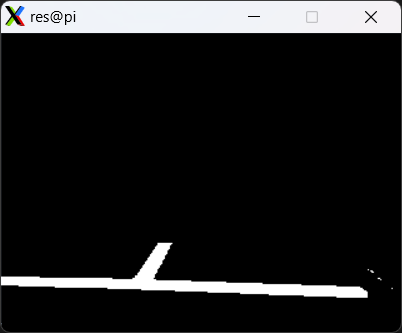
最终，我们想出了一种解决方案，由于同一个锥桶识别的话，它只会越来越靠近图片的底部，这时，如果这个时候有一个新的锥桶出现的话，它一定是在图像上方区域出现的，因此，我们用一个变量来存储上一次锥桶出现的横坐标，如果这次的横坐标减去上一次的横坐标大于一个阈值的话，就认为是新的红色锥桶。经过测试，这种方案成功1解决了同一个锥桶被识别多次的情况。

## 停车入库

停车入库的难题是如何控制小车的偏向以及如何判断已经成功入库并停止。

对于控制偏向这个问题，我们采用的是，根据识别到的A和B来将小车前进的目标点从道路中点变为道路中点偏移一定的像素位，经过测试，这个偏移的像素值为40时比较好。

而判断入库是否成功是依靠识别黄线来实现的。首先，我们截取了一个长320，宽48的长条状区域，根据我们的观察，看到小车刚入库的时候是能够识别到一个长条的横向黄线的，而在入库的过程中，只能识别到一根竖直的短线，而在入库快要结束的时候会再次识别到一个长条的横向黄线，也即，入库的时候黄色的面积是由多到少再变多的，因此在第二次黄色面积突然增大的时候就进行停车。可结合下述图片进行理解：



小车驶入库时检测到的黄线



小车入库过程中检测到的黄线



小车快到库终点时检测到的黄线

# 技术方案

## 环境配置部分

1. 本项目的代码结合了C++和Python，在物体检测部分使用Python，而在其余视觉、控制部分使用C++ ，充分发挥了两种语言的优势。
2. 在使用python的时候，将其所有的库都下载到python 3.7.3这个版本下，由于树莓派默认的python指向是python2，因此需要删除此软链接并将python新的指向改为python3.7.3。

## 视觉处理部分

### 蓝色挡板的实现：

将图像的指定区域对蓝色图像进行二值化掩膜处理，再经过膨胀腐蚀等操作来凸显出蓝色。经过上述操作蓝色部分会保留为白色而其余颜色会保留为黑色，此时对白色的面积进行计算，如果面积大于一定的值，则判断有蓝色挡板。在识别到蓝色挡板的前提下如果面积小于一定的值则认为蓝色挡板移开，进行发车。

### 道路检测的实现：

道路检测是使用边缘检测加上霍夫直线检测进行实现的。

进行边缘检测前，首先对原图像进行部分区域的截取，避免与跑道无关的噪点的影响，然后对截取的区域进行边缘检测和膨胀腐蚀，再送入霍夫直线检测中，得到一系列线的集合。通过对线的角度进行进一步筛选来减少一些杂线。经过多次测试之后，我们可以得到较好的跑道线。

### 锥桶避障的实现：

通过对红色锥桶进行红色检测，并将检测到的红色物体的纵坐标进行返回，再将中线更改为锥桶纵坐标向左或者向右偏移几个像素来实现锥桶的绕行。对于锥桶个数的判断，我们使用一个值来记住上一次红色物体的横坐标，如果当前红色物体的横坐标大于上一次红色物体横坐标一定的值，则认为出现了一个新的锥桶。经过测试，对锥桶个数的统计是无误的。

### 变道的实现：

基本的思路是让小车打一个偏转角度再直行到另一个道上，然后再恢复正常巡线。这里关键在于打偏转角度的时间和直行时间的控制，由于小车变道前运行的方向具有不确定性，直接设定打偏转角度的时间和直行时间有些过于依靠运气，因此，我们队伍采用了一种基于函数连续性的变道算法，能够让小车无论上次以何种方向运行都能稳定地变道到另一个车道上去。

这个算法的基本思路如下：

在同一车道时，道路中点的位置是关于时间的函数，具有连续性，且其一次导的数值在一定的范围内。在变道时，我们让小车根据需要将舵机打左或者打右，当某时刻检测出的道路中点位置超过了一定的阈值，我们可以认为小车已经朝向既定方向，此时让其回正方向。在这个过程中，道路中点的位置会越来越大，或越来越小，当小车到达另一车道时，会出现检测出的道路中点突然变小或突然变大的情况，根据函数的连续性判断此时小车已经将检测的道路切换到了目标车道，此时让其正常巡线，变道完成。

### 入库的实现：

这里检测的颜色为黄色。首先我们截取了一个长320，宽48的长条形区域，在这个区域内对黄线进行颜色检测，由于入库前和入库后的黄线是大面积的长条形，入库中的黄线区域只有一小部分，因此我们当第一次检测到大面积的黄色时，将黄色数量置为1，而在第二次检测到大面积黄色的时候将黄色数量置为2，并让小车停车。

## 控制部分

1. 远程网站控制

通过远程网站的电机和舵机的参数进行调节，最终可以达到流畅操作的效果。

1. PID调节控制

我们采用的是位置式PID，其优点为：算法简单，鲁棒性好，可靠性高。通过对其PID进行调节，最终得到一个能使小车稳定前行的参数。

# 方案实现

## 环境配置部分解决方案

修改python版本为3.7.3

1. 首先，删除原有链接
2. rm /usr/bin/python
3. 其次，找到python3的安装路径



1. 最后，建立python3到python的软链接
2. ln -s  /usr/local/bin/python3  /usr/bin/python

## 视觉处理部分解决方案

### 蓝色挡板方案实现：

首先对传入的图像使用 blue\_card\_process函数进行处理，对限定区域的蓝色进行识别，有蓝色则设置为白色，其他颜色为黑色：

1. Mat blue\_card\_process(Mat frame){
2. Mat frame\_a = frame;//深拷贝一帧
3. cvtColor(frame\_a, frame\_a, COLOR\_BGR2HSV);
4. Mat mask;
5. Scalar scalarl = Scalar(100, 43, 46);
6. Scalar scalarH = Scalar(124, 255, 255);
7. inRange(frame\_a, scalarl, scalarH, mask);//用于根据给定的颜色范围生成二值掩膜  mask 是输出的二值图像，只有在指定的颜色范围内的像素会被设置为白色（255），其他像素为黑色（0）
8. Mat kernel = getStructuringElement(MORPH\_RECT, Size(5, 5));//生成一个结构元素，用于形态学操作。
9. morphologyEx(mask, mask, MORPH\_OPEN, kernel);
10. morphologyEx(mask, mask, MORPH\_CLOSE, kernel);
11. **for** (**int** y = 0; y < mask.rows; y++)
12. {
13. **for** (**int** x = 0; x < mask.cols; x++)
14. {
15. **if** (y < 70 || y > 120)//只保留行索引在 70 到 120 之间的像素，其余的像素都将被设置为 0
16. {
17. mask.at<uchar>(y, x) = 0;//将当前像素的值设置为 0（黑色），表示该区域在掩膜中被去除。
18. }
19. }
20. }
21. **return** mask;
22. }

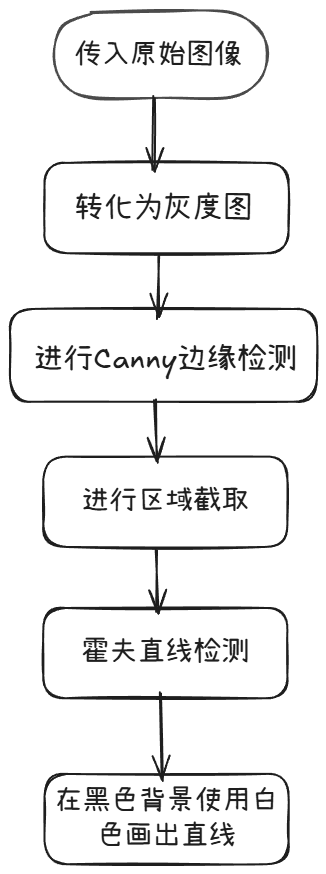
然后使用blue\_card\_find函数对返回的图像的白色面积进行计算，经过测试，若白色面积的大小超过10000时，则认为有蓝色挡板。

在找到蓝色挡板的基础之上，再调用blue\_card\_remove函数，同样计算图像的白色面积，当面积小于5000的时候认为蓝色挡板移除。

### 道路检测方案实现

首先先对传入的图像转为灰度图并进行Canny边缘检测，然后使用region\_selection函数对经过边缘检测的道路区域进行截取，防止与道路无关的区域对后续造成不必要的影响。然后将截取的区域传入detect\_lines函数中应用霍夫直线检测得到线集，再对线集进行过滤并返回。接着将返回的点使用draw\_lines\_on\_binary\_image函数画在一个新建的黑色背景图上，线使用白色画出，这个图像就是画出来的道路图。

程序流程图如下：



代码如下：

1. Mat frame\_processorByHF(Mat image, **int** cnt) {
2. Mat grayscale;
3. cvtColor(image, grayscale, COLOR\_BGR2GRAY);
5. **int** low\_t = 70; // 下限越大，噪点越少，但也容易不稳定
6. **int** high\_t = 150;
7. Mat edges;
8. Canny(grayscale, edges, low\_t, high\_t);
10. Mat region = region\_selection(edges);
12. vector<Vec4i> hough = detect\_lines(region);
13. Mat binary\_image = Mat::zeros(image.size(), CV\_8UC1);
14. draw\_lines\_on\_binary\_image(hough, binary\_image);
15. **return** binary\_image;
16. }

在得到道路图像之后，就可以传入Tracking函数进行计算道路中点。Tracking函数的实现思路如下：

外层循环由图像底部向上扫描，内层循环由中间向两边移动，将扫描到的白点分别存入left\_line容器和right\_line容器中，对每一行得到的left和right进行计算得到一个mid，将mid存入mid\_line容器中。最后取mid数组中间的五个点的纵坐标进行计算求均值，并返回这一张道路的中点值。

Tracking函数如下：

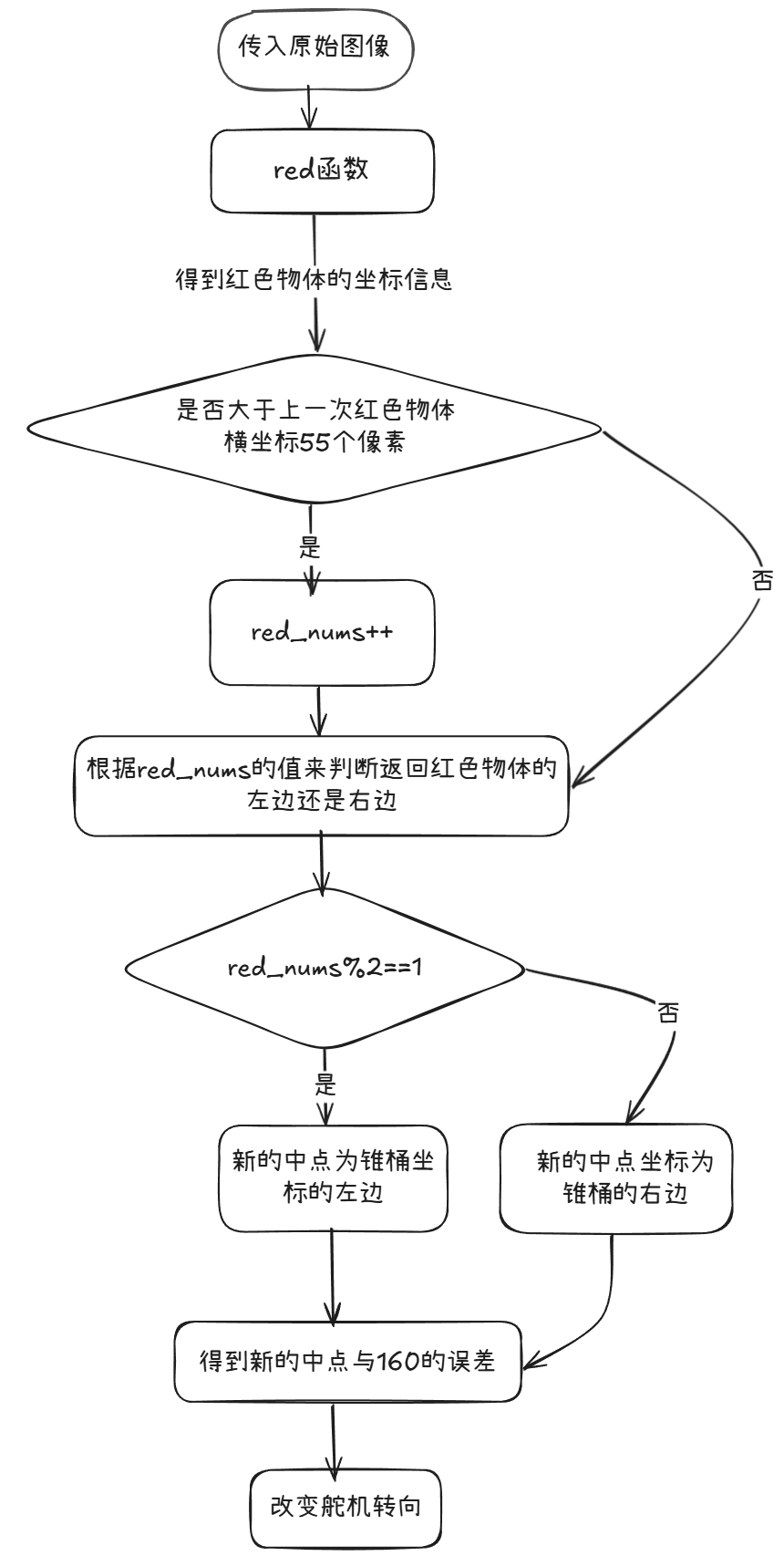
1. **int** Tracking(Mat dilated\_image, Mat frame) {
2. **int** begin = 160;
3. left\_line.clear();
4. right\_line.clear();
5. mid.clear();
6. **int** rows = dilated\_image.rows;
7. **int** cols = dilated\_image.cols;
8. **for** (**int** i = 239; i >= rows\*0.5; i--) {
9. **int** find\_l = 0;
10. **int** find\_r = 0;
11. **int** to\_left = begin;
12. **int** to\_right = begin;
13. **while** (to\_left != 1) {
14. **if** (dilated\_image.at<uchar>(i, to\_left) == 255 && dilated\_image.at<uchar>(i, to\_left + 1) == 255) {
15. find\_l = 1;
16. left\_line.push\_back(cv::Point(to\_left, i));
17. **break**;
18. } **else** {
19. to\_left--;
20. }
21. }
22. **if** (to\_left == 1) {
23. left\_line.push\_back(cv::Point(1, i));
24. }
25. **while** (to\_right != 318) {
26. **if** (dilated\_image.at<uchar>(i, to\_right) == 255 && dilated\_image.at<uchar>(i, to\_right - 2) == 255) {
27. find\_r = 1;
28. right\_line.push\_back(cv::Point(to\_right, i));
29. **break**;
30. } **else** {
31. to\_right++;
32. }
33. }
34. **if** (to\_right == 318) {
35. right\_line.push\_back(cv::Point(318, i));
36. }
37. cv::Point midx1 = left\_line.back();
38. cv::Point midx2 = right\_line.back();
39. mid.push\_back(cv::Point(**int**((midx1.x + midx2.x) / 2), i));
40. begin = (to\_right + to\_left) / 2;
41. **if** (to\_left == 1 && to\_right == 318) mid.pop\_back();
42. }
44. **if** (mid.size() == 0) {
45. **return** 160;//默认
46. }
47. **int** size = mid.size() / 2;
48. **if** (size + 5 > mid.size()) size = mid.size();
49. **else** size = size + 5;
50. mid\_final= 0;
51. **for** (**int** i = mid.size() / 2; i < size; i++) {
52. mid\_final += mid[i].x;
53. cout << "mid: " << mid[i] << "    ";
54. }
56. mid\_final /= (size - mid.size() / 2);
57. **return** mid\_final;
59. }

