第十七届全国大学生

智能汽车竞赛

技 术 报 告

华南理工大学

South China University of Technology

学　　校：华南理工大学

队伍名称：华工赤霄队

参赛队员：张　钿　钿

曹　永　豪

黄　亮　亮

郭　沈　燕

王　汝　瀚

带队教师：陈　　安

邓　晓　燕

关于技术报告和研究论文使用授权的说明

本人完全了解全国大学生智能汽车竞赛关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会可以在相关主页上收 录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料， 并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名 郭沈豪 黄亮亮 张翎翎 曹永鑫 王世瀚

带队教师签名：陈安

日　　期：2022年8月20日

摘要

本文介绍了华南理工大学完全模型组在备战第十七届智能车竞赛的期间，得出的一些技术经验和心得体会。

本队在此次比赛中采用了大赛组委会统一指定 I 型车模，以 EdgeboardFZ3B 计算卡、TC264 核心板为核心控制器，实现普通循线，以及通过深度学习部署的模型识别特殊赛道元素，以最短时间跑完整个赛道。

目录

第一章 引言

1.1 设计背景

本设计是为了参加第十七届全国大学生智能汽车竞赛而进行。全国大学生智能汽车大赛是受教育部高等教育司委托，由高等学校自动化专教学指导委员会负责主办全国大学生智能车竞赛。该项比赛已列入教育部主办的全国五大竞赛之一。自第一届来，此项大赛已成为各大高校的热点赛事。全国大学生智能汽车竞赛是以智能汽车为研究对象的创意性科技竞赛，是面向全国大学生的一种具有探索性工程实践活动，是教育部倡导的大学生科技竞赛之一。本届全国大学生智能车大赛全国总决赛将于将于 2022 年 8 月 22 日至 25 日在南京信息大学举行。

比赛涉及控制、传感技术、电子、计算机、机械等多个学科的专业知识，对学生的知识融合和动手能力的培养，对高等学校控制及汽车电子学科学术水平的提高，具有良好的推动作用。

大赛分为四轮电磁组，四轮摄像头组，多车编队组，平衡单车组，无线充电组平衡信标组，智能视觉组，极速越野组，完全模型组和创意组十个大组别，本文介绍的是针对完全模型组比赛进行的智能车设计，采用委会统一提供的竞赛 i 车模，选用百度 EdgeboardFZ3B 作为主控电路，选用杰锐微通 HX200D 摄像头作为赛道检测的主要方式，并针对本届比赛规则，设计传感器检测方案、方向控制及速度控制算法等，最终实现一套能够自主识别路线的智能车控制硬件、软件系统。比赛以准确和速度为主要评判标准，因此，设计主要以准确稳定提取赛道信息以及快速控制方向和速度为主要设计标准。

1.2 系统设计框架介绍

系统是以检测黑白赛道为基础，通过单片机处理信号实现对车体控制，实现车体能够准确沿着预设路径寻迹。系统电路部分需要包括单片机控制单元、电机驱动电路等部分，除此之外系统还需要一些外部设备，例如编码器测速、

伺服器控制转向、直流电机驱动车体。综上所述，本智能车系统包含了以下几个模块：根据以上系统方案设计，赛车共包括七大模块：

EdgeboardFZ3B 主控模块、传感器模块、电源模块、电机驱动模块、速度检测模块及辅助调试模块。各个模块作用如下：

1.Edgeboard 主控模块，作为整个智能车的控制中心，将采集摄像头信息、编码器等传感器的信号，根据控制算法做出控制决策，驱动电机完成对智能车的控制。

2.传感器模块，作为智能车获取赛道信息的关键，可以通过一定的前瞻性，提前感知前方的赛道信息，为智能车的控制中心做出决策提供必要的依据和充足的反应时间。

3.电源模块，将电池电源转化为合适的不同电压以驱动不同的模块。

4.电机驱动模块，驱动电机完成智能车的加减速控制和转向控制。

5.速度检测模块，检测反馈智能车后轮的转速，用于速度的闭环控制。

6.辅助调试模块，主要用于智能车系统的功能调试、赛车状态监控等方面。

系统整体框图如下图 1.1 所示。

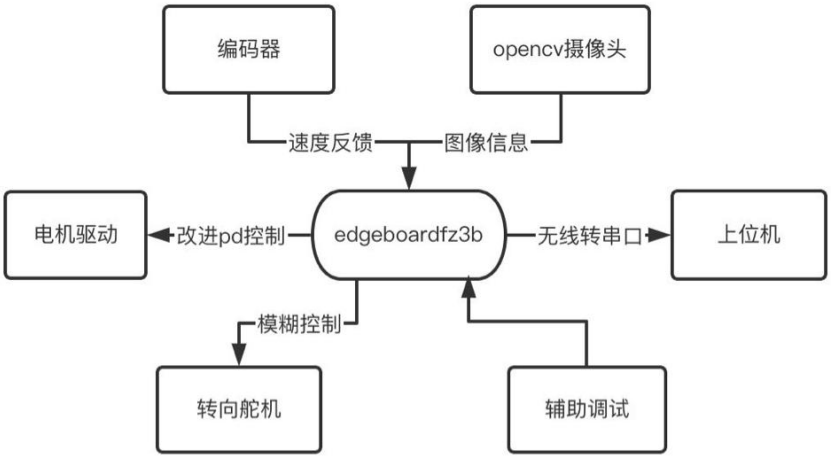

图 1.1 单片机程序设计框架

第二章 智能车机械结构调整与优化

2.1 车体参数

车体机械结构参数包括各模块位置分布，重心分布，前轮及舵机参数调教，摄像头前后位置以及高度，轮子抓地力，提高摩擦力提高小车行驶过程中的抓地力，这些参数对小车行驶的平稳起到关键作用。

车体效果图如下所示：

图 2.1 车体效果图

以下选择几个典型参数做分析。

2.2 重心分布

小车在行驶过程中重心的分布对转弯的灵敏度以及循迹的稳定性影响很大。首先，考虑车体平衡、转向对称等因素，小车的重心必然是在车身中轴线上。其次，小车的重心越低小车的抓地力越大，行驶起来就越贴地，而重心靠后对于舵机的打角就越灵敏，并且后轮压力大，抓地力就越大，就不容易在拐弯时漂移，所以前期调车时对于元件的分布有了专门的研究。

2.3 前轮及舵机参数调整

前轮作为转向轮，调整的参数对于小车的转向有较大影响。它的调整包括主销后倾角、主销内倾角、前轮外倾角以及前轮前束，本次调整主要在于主销内倾角以及前轮前束。主销内倾角前轮主销在小车水平面上的倾斜角度，它会在小车进行转向时与主销后倾角共同作用，使得车轮与水平面产生角度，增大转向时的向心力，进