### 锥桶避障方案实现

首先使用red函数将图像指定区域的红色物体识别出来并返回第一个检测到的红色物体的区域，然后在find\_red\_cone函数里面对返回的红色物体区域进行判断，如果这次检测到的红色物体区域大于上一次检测到的红色物体区域达到55个像素，则认为出现了一个新的锥桶，并对锥桶的数量进行加1。再根据锥桶的个数选择返回红色物体的左边坐标还是右边坐标。

最后根据锥桶的个数来决定新的中点是在红色锥桶坐标的偏左几个像素还是偏右几个像素，通过新中点与160的误差来改变舵机的转向，实现锥桶的绕行。

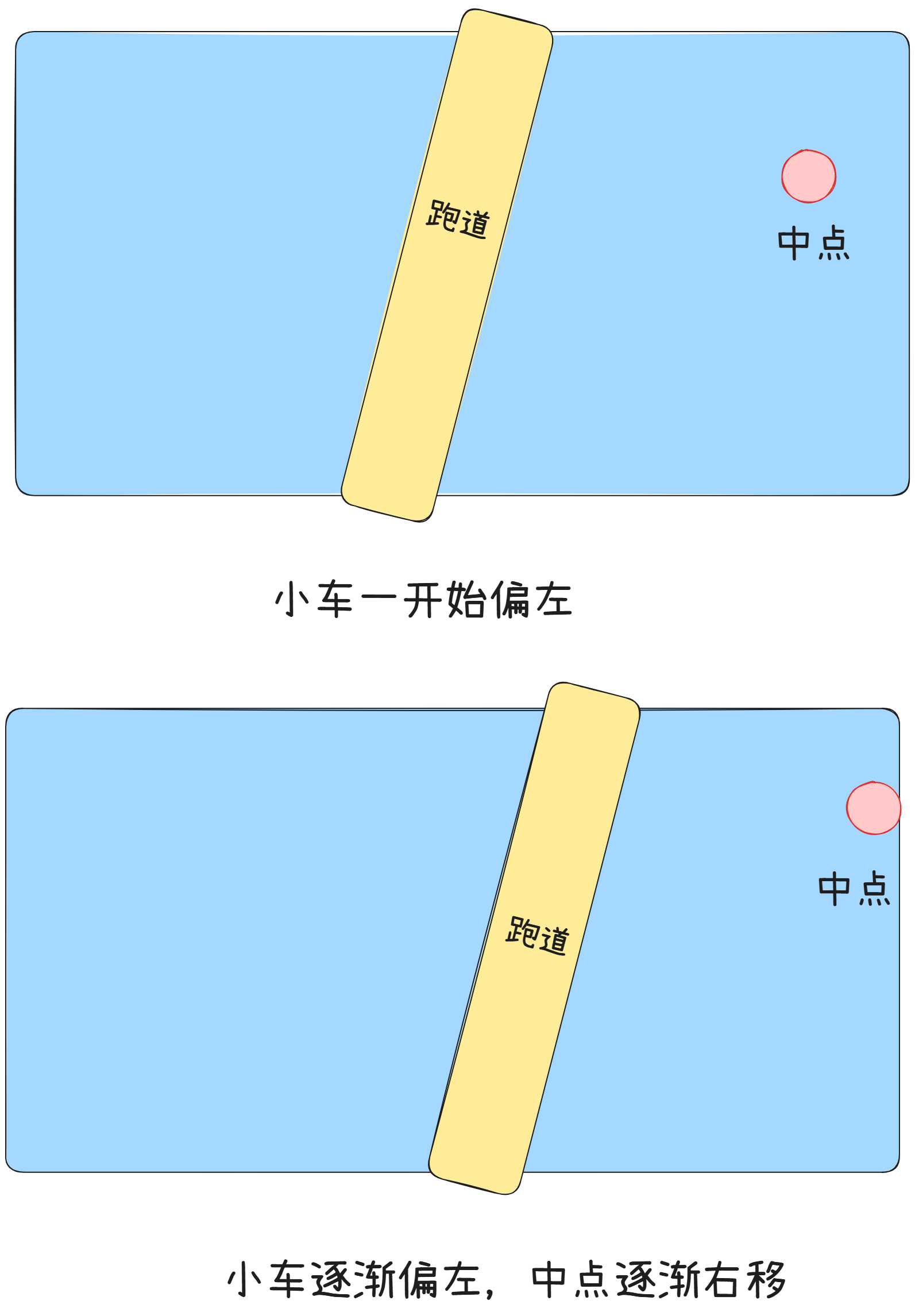
方案流程图如下：



### 变道方案实现

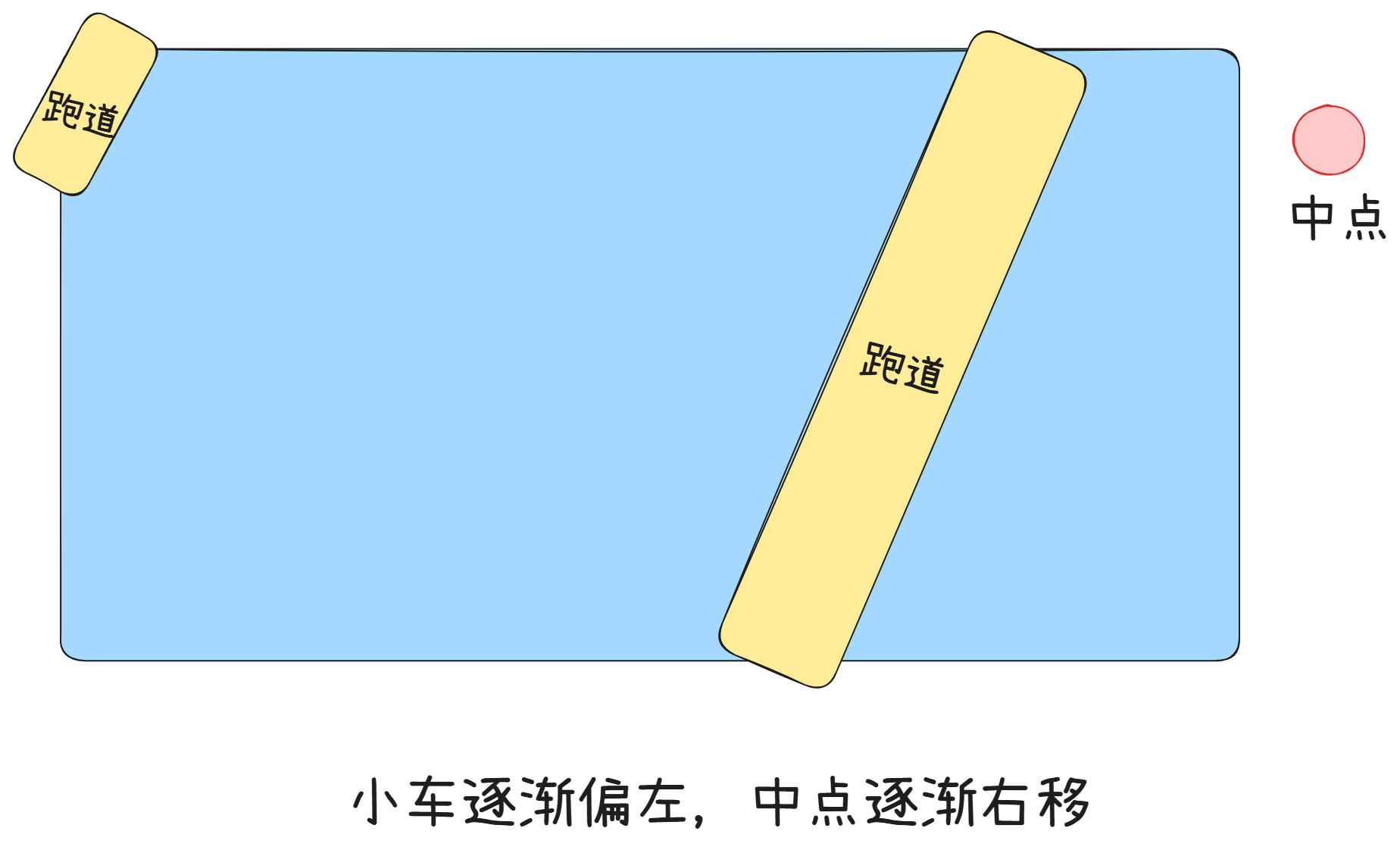
变道方案我们队伍创新性地提出了一种基于函数连续性的变道算法。其总体思路是先让小车打一个偏角之后再直行，最后恢复正常巡线，成功完成变道。

由于在同一车道时，道路中点的位置是关于时间的函数，具有连续性，且其一次导的数值在一定的范围内。在变道时，我们让小车根据需要将舵机打左或者打右。如下图：



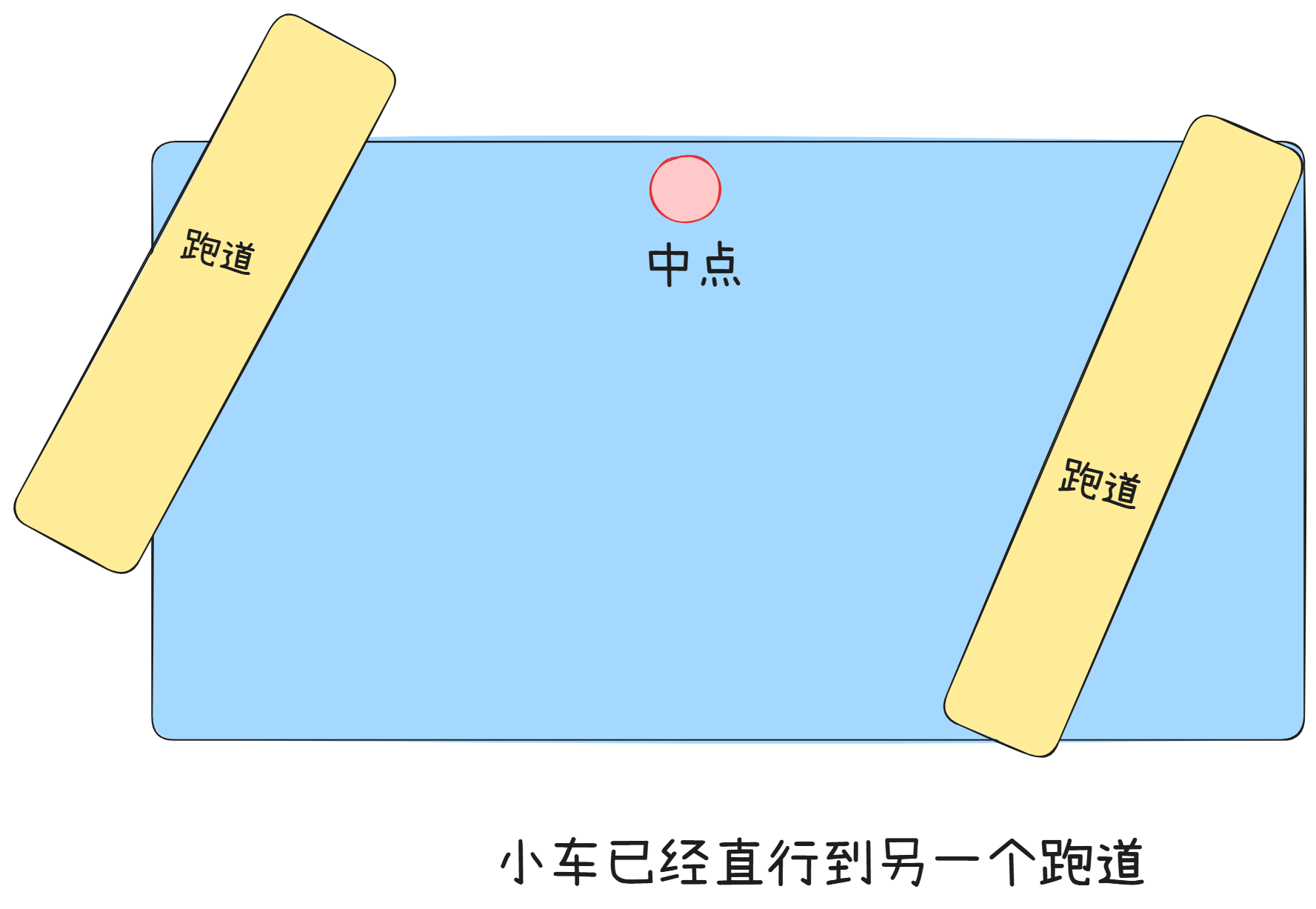
因为中点一定会向左或者右逐渐偏移，当某时刻检测出的道路中点位置超过了一定的阈值，我们可以认为小车已经朝向既定方向，此时让其回正方向。

在这个过程中，道路中点的位置会越来越大，或越来越小，如图：



可以看到，此时中点已经很大了，超出了图像的范围。

当小车到达另一车道时，会出现检测出的道路中点突然变小或突然变大的情况，根据函数的连续性判断此时小车已经将检测的道路切换到了目标车道，此时让其正常巡线，变道完成，如图：



### 入库方案实现

首先根据上一次A、B标志的指示，选择将小车前进的目标点从道路中点变为道路中点偏40像素的位置，然后进行对黄色进行颜色检测。首先我们截取了一个长320，宽48的长条形区域，在这个区域内对黄线进行颜色检测，由于入库前和入库后的黄线是大面积的长条形，入库中的黄线区域只有一小部分，因此我们当第一次检测到大面积的黄色时，将黄色数量置为1，而在第二次检测到大面积黄色的时候将黄色数量置为2，并让小车停车。

除此之外，为了保险起见，我们在第一次检测到黄色之后再过0.5s让小车实现自动停车。

黄色识别代码如下：

1. Mat yellow\_hsv(Mat img) {
2. Mat cropped\_image, image;
3. Scalar lowerb = Scalar(20, 20, 20);//室内
4. Scalar upperb = Scalar(34, 255, 255);
6. **int** yellow\_num = 0;
7. **int** rows = img.rows;
8. image = img.clone();
9. cropped\_image = image;
10. cvtColor(cropped\_image, cropped\_image, COLOR\_BGR2HSV);
11. morphologyEx(cropped\_image, cropped\_image, MORPH\_OPEN, getStructuringElement(MORPH\_RECT, Size(5, 5)));
12. inRange(cropped\_image, lowerb, upperb, cropped\_image);
13. Mat res = region\_selection1(cropped\_image);
15. **int** sum = 0;
16. **for** (**int** i = rows \* 0.9; i>= rows \* 0.7; i--) {
17. **for** (**int** j = 0; j < 320; j++) {
18. **if** (res.at<uchar>(i, j) == 255) {
19. sum++;
20. }
21. }
22. }
23. cout << sum << endl;
24. **if** (sum > 1000) {
25. **if** (num\_yellow) {
26. num\_yellow++;
27. } **else** {
28. flag\_yellow++;
29. }
30. } **else** **if**(sum<=800){
31. **if** (flag\_yellow) {
32. num\_yellow++;
33. flag\_yellow = 0;
34. Set\_dian(10000);
35. }
36. }
37. **return** res;
38. }

## 控制部分解决方案

我们采用的是位置式PID，如果小车相应过慢则增加Kp的值，如果小车震荡严重则增加Kd的值，经过不断地调试最终得到让小车稳定行驶的Kp和Kd。

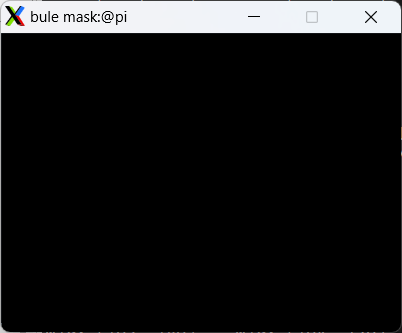
# 测试分析

1. 蓝色挡板的测试分析

下图是有蓝色挡板的图像：



下图是没有蓝色挡板的图像：



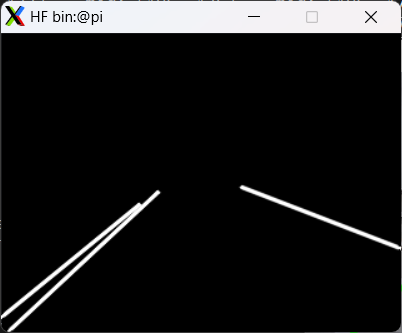
可以通过计算白色的面积来确认有无蓝色挡板

1. 道路检测的测试分析

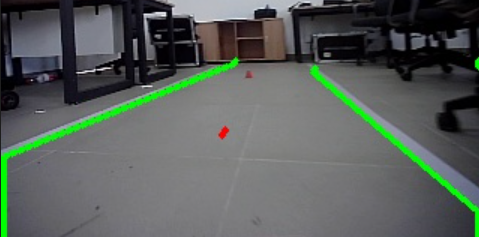
下图是经过Canny边缘检测后得到的道路图：



下图是经过霍夫变换后得到的道路图：



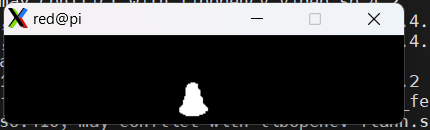
下图是经过调参之后的道路检测效果图片：绿色为检测出来的左右道路线，红色的点为由左右道路线计算而得的中点。



可以看出识别道路的效果十分地好

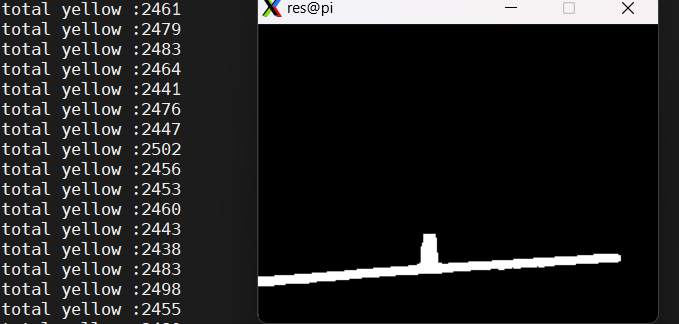
1. 锥桶避障的测试分析

可以看到对锥桶的检测效果也很好



1. 入库的测试分析

入库主要是通过检测黄色区域：



可以看到，检测黄色区域效果也很好，也成功输出了黄色的面积数量。

# 作品总结

在本次比赛中我们通过5G通信技术实现了对智能小车的远程驾驶，结合树莓派和opencv技术实现了智能小车的自主导航功能。通过结合官方提供的学习资料、网上查阅的资料以及团队之间的讨论合作，我们成功完成了这次比赛的所有功能模块，实现了小车的自主导航功能。

我们的主要代码都是由C++构成的，少部分目标检测采用Python脚本完成，因此程序运行的速度很快，获取图像的速度也很快。此外，我们的程序使用了模块化管理，将每个功能都单独抽取成一个.cpp文件，减少了代码的耦合性、增加了代码的可读性和功能的可扩展性。我们在主程序中使用标志位来分阶段开启识别任务，这不仅仅减少了大量到的if嵌套，而且加速了程序的运行速度以及增加了程序的可扩展性。在视觉处理方面我们也根据任务的不同采用不同的识别方法，比如识别斑马线的时候将其二值化为黑白图像，检测道路的时候将其进行边缘检测等。在变道任务中，我们队伍创新性地提出来一种基于函数连续性的变道算法，这个算法在实际的应用中也取得了很好的效果。

我们也发现了我们队伍一些不足的地方，比如道路检测受光线影响较大，如果未来可以根据图片自行调节光线就好了，在算法的调试和优化方面应该更加细心，否则可能会因为一个Bug耗费很多时间很精力。同时，也应该加强团队之间的交流和沟通，才能更高效地解决难题。

这次比赛是一次具有一定挑战难度的赛事，十分考验我们对于专业知识的理解和运用，通过这次比赛，学习到了更多关于视觉视觉，自主驾驶等的知识，也成功在智能小车上运用了我们的很多想法和思路，很有挑战性和趣味性，也学习到了很多解决问题的思路和方法，这些知识对我们未来帮助也很大。

总的来说，经过这一个多月的比赛，我们收获颇丰，全面提升了我们的专业能力和团队协作能力。

# 附录A

源程序

1. g5g.h:
2. #include <stdio.h>
3. #include <opencv4/opencv2/opencv.hpp>
4. #include <opencv4/opencv2/highgui/highgui.hpp>
5. #include <opencv4/opencv2/imgproc/imgproc.hpp>
6. #include<opencv4/opencv2/core/core.hpp>
7. #include<opencv4/opencv2/imgproc/types\_c.h>
8. #include <pthread.h>
9. #include <sys/stat.h>
10. #include <unistd.h>
11. #include <portaudio.h>
12. #include <AL/al.h>
13. #include <AL/alc.h>
14. #include <vector>
15. #include <fstream>
16. #include <cstdlib>
17. #include <sstream>
18. #include <cstring>
19. #include <chrono>
20. #include <iostream>
21. #include<stdlib.h>
22. #include <thread>
23. #include <condition\_variable>
24. #include <mutex>
25. #include "pigpio.h"
26. #include <cmath>
27. #include <csignal>

30. **using** **namespace** cv;
31. **using** **namespace** std;
32. **using** **namespace** std::chrono;

35. // add.h
36. #ifndef ADD\_H
37. #define ADD\_H
38. #define PI 3.1415926
40. **extern** cv::Mat frameToProcess;
41. **extern** **int** findab;
42. **extern** **int** blue\_card\_remove\_flag;
43. **extern** **int** find\_blue\_card;
44. **extern** **int** hasBanma;
45. **extern** **int** Banma\_finish;
46. **extern** **int** BanmaDesc;//检测到斑马线要减速的标志
48. **extern** **int** pass\_frame;
49. **extern** **int** red\_nums ;
50. **extern** **double** res\_y ;
52. **extern** **int** num\_yellow;
53. **extern** **int** flag\_yellow;
55. //在这个里面放上声明的函数：
56. /\*
57. 检测斑马线的函数
58. \*/
59. **int** crossroad(Mat frame,**int** isVisual);
61. /\*
62. 将图像转化为灰度图，消除噪声和进行边缘检测并返回图像
63. \*/
64. Mat preprocess\_frame(Mat frame);
66. /\*
67. 截取感兴趣区域(roi),并将截取的区域进行返回
68. \*/
69. Mat region\_of\_interest(Mat edges);
71. /\*
72. 检测跑道
73. \*/
74. Mat frame\_processorByEZ(Mat image);//二值化
75. Mat frame\_processorByHF(Mat image, **int** cnt);//霍夫
77. /\*
78. 检测图像中的直线，并且返回检测到的点的集合
79. \*/
80. vector<Vec4i> detect\_lines(Mat cropped\_edges);
82. /\*
83. 在frame 上，将lines上的点用绿色画出来
84. \*/
85. **void** draw\_lines(Mat frame, vector<Vec4i>& lines);
87. /\*
88. 语音模块
89. \*/
90. **struct** WavHeader {
91. **char** chunkID[4];
92. uint32\_t chunkSize;
93. **char** format[4];
94. **char** subchunk1ID[4];
95. uint32\_t subchunk1Size;
96. uint16\_t audioFormat;
97. uint16\_t numChannels;
98. uint32\_t sampleRate;
99. uint32\_t byteRate;
100. uint16\_t blockAlign;
101. uint16\_t bitsPerSample;
102. **char** subchunk2ID[4];
103. uint32\_t subchunk2Size;
104. };
106. /\*
107. 从指定的 WAV 文件中读取采样率
108. \*/
109. uint32\_t GetSampleRate(**const** **char**\* file);
111. /\*
112. 播放语言的函数
113. \*/
114. **bool** yuyin();
116. /\*
117. yolo 目标检测 a,b,左右转
118. \*/
119. **int** yoloAB();
120. **int** yoloLR();
122. //蓝色挡板
123. **void** blue\_card\_find(Mat frame);
124. **void** blue\_card\_remove(Mat frame);
125. Mat blue\_card\_process(Mat frame);
126. **int** detectBlueAreas(**const** cv::Mat inputImage);
128. // 红色锥桶
129. **int** find\_red\_cone(Mat img,**int** isVisual,**int** isCone);
130. Rect red(Mat img,**int** isCone);
132. //计算面积
133. **bool** Contour\_Area(vector<Point> contour1, vector<Point> contour2);
135. //黄色线：
136. Mat yellow\_hsv(Mat img);
137. /\*
138. 找黄色中线 返回黄线的坐标
139. \*/
140. **int** Tracking\_yellow(Mat iamge);
142. //驱动相关
143. **void** Set\_gpio();
144. **void** Set\_dian(**int** value);
145. **void** Set\_duo(**int** angle);
146. **double** PID(**double** error1);
148. #endif // ADD\_H
149. g5g.cpp:
150. #include "g5g.h"
152. cv::Mat frameToProcess;
153. **int** findab;
154. **int** find\_blue\_card;//找蓝色挡板的标志
155. **int** blue\_card\_remove\_flag;//蓝色挡板移除标志
156. **int** hasBanma;//有无斑马线标志
157. **int** Banma\_finish;//斑马线结束标志
158. **int** BanmaDesc;//斑马减速标志
160. **int** red\_nums ;
161. **double** res\_y ;
163. **int** num\_yellow;
164. **int** flag\_yellow ;
165. crossroad.cpp:
166. #include "g5g.h"
167. /\*
168. 检测斑马线   目前这个是最好的参数
170. \*/
171. **int** maskcount=0;
172. **int** crossroad(Mat frame,**int** isVisual) {
174. **int** rows = frame.rows;
175. **int** cols = frame.cols;
177. Mat src,mask1;
178. // 定义白色范围
179. //Scalar lower\_white(0, 0, 180);//操场
180. //Scalar upper\_white(180, 30, 255);//操场
181. Scalar lower\_white(0, 0, 170);//shinei
182. Scalar upper\_white(180, 50, 255);//shinei

185. Mat hsv;
186. cvtColor(frame, hsv, COLOR\_BGR2HSV);

189. inRange(hsv, lower\_white, upper\_white, mask1);
190. Mat kernel = getStructuringElement(MORPH\_RECT, Size(3, 3));
191. dilate(mask1, mask1, kernel);
192. erode(mask1, mask1, kernel);

195. // 计算起始行和结束行的索引
196. // int start\_row = static\_cast<int>(frame.rows \* 0.5);
197. // int end\_row = static\_cast<int>(frame.rows \* 0.9);
199. // // 创建矩形区域
200. // Rect roi(0, start\_row, frame.cols, end\_row - start\_row);
202. Point bottom\_left(0, rows );
203. Point top\_left(cols \* 0.4, rows \* 0.6);
204. Point top\_right(cols \* 0.6, rows \* 0.6);
205. Point bottom\_right(cols , rows );
206. Point mid\_left(0, rows \* 0.7);
207. Point mid\_right(cols , rows\*0.7);
208. vector<Point> vertices = {bottom\_left, mid\_left,top\_left,  top\_right,mid\_right,bottom\_right};
209. vector<vector<Point>> pts = {vertices};
211. // 创建掩码并填充多边形区域
212. Mat polyMask = Mat::zeros(frame.size(), CV\_8UC1);
213. fillPoly(polyMask, pts, Scalar(255));
215. // 提取多边形区域
216. mask1.copyTo(src, polyMask);
218. // src = mask1(roi);
220. **int** cout1 = 0;//用来记录每一行有多少组白色区域
221. **int** cout2 = 0;//用来记录每一组白色区域的宽度
222. **int** flag = 0;
224. **int** BanMaWidth = 5;//斑马线宽度的门限值
225. **int** BanMaNums = 4;//一行的斑马线组数门限值;
226. **for** (**int** i = rows\*0.62; i < rows\*0.9; i+=2) {
227. **if** (cout1 < BanMaNums) {
228. flag = 0;
229. }
230. cout1 = 0;
231. **for** (**int** j = 10; j < cols - 10;) {
232. cout2 = 0;
233. **if**(src.at<uchar>(i,j)==0){//这个像素点为黑色
234. j++;
235. **while**(j<src.cols - 10 && src.at<uchar>(i,j) == 255){
236. j++;
237. cout2++;
238. }
239. **if**(cout2 >= BanMaWidth&&cout2<40){//找到一个斑马块
240. cout1++;//斑马线组数++
241. }
242. }**else**{//如果是白色像素点，则后移即可
243. j++;
244. }
246. **if** (cout1 >= BanMaNums) {
247. flag++;
248. BanmaDesc=1;
249. **if**(isVisual){
250. cout<<"斑马线减速\n";
251. }
252. **if** (flag >= 3) {//连续三行都是
253. hasBanma = 1;
255. }
256. **break**;
257. }
258. }
259. }
260. **if**(isVisual){
262. imwrite("./image/mask/"+to\_string(maskcount++)+".jpg",src);
263. }
264. **return** hasBanma;
265. }
266. paodao.cpp:
267. #include "g5g.h"
268. /\*
269. \*   道路检测
270. \*/
272. Mat region\_selection( Mat image) {
273. Mat mask = Mat::zeros(image.size(), image.type());
274. **int** channel\_count = image.channels();
275. Scalar ignore\_mask\_color = (channel\_count == 3) ? Scalar(255, 255, 255) : Scalar(255);
276. **int** rows = image.rows;
277. **int** cols = image.cols;
278. Point bottom\_left(0, rows);
279. Point top\_left(cols \* 0.42, rows \* 0.5);
280. Point top\_right(cols \* 0.58, rows \* 0.5);
281. Point bottom\_right(cols , rows );
282. Point mid\_left(0, rows \* 0.65);
283. Point mid\_right(cols , rows\*0.65);
284. vector<Point> vertices = {bottom\_left, mid\_left,top\_left,  top\_right,mid\_right,bottom\_right};
285. vector<vector<Point>> pts = {vertices};
286. fillPoly(mask, pts, ignore\_mask\_color);
287. Mat masked\_image;
288. bitwise\_and(image, mask, masked\_image);
289. **return** masked\_image;
290. }
292. vector<Vec4i> detect\_lines(Mat image) {
293. **int** rho = 2;
294. **double** theta = CV\_PI / 180;
295. **int** threshold =30;
296. **int** minLineLength = 100;
297. **int** maxLineGap = 20;
298. vector<Vec4i> temp;
299. HoughLinesP(image, temp, rho, theta, threshold, minLineLength, maxLineGap);
300. vector<Vec4i>lines;
301. **for** (**const** auto& line : temp) {
302. **int** x1 = line[0];
303. **int** y1 = line[1];
304. **int** x2 = line[2];
305. **int** y2 = line[3];
306. **double** angle = atan2(line[3] - line[1], line[2] - line[0]) \* 180.0 / CV\_PI;
308. **if**(angle>20||angle<-20){//angle<70&&angle>20||angle<-20&&angle>-70
309. lines.push\_back(line);
310. }
311. }
312. **return** lines;
313. }
314. **void** draw\_lines(Mat frame, vector<Vec4i>& lines) {
315. **for** (**const** auto& line : lines) {
316. **int** x1 = line[0];
317. **int** y1 = line[1];
318. **int** x2 = line[2];
319. **int** y2 = line[3];
320. Scalar color(255);
321. **int** thickness = 2;
322. cv::line(frame, cv::Point(x1, y1), cv::Point(x2, y2), color, thickness);
323. }
324. }
325. **void** draw\_lines\_on\_binary\_image(**const** vector<Vec4i>& lines, Mat& binary\_image) {
326. **for** (**const** Vec4i& detected\_line : lines) {  // Change 'line' to 'detected\_line' to avoid conflict
327. Point pt1(detected\_line[0], detected\_line[1]);
328. Point pt2(detected\_line[2], detected\_line[3]);
329. line(binary\_image, pt1, pt2, Scalar(255), 3, LINE\_AA); // Draw the line
330. }
331. }
332. Mat frame\_processorByHF(Mat image, **int** cnt) {
333. Mat grayscale;
334. cvtColor(image, grayscale, COLOR\_BGR2GRAY);
335. //操场试试不用高斯滤波
336. **int** kernel\_size = 5; // 核越大噪点越少
337. GaussianBlur(grayscale, grayscale, Size(kernel\_size, kernel\_size), 0);//开一下试试 室内的 操场不一定需要
338. **int** low\_t = 70; // 下限越大，噪点越少，但也容易不稳定
339. **int** high\_t = 150;
340. /\*操场可以改\*/
341. // int low\_t = 30; // 下限越大，噪点越少，但也容易不稳定
342. // int high\_t = 100;
343. Mat edges;
344. Canny(grayscale, edges, low\_t, high\_t);
345. //imshow("edges", edges);
346. Mat region = region\_selection(edges);
347. imshow("HF region:",region);
348. //imwrite("./image/check/" + to\_string(cnt) + ".jpg" ,region);
350. vector<Vec4i> hough = detect\_lines(region);
351. draw\_lines(region,hough);
352. Mat binary\_image = Mat::zeros(image.size(), CV\_8UC1);
353. draw\_lines\_on\_binary\_image(hough, binary\_image);
354. //imwrite("./image/check/" + to\_string(cnt) + ".jpg" ,binary\_image);
355. imshow("HF bin:",binary\_image);
356. **return** binary\_image;
357. }
358. qudong.cpp:
359. #include"g5g.h"
361. //驱动相关参数
362. **double** kp = 0.25;
363. **double** ki = 0.00;
364. **double** kd = 0.125;
365. **double** last\_error = 0;
366. **double** sum\_error = 0;

369. **void** Set\_gpio() {
370. **if** (gpioInitialise() < 0) exit(1);
371. gpioSetMode(13, PI\_OUTPUT);
372. gpioSetPWMrange(13, 40000);
373. gpioSetPWMfrequency(13, 200);
375. **if** (gpioInitialise() < 0) exit(1);
376. gpioSetMode(12, PI\_OUTPUT);
377. gpioSetPWMrange(12,30000);
378. gpioSetPWMfrequency(12,50);
380. gpioPWM(13, 10000);
381. Set\_duo(156.5);
382. }
384. **int** last\_dian = 10000;
385. **void** Set\_dian(**int** value) {
386. **if** (value > 10800) {
387. **for** (**int** i = max(10800, last\_dian); i <= value; i += 50) {
388. gpioPWM(13, min(i, value));
389. usleep(25000);
390. }
391. } **else** gpioPWM(13, value);
392. last\_dian = value;
393. }
395. **double** PID(**double** error1) {
397. **double** angle = kp\*error1  + kd\*(error1-last\_error);
398. last\_error = error1;
399. **return** angle;
400. }
401. **void** Set\_duo(**int** angle) {
402. **double** value = (0.5 + (2 / 270.0) \* angle) / 20 \* 30000;
403. cout << "value: " << value << "\n";
404. gpioPWM(12, value);
405. }
406. bule\_card.cpp:
407. #include"g5g.h"
408. **bool** Contour\_Area(vector<Point> contour1, vector<Point> contour2)
409. {
410. **return** contourArea(contour1) > contourArea(contour2);
411. }
412. **int** blue\_img\_count=0;
413. Mat blue\_card\_process(Mat frame){
414. Mat frame\_a = frame;//深拷贝一帧
415. cvtColor(frame\_a, frame\_a, COLOR\_BGR2HSV);//表示要从 BGR 转换到 HSV 颜色空间,frame\_a 是转换后的图像
416. //HSV 颜色空间 这里对应蓝色
417. Mat mask;
418. Scalar scalarl = Scalar(100, 43, 46);//scalarl 定义了一个 HSV 颜色范围的下限，分别代表 H、S、V 的三个通道值
419. Scalar scalarH = Scalar(124, 255, 255);//scalarl 定义了一个 HSV 颜色范围的上限，分别代表 H、S、V 的三个通道值
421. inRange(frame\_a, scalarl, scalarH, mask);//用于根据给定的颜色范围生成二值掩膜  mask 是输出的二值图像，只有在指定的颜色范围内的像素会被设置为白色（255），其他像素为黑色（0）
422. Mat kernel = getStructuringElement(MORPH\_RECT, Size(5, 5));//生成一个结构元素，用于形态学操作。 ORPH\_RECT 指定了结构元素的形状为矩形 Size(5, 5) 指定了结构元素的大小为 5x5 像素。
423. morphologyEx(mask, mask, MORPH\_OPEN, kernel);//去除小的噪声 输入和输出都是 mask
424. morphologyEx(mask, mask, MORPH\_CLOSE, kernel);//用于填补小的空洞，进一步改善图像的连通性。
426. **for** (**int** y = 0; y < mask.rows; y++)
427. {
428. **for** (**int** x = 0; x < mask.cols; x++)
429. {
430. **if** (y < 70 || y > 120)//只保留行索引在 70 到 120 之间的像素，其余的像素都将被设置为 0
431. {
432. mask.at<uchar>(y, x) = 0;//将当前像素的值设置为 0（黑色），表示该区域在掩膜中被去除。
433. }
434. }
435. }
436. // imshow("bule mask:",mask);
437. imwrite("./image/blue/"+to\_string(blue\_img\_count++)+".jpg",mask);
438. **return** mask;
439. }
440. **void** blue\_card\_find(Mat frame)
441. {
443. Mat mask = blue\_card\_process(frame);
444. vector<vector<Point>> contours;
445. vector<Vec4i> hierarcy;
446. findContours(mask, contours, hierarcy, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE);
447. **if** (contours.size() > 0)
448. {
449. sort(contours.begin(), contours.end(), Contour\_Area);
450. vector<vector<Point>> newContours;
451. **for** (**const** vector<Point> &contour : contours)
452. {
453. Point2f center;
454. **float** radius;
455. minEnclosingCircle(contour, center, radius);
456. **if** (center.y > 70 && center.y < 120)
457. {
458. newContours.push\_back(contour);
459. }
460. }
462. contours = newContours;
464. **if** (contours.size() > 0)
465. {
466. **if** (contourArea(contours[0]) > 10000)
467. {
468. cout << "find biggest blue," << "contourArea(contours[0])="<<contourArea(contours[0])<<endl;
469. Point2f center;
470. **float** radius;
471. minEnclosingCircle(contours[0], center, radius);
472. //circle(frame, center, static\_cast<int>(radius), Scalar(0, 255, 0), 2);
473. find\_blue\_card = 1;//找到了蓝色挡板
474. }
475. **else**
476. {
477. cout << "not found blue" << endl;
478. }
479. }
480. }
481. **else**
482. {
483. cout << "not found blue" << endl;
484. }
485. }
487. **void** blue\_card\_remove(Mat frame)
488. {
489. Mat mask = blue\_card\_process(frame);
490. //cout << "entry move blue process" << endl;
491. vector<vector<Point>> contours;
492. vector<Vec4i> hierarcy;
493. findContours(mask, contours, hierarcy, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE);
494. **if** (contours.size() > 0)
495. {
496. sort(contours.begin(), contours.end(), Contour\_Area);
497. vector<vector<Point>> newContours;
498. cout<<"蓝色挡板移开面积："<<contourArea(contours[0])<<"\n";
499. **if** (contourArea(contours[0]) <=5000)
500. {
501. cout<<"蓝色挡板移开面积："<<contourArea(contours[0])<<"\n";
502. blue\_card\_remove\_flag = 1;
503. cout<<"蓝色挡板移开\n";
504. cout << "move" << endl;
505. }

508. }
509. **else**
510. {
511. blue\_card\_remove\_flag = 1;
512. cout << "蓝色挡板移开" << endl;
514. }
515. }
516. red\_cone.cpp:
517. #include "g5g.h"
518. Rect red(Mat img,**int** isCone)//识别到红色
519. {
520. Mat frame\_a = img.clone();//深拷贝一帧
521. Mat kernel\_3 = Mat::ones(Size(3, 3), CV\_8U);
522. Mat HSV;
523. // 计算起始行和结束行的索引
524. **int** start\_row,end\_row;
525. **if**(isCone){
526. start\_row = **static\_cast**<**int**>(frame\_a.rows \* 0.6);
527. end\_row = **static\_cast**<**int**>(frame\_a.rows \* 0.9);
529. }**else**{
530. start\_row = **static\_cast**<**int**>(frame\_a.rows \* 0.60);
531. end\_row = **static\_cast**<**int**>(frame\_a.rows \* 0.9);
533. }

536. // // 创建矩形区域
537. Rect region(0, start\_row, frame\_a.cols, end\_row - start\_row);
539. Mat roi = frame\_a(region);
540. cvtColor(roi, HSV, COLOR\_BGR2HSV);
542. // 定义红色的 HSV 范围
543. Scalar lower\_red1 = Scalar(0, 43, 46);
544. Scalar upper\_red1 = Scalar(10, 255, 255);
545. Scalar lower\_red2 = Scalar(153, 43, 46);
546. Scalar upper\_red2 = Scalar(180, 255, 255);
548. // 根据给定的颜色范围生成二值掩膜
549. Mat mask1, mask2, mask;
550. inRange(HSV, lower\_red1, upper\_red1, mask1);
551. inRange(HSV, lower\_red2, upper\_red2, mask2);
552. mask = mask1 | mask2; // 合并两个掩膜
554. Mat erosion;
555. erode(mask, erosion, kernel\_3, Point(-1, -1), 1, BORDER\_CONSTANT, morphologyDefaultBorderValue());
556. //erode(mask, erosion, kernel\_3, Point(-1, -1), 1, BORDER\_CONSTANT, morphologyDefaultBorderValue());
557. Mat dilation;
558. dilate(erosion, dilation, kernel\_3, Point(-1, -1), 1, BORDER\_CONSTANT, morphologyDefaultBorderValue());
559. //dilate(erosion, dilation, kernel\_3, Point(-1, -1), 1, BORDER\_CONSTANT, morphologyDefaultBorderValue());
560. Mat target;
561. bitwise\_and(roi, roi, target, dilation);
562. Mat binary;
563. threshold(dilation, binary, 127, 255, THRESH\_BINARY);
564. vector<vector<Point>> contours;
565. findContours(binary, contours, RETR\_EXTERNAL, CHAIN\_APPROX\_SIMPLE);
566. **if** (contours.size() == 0) {
567. **return** Rect();
568. }
569. **else** {
570. // 绘制轮廓
571. **return** boundingRect(contours[0]);
572. }
573. }
574. **int** find\_red\_cone(Mat img,**int** isVisual,**int** isCone)//识别到锥桶
575. {
576. Rect obstacle\_object = red(img,isCone);//识别到的红色物体
577. **if** (obstacle\_object.height > 5 && obstacle\_object.height < 100)//对识别到的红色物体限幅,
578. {
579. **if**(isVisual){
580. Mat image = img.clone();
581. rectangle(image, Point(obstacle\_object.x, obstacle\_object.y ), Point(obstacle\_object.x + obstacle\_object.width, obstacle\_object.y + obstacle\_object.height ), Scalar(255, 0, 0), 5);
582. imwrite("./image/mask/red.jpg",image);
584. //imshow("detect red",image);
585. }
587. **if** (isCone&&(obstacle\_object.y < res\_y - 55)) {
588. red\_nums++;
589. cout<<"检测到第"<<red\_nums<<"个锥桶\n";
590. }
591. **if**(isCone) res\_y = obstacle\_object.y;
592. **if**(red\_nums%2==0){
593. **return** obstacle\_object.x+ obstacle\_object.width;
594. }**else**{
595. **return** obstacle\_object.x;
596. }
597. }
598. **return** -1;
599. }
600. yellow\_edges.cpp
601. #include"g5g.h"
602. **int** num\_yellow = 0;
603. **int** flag\_yellow = 0;
605. std::vector<cv::Point> mid\_yellow;
606. **int** yellow\_image\_count = 0;
607. **int** yellowImageCount=0;
608. Mat region\_selection1( Mat image) {
609. Mat mask = Mat::zeros(image.size(), image.type());
610. **int** channel\_count = image.channels();
611. Scalar ignore\_mask\_color = (channel\_count == 3) ? Scalar(255, 255, 255) : Scalar(255);
612. **int** rows = image.rows;
613. **int** cols = image.cols;
614. Point bottom\_left(0, rows \* 0.9);
615. Point top\_left(0, rows \* 0.7);
616. Point top\_right(cols, rows \* 0.7);
617. Point bottom\_right(cols , rows \* 0.9);
618. vector<Point> vertices = {bottom\_left, top\_left, top\_right, bottom\_right};
619. vector<vector<Point>> pts = {vertices};
620. fillPoly(mask, pts, ignore\_mask\_color);
621. Mat masked\_image;
622. bitwise\_and(image, mask, masked\_image);
623. **return** masked\_image;
624. }
625. /\*
626. 检测图像中黄色区域
627. \*/
628. Mat yellow\_hsv(Mat img) {
629. Mat cropped\_image, image;
630. //黄色标准
631. //Scalar lowerb = Scalar(26, 43, 46);
632. //Scalar upperb = Scalar(34, 255, 255);
634. Scalar lowerb = Scalar(20, 20, 20);//室内
635. //Scalar lowerb = Scalar(0, 0, 0);//操场
636. Scalar upperb = Scalar(34, 255, 255);
638. **int** yellow\_num = 0;
639. **int** rows = img.rows;
640. image = img.clone();
641. cropped\_image = image;
642. cvtColor(cropped\_image, cropped\_image, COLOR\_BGR2HSV);
643. morphologyEx(cropped\_image, cropped\_image, MORPH\_OPEN, getStructuringElement(MORPH\_RECT, Size(5, 5)));
644. inRange(cropped\_image, lowerb, upperb, cropped\_image);
645. Mat res = region\_selection1(cropped\_image);
646. //imshow("yuan", img);
647. //imshow("title", res);
648. string filename = "./image/yellow/" + to\_string(yellow\_image\_count++) + "yellow\_" + ".jpg";
649. imwrite(filename, res);
650. **int** sum = 0;
651. **for** (**int** i = rows \* 0.9; i>= rows \* 0.7; i--) {
652. **for** (**int** j = 0; j < 320; j++) {
653. **if** (res.at<uchar>(i, j) == 255) {
654. sum++;
655. }
656. }
657. }
658. cout << sum << endl;
659. **if** (sum > 1000) {
660. **if** (num\_yellow) {
661. num\_yellow++;
662. } **else** {
663. flag\_yellow++;
664. }
665. } **else** **if**(sum<=800){
666. **if** (flag\_yellow) {
667. num\_yellow++;
668. flag\_yellow = 0;
669. Set\_dian(10000);
670. }
671. }
672. **return** res;
674. }
675. /\*
676. 找黄色中线 返回黄线的坐标
677. \*/
678. **int** Tracking\_yellow(Mat image) {
679. mid\_yellow.clear();
680. **int** begin = 160;
681. **int** rows = image.rows;
682. **int** cols = image.cols;
683. **for**(**int** i = rows\*0.7;i<=rows\*0.9;i++){
684. **for**(**int** j = 10;j<=310;j++){
685. **if**(image.at<uchar>(i, j) == 255&&image.at<uchar>(i, j+1) == 255&&image.at<uchar>(i, j+3) == 255){
686. //找到黄线了 列在前 行在后
687. mid\_yellow.push\_back(cv::Point(j, i));
688. **break**;
689. }
690. }
691. }
692. **if**(mid\_yellow.size()<=0){
693. **return** 240;
694. }
695. **int** size = mid\_yellow.size()/2;
696. **if**(size+5>mid\_yellow.size()){
697. size = mid\_yellow.size();
698. }
699. **int** mid\_line = 0;
700. **for**(**int** i = mid\_yellow.size()/2;i<size;i++){
701. mid\_line += mid\_yellow[i].x;
702. }
703. mid\_line/= (size - mid\_yellow.size() / 2);
705. **return** mid\_line;
707. }
708. yolo.cpp:
709. #include "g5g.h"
710. #include <cstdlib>
711. **int** yoloAB(){
713. //setenv("PATH", "/usr/local/bin:/usr/bin:/bin", 1);  // 设置正确的路径
714. system("/usr/local/bin/python3 /home/pi/g5g-new/yolo.py > res.txt");
715. // 打开temp.txt文件以读取输出
716. std::ifstream file("res.txt");
717. **if** (file.is\_open()) {
718. std::string line;
719. **while** (getline(file, line)) {
720. // 打印每一行输出
721. **return** std::stoi(line);
722. }
723. file.close();
724. } **else** {
725. std::cerr << "无法打开文件" << std::endl;
726. }
727. **return** -1;
728. }
729. **int** yoloLR(){
731. //setenv("PATH", "/usr/local/bin:/usr/bin:/bin", 1);  // 设置正确的路径
732. system("/usr/local/bin/python3 /home/pi/g5g-new/yoloLR.py > lr.txt");
733. // 打开temp.txt文件以读取输出
734. std::ifstream file("lr.txt");
735. **if** (file.is\_open()) {
736. std::string line;
737. **while** (getline(file, line)) {
738. // 打印每一行输出
739. **return** std::stoi(line);
740. }
741. file.close();
742. } **else** {
743. std::cerr << "无法打开文件" << std::endl;
744. }
745. **return** -1;
746. }
747. ///home/pi/.local/lib/python3.7/site-packages/onnxruntime/\_\_init\_\_.py
748. yuyin.cpp:
749. #include "g5g.h"
751. uint32\_t GetSampleRate(**const** **char**\* file)
752. {
753. ifstream ifs(file, ios::in|ios::binary);
754. **if**(!ifs.is\_open())
755. {
756. std::cerr << "Can't open file:" << file << endl;
757. **return** 0;
758. }
759. WavHeader header;
760. ifs.read(**reinterpret\_cast**<**char** \*>(&header), **sizeof**(WavHeader));
761. ifs.close();
762. **if** (std::string(header.chunkID, 4) != "RIFF" ||
763. std::string(header.format, 4) != "WAVE" ||
764. std::string(header.subchunk1ID, 4) != "fmt " ||
765. (std::string(header.subchunk2ID, 4) != "data" &&
766. std::string(header.subchunk2ID, 4) != "LIST")
767. ) {
768. std::cerr << "Invalid WAV file" << std::endl;
769. **return** 0;
770. }
771. **return** header.sampleRate;
773. }
775. **bool** yuyin()
776. {
777. **const** **char**\* file = "ans.wav";
778. // 初始化OpenAL
779. ALCdevice\* device = alcOpenDevice(nullptr);
780. ALCcontext\* context = alcCreateContext(device, nullptr);
781. alcMakeContextCurrent(context);
783. **const** unsigned **int** False = 0;
784. unsigned **int** nSampleRate = GetSampleRate(file);
785. **if**(False == nSampleRate)
786. **return** **false**;
788. std::ifstream ifs(file, ios::binary|ios::in);
789. std::ostringstream tmp;
790. tmp << ifs.rdbuf();
791. std::string str = tmp.str();
792. ifs.close();
794. ALuint source;
795. alGenSources(1, &source);
796. ALuint bufferId;
797. alGenBuffers(1, &bufferId);
798. alBufferData(bufferId, AL\_FORMAT\_MONO16, str.c\_str(), str.size(), nSampleRate);
799. alSourcei(source, AL\_BUFFER, bufferId);
800. alSourcePlay(source);
802. // 等待音频播放完毕
803. ALint state;
804. **do**
805. {
806. alGetSourcei(source, AL\_SOURCE\_STATE, &state);
807. } **while** (state == AL\_PLAYING);
809. // 清理
810. alDeleteSources(1, &source);
811. alDeleteBuffers(1, &bufferId);
812. alcDestroyContext(context);
813. alcCloseDevice(device);
814. **return** **true**;
815. }
816. line.cpp:
817. #include "g5g.h"
819. /\*
820. a b 标志不要离车库太近
821. \*/
823. // 打开摄像头（参数2表示第二个摄像头，可以改为视频文件路径）
824. //-1 0 1 2
825. // 初始化帧计数器
826. **int** frame\_count = 0;
828. **int** find\_blue\_card=0;//蓝色挡板标志->0：未找到蓝色挡板
829. **int** blue\_card\_remove\_flag=0;//蓝色挡板移开标志->0：未移开蓝色挡板
830. **int** hasBanma=0;//有无斑马线标志
831. **int** Banma\_finish=0;//斑马线结束标志
832. **int** BanmaDesc = 0;//斑马线减速标志

835. **double** error1 = 0;
836. **double** min\_angle = 0;
837. **double** max\_angle =200;
838. //红色锥桶
839. **int** red\_nums =0;
840. **double** res\_y = 240.0;
841. **int** pass\_nums =0;
843. **int** LandR = -1;
844. **int** changeDao\_finish =0;//左右变道结束标志->测试结果：不太稳定
845. **bool** changeroad = 0;//需要变道
846. **bool** turn\_left = 0;//需要左转向
847. **bool** turn\_right = 0;//需要右转
848. **int** check\_again = 0;
849. **int** red\_cone\_finish=0; //红色锥桶结束标志
850. **int** AB\_finish=0;//ab 结束标志
851. **int** AandB =-1;
852. **int** yellow\_finish=0;
854. **int** start\_clock = 0;
855. **clock\_t** yellow\_clock\_start;
856. **clock\_t** yellow\_clock\_end;
857. **int** LandRpass=0;


861. //左右中线
862. std::vector<cv::Point> left\_line;
863. std::vector<cv::Point> right\_line;
864. std::vector<cv::Point> mid;
865. **int** mid\_final = 160;//中线

868. **int** last\_mid;
870. // 全局变量，用于控制循环
871. **bool** running = **true**;
872. // 信号处理函数
873. **void** signalHandler(**int** signum) {
874. cout << "Interrupt signal (" << signum << ") received.\n";
876. running = **false**; // 设置运行标志为false，以退出循环
877. }
879. Mat applyGammaCorrection(**const** Mat& image, **double** gamma) {
880. CV\_Assert (gamma >= 0);
881. Mat lookupTable(1, 256, CV\_8U);
882. uchar\* p = lookupTable.ptr();
883. **for** (**int** i = 0; i < 256; ++i) {
884. p[i] = saturate\_cast<uchar> (pow(i / 255.0, gamma) \* 255.0);
885. }
886. Mat result;
887. LUT(image, lookupTable, result);
888. **return** result;
889. }
890. // Mat gammaCorrected = applyGammaCorrection(image, 0.6); // Gamma correction

893. string str = "sudo cp /home/pi/.Xauthority /root/";
894. **int** flag = system(str.c\_str());
896. **int** imageCount=0;
897. **bool** check\_fps = 0;
898. **int** Tracking(Mat dilated\_image, Mat frame) {
899. **int** begin = 160;
900. left\_line.clear();
901. right\_line.clear();
902. mid.clear();
903. **int** rows = dilated\_image.rows;
904. **int** cols = dilated\_image.cols;
905. **for** (**int** i = 239; i >= rows\*0.5; i--) {
906. **int** find\_l = 0;
907. **int** find\_r = 0;
908. **int** to\_left = begin;
909. **int** to\_right = begin;
910. **while** (to\_left != 1) {
911. **if** (dilated\_image.at<uchar>(i, to\_left) == 255 && dilated\_image.at<uchar>(i, to\_left + 1) == 255) {
912. find\_l = 1;
913. left\_line.push\_back(cv::Point(to\_left, i));
914. **break**;
915. } **else** {
916. to\_left--;
917. }
918. }
919. **if** (to\_left == 1) {
920. left\_line.push\_back(cv::Point(1, i));
921. }
922. **while** (to\_right != 318) {
923. **if** (dilated\_image.at<uchar>(i, to\_right) == 255 && dilated\_image.at<uchar>(i, to\_right - 2) == 255) {
924. find\_r = 1;
925. right\_line.push\_back(cv::Point(to\_right, i));
926. **break**;
927. } **else** {
928. to\_right++;
929. }
930. }
931. **if** (to\_right == 318) {
932. right\_line.push\_back(cv::Point(318, i));
933. }
934. cv::Point midx1 = left\_line.back();
935. cv::Point midx2 = right\_line.back();
936. mid.push\_back(cv::Point(**int**((midx1.x + midx2.x) / 2), i));
937. begin = (to\_right + to\_left) / 2;
938. **if** (to\_left == 1 && to\_right == 318) mid.pop\_back();
939. }
941. **if** (mid.size() == 0) {
942. imwrite("./image/mid/"+to\_string(imageCount++)+".jpg",frame);
943. **return** 160;//默认
944. }
945. **int** size = mid.size() / 2;
946. **if** (size + 5 > mid.size()) size = mid.size();
947. **else** size = size + 5;
948. mid\_final= 0;
949. **for** (**int** i = mid.size() / 2; i < size; i++) {
950. mid\_final += mid[i].x;
951. cout << "mid: " << mid[i] << "    ";
952. cv::circle(frame, mid[i], 2, cv::Scalar(0, 0, 255), -1);
953. }
954. **for** (**int** i = 0; i < left\_line.size(); i++) {
955. cv::circle(frame, left\_line[i], 2, cv::Scalar(0, 255, 0), -1);
956. }
957. **for** (**int** i = 0; i < right\_line.size(); i++) {
958. cv::circle(frame, right\_line[i], 2, cv::Scalar(0, 255, 0), -1);
959. }
960. imwrite("./image/mid/"+to\_string(imageCount++)+".jpg",frame);
961. mid\_final /= (size - mid.size() / 2);
962. **return** mid\_final;

965. }
966. **int** last\_left;
967. **int** last\_mid\_final=160;
968. **int** left\_mid(){
969. **int** size = left\_line.size()/2;
970. **if**(size+5>left\_line.size()){
971. size = left\_line.size();
972. }
973. **int** left\_total = 0;
974. **for**(**int** i = left\_line.size()/2;i<size;i++){
975. left\_total += left\_line[i].x;
976. }
977. **if**(abs(last\_left-left\_total/(size-left\_line.size()/2))>=50){
978. changeDao\_finish=1;//防止把左右变道识别为红锥桶
979. cout<<"变道结束\n";
980. }
981. **return** (last\_left = left\_total/(size-left\_line.size()/2));
982. }
983. **void** blue\_card(){
984. VideoCapture cap(-1);
985. **if** (!cap.isOpened()) {
986. printf("Error: Could not open video source.");
987. printf("无法捕获摄像头");
988. find\_blue\_card = blue\_card\_remove\_flag=1;
989. }
990. cap.set(CAP\_PROP\_FRAME\_WIDTH, 320);
991. cap.set(CAP\_PROP\_FRAME\_HEIGHT, 240);
992. Mat frame;
993. **while**(find\_blue\_card==0||blue\_card\_remove\_flag==0){
994. cap >> frame;
995. **if** (frame.empty()) {
996. cout<<"帧读取错误\n";
997. **break**;
998. }
999. **if**(find\_blue\_card==0){//一开始先找蓝色挡板
1000. cout<<"找蓝色挡板开启\n";
1001. blue\_card\_find(frame);//找挡板
1002. **if**(find\_blue\_card){
1003. **continue**;
1004. }
1006. }
1007. **if**(blue\_card\_remove\_flag==0&&find\_blue\_card){//找到了蓝色挡板，但是还未移除
1008. cout<<"找到了蓝色挡板，但是还未移除\n";
1009. blue\_card\_remove(frame);//检查是否移除
1010. **if**(blue\_card\_remove\_flag){//减缓帧堆积
1011. cout<<"移除\n";
1012. **break**;
1014. }
1015. }
1016. }
1017. cap.release();
1018. destroyAllWindows();
1020. }
1021. **void** solve() {
1022. VideoCapture cap(2);
1023. // 检查摄像头是否打开成功
1024. **if** (!cap.isOpened()) {
1025. printf("Error: Could not open video source.");
1026. exit(1);
1027. }
1028. cap.set(CAP\_PROP\_FRAME\_WIDTH, 320);
1029. cap.set(CAP\_PROP\_FRAME\_HEIGHT, 240);
1030. //第一步：初始化python环境
1032. **int** red\_position=-1;  //红色锥桶坐标
1034. **double** angle;
1036. **double** fps = 0.0;
1037. **int** frameCount = 0;
1038. **int** startTime = **static\_cast**<**int**>(getTickCount());
1039. **int** st =-1;
1041. high\_resolution\_clock::time\_point time1,time2;
1042. **clock\_t** start = clock();
1044. Mat frame,frame\_with\_line ;
1045. **while** (running&&yellow\_finish==0) {
1046. cout<<"fps:"<<fps<<"\n";
1048. cap >> frame;
1049. **if** (frame.empty()) {
1050. cout<<"帧读取错误\n";
1051. **break**;
1052. }
1053. //frame = applyGammaCorrection(frame, 0.9);
1054. **clock\_t** end = clock();
1056. // // 计算CPU时间并转换为秒
1057. **double** cpu\_time\_used = **static\_cast**<**double**>(end - start) / CLOCKS\_PER\_SEC;
1058. **if** (cpu\_time\_used < 0.1){
1059. cout<<"正在加载\n";
1060. **continue**;
1061. }
1062. **if**(blue\_card\_remove\_flag&&hasBanma == 0){//蓝色挡板移除，可以开始运动
1063. **if** (check\_fps) {
1064. Set\_dian(10900);
1065. }
1066. cout<<"蓝色挡板移除\n";
1067. cout<<"正在找斑马线\n";
1068. hasBanma = crossroad(frame, **true**);
1069. }
1071. **if**(hasBanma&&Banma\_finish==0){//找到斑马线，控制小车停下，播语音
1072. Banma\_finish = 1;//标记为结束
1073. cout<<"找到斑马线\n";
1074. Set\_duo(156.6);
1075. Set\_dian(10000);
1076. std::this\_thread::sleep\_for(std::chrono::seconds(2));
1077. cout<<"播报语音\n";
1078. yuyin();
1079. std::this\_thread::sleep\_for(std::chrono::seconds(2));
1080. cout<<"斑马线部分结束\n";
1081. Set\_dian(10800);
1083. }
1085. **if**(Banma\_finish&&LandR==-1&&changeDao\_finish==0){
1086. cout<<"斑马线结束\n";
1087. Set\_dian(10800);
1088. **if**(find\_red\_cone(frame,**false**,**false**)!=-1){
1089. cout<<"检测到红色 yolo 开始 检测左右变道\n";
1090. //usleep(250000);
1091. Set\_dian(10000);
1092. LandR = yoloLR();
1093. cout<<"左右结果为："<<LandR<<"\n";
1094. **if**(LandR!=-1){
1095. changeroad = 1;//需要变道
1096. **if**(LandR==3){
1097. turn\_left = 1;//需要左转向
1098. }
1099. **else**{
1100. turn\_right =1;//右转
1101. }
1102. cout<<"LR 结束 开始变道\n";
1104. }
1105. }

1108. }
1109. **if**(changeDao\_finish==0&&LandR!=-1){
1110. **if** (check\_fps) Set\_dian(10800);
1112. }
1113. **if**(changeDao\_finish&&red\_cone\_finish==0){
1114. cout<<"开始红色锥桶识别\n";
1115. **if**(check\_fps)Set\_dian(10900);
1116. red\_position = find\_red\_cone(frame,**false**,**true**);
1117. **if**(red\_nums>=3){
1118. pass\_nums++;
1119. **if**(pass\_nums>80){
1120. red\_cone\_finish =1;
1121. cout<<"红色锥桶识别结束\n";
1122. }
1123. }
1124. **if**(red\_position!=-1){
1125. cout<<"找到红锥桶" << red\_nums << "\n";
1126. **if**(red\_nums%2==1){
1127. error1 = 160 - (red\_position-53);
1128. cv::circle(frame, Point(red\_position-53,120), 2, cv::Scalar(0, 0, 255), -1);
1129. }**else**{
1130. error1 = 160 - (red\_position+53);
1131. cv::circle(frame, Point(red\_position+53,120), 2, cv::Scalar(0, 0, 255), -1);
1132. }
1134. angle = PID(error1);
1135. angle = 156.5 + angle;
1136. cout << "after angle: " << angle << endl;
1137. **if** (angle > max\_angle) angle = max\_angle;
1138. **if** (angle < min\_angle) angle = min\_angle;
1139. Set\_duo(angle);
1140. frameCount++;
1141. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1142. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1143. **if** (timeDiff >= 1.0) {
1144. fps = frameCount / timeDiff;
1145. frameCount = 0;
1146. startTime = currentTime;
1147. **if** (fps >= 10) check\_fps = **true**;
1148. }
1149. **continue**;
1150. }
1152. }
1153. **if**(red\_cone\_finish&&AB\_finish==0){//避障结束而AB 识别未结束
1154. cout<<"避障结束，开始ab识别\n";
1155. **if**(check\_fps)Set\_dian(10800);
1156. **if**(find\_red\_cone(frame,**true**,**false**)!=-1){
1157. Set\_dian(10000);
1158. usleep(250000);
1159. cout<<"有A B 红色出现，开启yolo\n";
1160. AandB = yoloAB();
1161. cout<<"AB识别结果为："<<AandB<<"\n";
1163. **if**(AandB!=-1){
1164. AB\_finish = 1;
1165. cout<<"AB 识别结束\n";
1166. }
1167. }

1170. }
1171. **if**(AB\_finish){
1172. cout<<"AB 结束，开始找黄线\n";
1173. Set\_dian(10800);
1174. Mat res = yellow\_hsv(frame);
1175. **if**(num\_yellow>=1){
1176. **if**(start\_clock==0){
1177. cout<<"黄色开始计时\n";
1178. yellow\_clock\_start = clock();
1179. start\_clock = 1;
1180. }**else**{
1181. yellow\_clock\_end = clock();
1182. **double** time\_used = **static\_cast**<**double**>(yellow\_clock\_end - yellow\_clock\_start) / CLOCKS\_PER\_SEC;
1183. cout<<"所用时间："<<time\_used<<"\n";
1184. **if**(time\_used >=0.5){
1185. cout<<"入库时间到，停止入库\n";
1186. yellow\_finish = 1;
1187. gpioPWM(13, 10000);
1188. }
1189. }
1190. }
1191. **if** (num\_yellow == 2) {
1192. yellow\_finish = 1;
1193. cout << "停车!" << endl;
1194. gpioPWM(13, 10000);
1195. **break**;
1196. }
1198. }

1201. Mat edges = frame\_processorByHF(frame, frameCount);
1202. mid\_final = Tracking(edges, frame);
1204. cout<<"中线为：:"<<mid\_final<<"\n";
1205. **if** (changeroad) {
1207. **if** (turn\_left) {//变向阶段, 目标是使中点值变成 170
1208. cout<<"正在左转\n";
1209. **if** (mid\_final > 230) {//170是为了让方向可以到达旁边的跑道了，不要再拐弯了，直接直走。180是为了防止变道前方向太偏左的问题
1210. cout<<"打直线\n";
1211. turn\_left = 0; // 不用再打角度了
1212. Set\_duo(156.5);
1213. last\_mid = mid\_final;
1214. frameCount++;
1215. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1216. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1217. **if** (timeDiff >= 1.0) {
1218. fps = frameCount / timeDiff;
1219. frameCount = 0;
1220. startTime = currentTime;
1221. **if** (fps >= 10) check\_fps = **true**;
1222. }
1223. **continue**;
1224. } **else** {//位置不对就让他变
1225. cout<<"打左\n";
1226. Set\_duo(170);
1227. frameCount++;
1228. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1229. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1230. **if** (timeDiff >= 1.0) {
1231. fps = frameCount / timeDiff;
1232. frameCount = 0;
1233. startTime = currentTime;
1234. **if** (fps >= 10) check\_fps = **true**;
1235. }
1236. **continue**;
1237. }
1238. } **if** (turn\_right) {//变向阶段, 目标是使中点值变成 170
1239. cout<<"正在右转\n";
1240. **if** (mid\_final < 110) {//170是为了让方向可以到达旁边的跑道了，不要再拐弯了，直接直走。180是为了防止变道前方向太偏左的问题
1241. cout<<"打直线\n";
1242. turn\_right = 0; // 不用再打角度了
1243. Set\_duo(156.5);
1244. last\_mid = mid\_final;
1245. frameCount++;
1246. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1247. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1248. **if** (timeDiff >= 1.0) {
1249. fps = frameCount / timeDiff;
1250. frameCount = 0;
1251. startTime = currentTime;
1252. **if** (fps >= 10) check\_fps = **true**;
1253. }
1254. **continue**;
1255. } **else** {//位置不对就让他变
1256. cout<<"打右\n";
1257. Set\_duo(140);
1258. frameCount++;
1259. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1260. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1261. **if** (timeDiff >= 1.0) {
1262. fps = frameCount / timeDiff;
1263. frameCount = 0;
1264. startTime = currentTime;
1265. **if** (fps >= 10) check\_fps = **true**;
1266. }
1267. **continue**;
1268. }
1269. } **else** { // 直行阶段
1270. **if** (last\_mid - mid\_final > 30 && mid\_final != 160) { //如果还在原来的跑道的话，last\_mid 是会<= mid\_final的
1271. check\_again++;
1272. **if** (check\_again >= 2) {
1273. cout << "开始正常巡道\n";
1274. changeroad = 0;// 说明现在已经识别到了左边的跑道，现在让其正常寻道
1275. changeDao\_finish =1;
1276. }
1277. last\_mid = max(last\_mid, mid\_final);
1278. **continue**;
1279. }
1280. last\_mid = max(last\_mid, mid\_final);
1281. **continue**;
1282. }
1283. }
1284. **else** **if**(AB\_finish){
1285. cout<<"正在转向入库\n";//->不会
1286. // error1 = 160 - (mid\_final + 40);//操场
1287. **if**(AandB ==2){
1288. error1 = 160 - (mid\_final + 40);//室内
1289. }**else** **if**(AandB ==1){
1290. error1 = 160 - (mid\_final - 40);
1291. }

1294. }**else**{
1295. error1 = 160 - mid\_final;
1296. }
1298. angle = PID(error1);
1299. angle = 156.5 + angle;
1300. cout << "after angle: " << angle << endl;
1301. **if** (angle > max\_angle) angle = max\_angle;
1302. **if** (angle < min\_angle) angle = min\_angle;
1303. Set\_duo(angle);
1304. //计算并打印帧率
1305. frameCount++;
1306. **int** currentTime = **static\_cast**<**int**>(getTickCount());
1307. **double** timeDiff = (currentTime - startTime) / getTickFrequency();
1309. **if** (timeDiff >= 1.0) {
1310. fps = frameCount / timeDiff;
1311. frameCount = 0;
1312. startTime = currentTime;
1313. **if** (fps >= 10) check\_fps = **true**;
1314. }
1316. //putText(frame, "FPS: " + std::to\_string(fps), Point(10, 30), FONT\_HERSHEY\_SIMPLEX, 1, Scalar(255, 255, 255), 2);
1318. **if**(waitKey(1)>=0){
1319. **break**;
1320. }
1321. }
1323. // 释放资源
1325. cap.release();
1326. destroyAllWindows();

1329. }
1331. **int** main() {
1332. //setenv("PATH", "/usr/local/bin:/usr/bin:/bin", 1);  // 设置正确的路径
1333. //初始化
1334. Set\_gpio();

1337. // cap.set(CAP\_PROP\_FPS, 30);
1339. // 注册信号 SIGINT 和信号处理程序
1340. signal(SIGINT, signalHandler);
1341. blue\_card();
1342. cout<<"找蓝色挡板结束\n";
1343. // pthread\_t tid;
1344. // pthread\_create(&tid, NULL, solve, &frame\_count);
1345. // pthread\_join(tid, NULL);
1346. solve();


1350. **return** 0;
1351. }
1352. yolo.py:
1353. # -\*- coding: utf-8 -\*-
1354. **import** sys
1355. **import** os
1356. **import** cv2
1357. **import** time
1358. **import** numpy as np
1360. **try**:
1361. **import** onnxruntime as ort
1362. **except** ModuleNotFoundError as e:
1363. **print**(f"Error importing onnxruntime: {e}")
1364. **raise**
1366. **class** yolov5\_lite():
1367. **def** \_\_init\_\_(self, model\_pb\_path, label\_path, confThreshold=0.5, nmsThreshold=0.5):
1368. so = ort.SessionOptions()
1369. so.log\_severity\_level = 3
1370. self.net = ort.InferenceSession(model\_pb\_path, so)
1371. self.classes = list(map(**lambda** x: x.strip(), open(label\_path, 'r').readlines()))
1373. self.confThreshold = confThreshold
1374. self.nmsThreshold = nmsThreshold
1375. self.input\_shape = (self.net.get\_inputs()[0].shape[2], self.net.get\_inputs()[0].shape[3])
1377. **def** letterBox(self, srcimg, keep\_ratio=True):
1378. top, left, newh, neww = 0, 0, self.input\_shape[0], self.input\_shape[1]
1379. **if** keep\_ratio **and** srcimg.shape[0] != srcimg.shape[1]:
1380. hw\_scale = srcimg.shape[0] / srcimg.shape[1]
1381. **if** hw\_scale > 1:
1382. newh, neww = self.input\_shape[0], int(self.input\_shape[1] / hw\_scale)
1383. img = cv2.resize(srcimg, (neww, newh), interpolation=cv2.INTER\_AREA)
1384. left = int((self.input\_shape[1] - neww) \* 0.5)
1385. img = cv2.copyMakeBorder(img, 0, 0, left, self.input\_shape[1] - neww - left, cv2.BORDER\_CONSTANT,
1386. value=0)  # add border
1387. **else**:
1388. newh, neww = int(self.input\_shape[0] \* hw\_scale), self.input\_shape[1]
1389. img = cv2.resize(srcimg, (neww, newh), interpolation=cv2.INTER\_AREA)
1390. top = int((self.input\_shape[0] - newh) \* 0.5)
1391. img = cv2.copyMakeBorder(img, top, self.input\_shape[0] - newh - top, 0, 0, cv2.BORDER\_CONSTANT, value=0)
1392. **else**:
1393. img = cv2.resize(srcimg, self.input\_shape, interpolation=cv2.INTER\_AREA)
1394. **return** img, newh, neww, top, left
1396. **def** postprocess(self, frame, outs, pad\_hw):
1397. newh, neww, padh, padw = pad\_hw
1398. frameHeight = frame.shape[0]
1399. frameWidth = frame.shape[1]
1400. ratioh, ratiow = frameHeight / newh, frameWidth / neww
1401. classIds = []
1402. confidences = []
1403. boxes = []
1404. **for** detection **in** outs:
1405. scores, classId = detection[4], detection[5]
1406. **if** scores > self.confThreshold:  # and detection[4] > self.objThreshold:
1407. x1 = int((detection[0] - padw) \* ratiow)
1408. y1 = int((detection[1] - padh) \* ratioh)
1409. x2 = int((detection[2] - padw) \* ratiow)
1410. y2 = int((detection[3] - padh) \* ratioh)
1411. classIds.append(classId)
1412. confidences.append(scores)
1413. boxes.append([x1, y1, x2, y2])
1415. # # Perform non maximum suppression to eliminate redundant overlapping boxes with
1416. # # lower confidences.
1418. #indices = cv2.dnn.NMSBoxes(boxes, confidences, self.confThreshold, self.nmsThreshold)
1419. indices = self.NMSBoxes(boxes, confidences, self.nmsThreshold)
1421. **for** ind **in** indices :
1422. frame = self.drawPred(frame, classIds[ind], confidences[ind], boxes[ind][0], boxes[ind][1], boxes[ind][2], boxes[ind][3])
1423. **return** frame
1425. **def** drawPred(self, frame, classId, conf, x1, y1, x2, y2):
1426. # Draw a bounding box.
1427. cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), thickness=2)
1429. label = '%.2f' % conf
1430. text = '%s:%s' % (self.classes[int(classId)], label)
1431. #print("classId:",classId)
1433. # Display the label at the top of the bounding box
1434. labelSize, baseLine = cv2.getTextSize(text, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 1)
1435. y1 = max(y1, labelSize[1])
1436. cv2.putText(frame, text, (x1, y1 - 10), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (0, 255, 0), thickness=1)
1437. **return** frame
1439. **def** detect(self, srcimg):
1440. img, newh, neww, top, left = self.letterBox(srcimg)
1441. img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)
1442. img = img.astype(np.float32) / 255.0
1443. blob = np.expand\_dims(np.transpose(img, (2, 0, 1)), axis=0)
1445. t1 = time.time()
1446. outs = self.net.run(None, {self.net.get\_inputs()[0].name: blob})[0]
1447. cost\_time = time.time() - t1
1448. #print("type:",type(outs))
1450. **if** outs **is** **not** None **and** len(outs) > 0:
1451. **print**(outs[0][5])
1452. **else**:
1453. **print**(-1)
1455. #srcimg = self.postprocess(srcimg, outs, (newh, neww, top, left))
1456. #infer\_time = 'Inference Time: ' + str(int(cost\_time \* 1000)) + 'ms'
1457. #cv2.putText(srcimg, infer\_time, (5, 20), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (0, 0, 0), thickness=1)
1458. **return** outs#srcimg
1460. **def** NMSBoxes(self,boxes, scores, threshold):
1461. **if** len(boxes) == 0:
1462. **return** []
1463. boxes = np.array(boxes)
1465. # ����߽�����
1466. areas = (boxes[:, 2] - boxes[:, 0]) \* (boxes[:, 3] - boxes[:, 1])
1468. scores = np.array(scores)
1470. # �����Ŷȵ÷ֽ�������߽��
1471. order = scores.argsort()[::-1]
1473. keep = []
1474. **while** order.size > 0:
1475. i = order[0]
1476. keep.append(i)
1478. # ���㽻����������
1479. xx1 = np.maximum(boxes[i, 0], boxes[order[1:], 0])
1480. yy1 = np.maximum(boxes[i, 1], boxes[order[1:], 1])
1481. xx2 = np.minimum(boxes[i, 2], boxes[order[1:], 2])
1482. yy2 = np.minimum(boxes[i, 3], boxes[order[1:], 3])
1484. # ���㽻������Ŀ��Ⱥ͸߶�
1485. w = np.maximum(0.0, xx2 - xx1)
1486. h = np.maximum(0.0, yy2 - yy1)
1488. # ���㽻����������
1489. inter = w \* h
1491. # ���㽻���ȣ�IoU��
1492. iou = inter / (areas[i] + areas[order[1:]] - inter)
1494. # ���� IoU С����ֵ�ı߽��
1495. inds = np.where(iou <= threshold)[0]
1496. order = order[inds + 1]
1498. **return** keep

1501. **def** yolo\_main():
1502. # ����Ĭ�ϲ���
1503. modelpath = 'BestAB.onnx'
1504. classfile = 'AB.names'
1505. confThreshold = 0.4
1506. nmsThreshold = 0.6
1508. '''''
1509. parser = argparse.ArgumentParser()
1510. parser.add\_argument('--modelpath', type=str, default='best.onnx', help="onnx filepath")
1511. parser.add\_argument('--classfile', type=str, default='coco.names', help="classname filepath")
1512. parser.add\_argument('--confThreshold', default=0.2, type=float, help='class confidence')
1513. parser.add\_argument('--nmsThreshold', default=0.6, type=float, help='nms iou thresh')
1514. args = parser.parse\_args()
1515. '''
1517. net = yolov5\_lite(modelpath, classfile, confThreshold, nmsThreshold=nmsThreshold)
1519. # ������ͷ
1521. cap = cv2.VideoCapture(-1)
1522. **if** **not** cap.isOpened():
1523. **print**(-1)
1524. **return**
1526. cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)
1527. cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)



1532. # ������ͷ��ȡһ֡ͼ��
1533. ret, frame = cap.read()
1534. **if** frame **is** None:
1535. **print**("�޷���ȡͼ��")
1536. **return** None
1538. outs = net.detect(frame.copy())

1541. #return outs
1542. yolo\_main()

1545. yoloLR.py:
1546. # -\*- coding: utf-8 -\*-
1547. **import** sys
1548. **import** os
1549. **import** cv2
1550. **import** time
1551. **import** numpy as np
1552. **try**:
1553. **import** onnxruntime as ort
1554. **except** ModuleNotFoundError as e:
1555. **print**(f"Error importing onnxruntime: {e}")
1556. **raise**
1558. **class** yolov5\_lite():
1559. **def** \_\_init\_\_(self, model\_pb\_path, label\_path, confThreshold=0.5, nmsThreshold=0.5):
1560. so = ort.SessionOptions()
1561. so.log\_severity\_level = 3
1562. self.net = ort.InferenceSession(model\_pb\_path, so)
1563. self.classes = list(map(**lambda** x: x.strip(), open(label\_path, 'r').readlines()))
1565. self.confThreshold = confThreshold
1566. self.nmsThreshold = nmsThreshold
1567. self.input\_shape = (self.net.get\_inputs()[0].shape[2], self.net.get\_inputs()[0].shape[3])
1569. **def** letterBox(self, srcimg, keep\_ratio=True):
1570. top, left, newh, neww = 0, 0, self.input\_shape[0], self.input\_shape[1]
1571. **if** keep\_ratio **and** srcimg.shape[0] != srcimg.shape[1]:
1572. hw\_scale = srcimg.shape[0] / srcimg.shape[1]
1573. **if** hw\_scale > 1:
1574. newh, neww = self.input\_shape[0], int(self.input\_shape[1] / hw\_scale)
1575. img = cv2.resize(srcimg, (neww, newh), interpolation=cv2.INTER\_AREA)
1576. left = int((self.input\_shape[1] - neww) \* 0.5)
1577. img = cv2.copyMakeBorder(img, 0, 0, left, self.input\_shape[1] - neww - left, cv2.BORDER\_CONSTANT,
1578. value=0)  # add border
1579. **else**:
1580. newh, neww = int(self.input\_shape[0] \* hw\_scale), self.input\_shape[1]
1581. img = cv2.resize(srcimg, (neww, newh), interpolation=cv2.INTER\_AREA)
1582. top = int((self.input\_shape[0] - newh) \* 0.5)
1583. img = cv2.copyMakeBorder(img, top, self.input\_shape[0] - newh - top, 0, 0, cv2.BORDER\_CONSTANT, value=0)
1584. **else**:
1585. img = cv2.resize(srcimg, self.input\_shape, interpolation=cv2.INTER\_AREA)
1586. **return** img, newh, neww, top, left
1588. **def** postprocess(self, frame, outs, pad\_hw):
1589. newh, neww, padh, padw = pad\_hw
1590. frameHeight = frame.shape[0]
1591. frameWidth = frame.shape[1]
1592. ratioh, ratiow = frameHeight / newh, frameWidth / neww
1593. classIds = []
1594. confidences = []
1595. boxes = []
1596. **for** detection **in** outs:
1597. scores, classId = detection[4], detection[5]
1598. **if** scores > self.confThreshold:  # and detection[4] > self.objThreshold:
1599. x1 = int((detection[0] - padw) \* ratiow)
1600. y1 = int((detection[1] - padh) \* ratioh)
1601. x2 = int((detection[2] - padw) \* ratiow)
1602. y2 = int((detection[3] - padh) \* ratioh)
1603. classIds.append(classId)
1604. confidences.append(scores)
1605. boxes.append([x1, y1, x2, y2])
1607. # # Perform non maximum suppression to eliminate redundant overlapping boxes with
1608. # # lower confidences.
1610. #indices = cv2.dnn.NMSBoxes(boxes, confidences, self.confThreshold, self.nmsThreshold)
1611. indices = self.NMSBoxes(boxes, confidences, self.nmsThreshold)
1613. **for** ind **in** indices :
1614. frame = self.drawPred(frame, classIds[ind], confidences[ind], boxes[ind][0], boxes[ind][1], boxes[ind][2], boxes[ind][3])
1615. **return** frame
1617. **def** drawPred(self, frame, classId, conf, x1, y1, x2, y2):
1618. # Draw a bounding box.
1619. cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), thickness=2)
1621. label = '%.2f' % conf
1622. text = '%s:%s' % (self.classes[int(classId)], label)
1623. #print("classId:",classId)
1625. # Display the label at the top of the bounding box
1626. labelSize, baseLine = cv2.getTextSize(text, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 1)
1627. y1 = max(y1, labelSize[1])
1628. cv2.putText(frame, text, (x1, y1 - 10), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (0, 255, 0), thickness=1)
1629. **return** frame
1631. **def** detect(self, srcimg):
1632. img, newh, neww, top, left = self.letterBox(srcimg)
1633. img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)
1634. img = img.astype(np.float32) / 255.0
1635. blob = np.expand\_dims(np.transpose(img, (2, 0, 1)), axis=0)
1637. t1 = time.time()
1638. outs = self.net.run(None, {self.net.get\_inputs()[0].name: blob})[0]
1639. cost\_time = time.time() - t1
1640. #print("type:",type(outs))
1642. **if** outs **is** **not** None **and** len(outs) > 0:
1643. **print**(outs[0][5])
1644. **else**:
1645. **print**(-1)
1647. #srcimg = self.postprocess(srcimg, outs, (newh, neww, top, left))
1648. #infer\_time = 'Inference Time: ' + str(int(cost\_time \* 1000)) + 'ms'
1649. #cv2.putText(srcimg, infer\_time, (5, 20), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (0, 0, 0), thickness=1)
1650. **return** outs#srcimg
1652. **def** NMSBoxes(self,boxes, scores, threshold):
1653. **if** len(boxes) == 0:
1654. **return** []
1655. boxes = np.array(boxes)
1657. # ����߽�����
1658. areas = (boxes[:, 2] - boxes[:, 0]) \* (boxes[:, 3] - boxes[:, 1])
1660. scores = np.array(scores)
1662. # �����Ŷȵ÷ֽ�������߽��
1663. order = scores.argsort()[::-1]
1665. keep = []
1666. **while** order.size > 0:
1667. i = order[0]
1668. keep.append(i)
1670. # ���㽻����������
1671. xx1 = np.maximum(boxes[i, 0], boxes[order[1:], 0])
1672. yy1 = np.maximum(boxes[i, 1], boxes[order[1:], 1])
1673. xx2 = np.minimum(boxes[i, 2], boxes[order[1:], 2])
1674. yy2 = np.minimum(boxes[i, 3], boxes[order[1:], 3])
1676. # ���㽻������Ŀ��Ⱥ͸߶�
1677. w = np.maximum(0.0, xx2 - xx1)
1678. h = np.maximum(0.0, yy2 - yy1)
1680. # ���㽻����������
1681. inter = w \* h
1683. # ���㽻���ȣ�IoU��
1684. iou = inter / (areas[i] + areas[order[1:]] - inter)
1686. # ���� IoU С����ֵ�ı߽��
1687. inds = np.where(iou <= threshold)[0]
1688. order = order[inds + 1]
1690. **return** keep

1693. **def** yolo\_main():
1694. # ����Ĭ�ϲ���
1695. modelpath = 'LR.onnx'
1696. classfile = 'LR.names'
1697. confThreshold = 0.4
1698. nmsThreshold = 0.6
1700. '''''
1701. parser = argparse.ArgumentParser()
1702. parser.add\_argument('--modelpath', type=str, default='best.onnx', help="onnx filepath")
1703. parser.add\_argument('--classfile', type=str, default='coco.names', help="classname filepath")
1704. parser.add\_argument('--confThreshold', default=0.2, type=float, help='class confidence')
1705. parser.add\_argument('--nmsThreshold', default=0.6, type=float, help='nms iou thresh')
1706. args = parser.parse\_args()
1707. '''
1709. net = yolov5\_lite(modelpath, classfile, confThreshold, nmsThreshold=nmsThreshold)

1712. cap = cv2.VideoCapture(-1)
1713. cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)
1714. cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)

1717. # ������ͷ��ȡһ֡ͼ��
1718. ret, frame = cap.read()
1719. **if** frame **is** None:
1720. **print**("�޷���ȡͼ��")
1721. **return** None
1723. outs = net.detect(frame.copy())

1726. #return outs
1727. yolo\_main()
1728. Makefile:
1729. CXX = g++
1730. PYTHON\_CFLAGS = $(shell python3.7-config --cflags)
1731. PYTHON\_LDFLAGS = $(shell python3.7-config --ldflags)
1732. CXXFLAGS = `pkg-config --cflags opencv4` -Wall  $(PYTHON\_CFLAGS)
1733. LDFLAGS = `pkg-config --libs opencv4` -lpigpio -lpthread -lopenal $(PYTHON\_LDFLAGS) `python3.7-config --ldflags` -L/home/pi/.local/lib/python3.7/site-packages/numpy -lpython3.7
1734. TARGET = line
1735. SRCS = crossroad.cpp yuyin.cpp paodao.cpp line.cpp bule\_card.cpp red\_cone.cpp yellow\_edges.cpp qudong.cpp yolo.cpp
1736. OBJS = $(SRCS:.cpp=.o)
1737. HEADERS = g5g.h

1740. all: $(TARGET)
1742. $(TARGET): $(OBJS)
1743. $(CXX) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)
1745. %.o: %.cpp
1746. $(CXX) $(CXXFLAGS) -c $**<** **-o** $@
1748. clean:
1749. rm -f $(OBJS) $(TARGET)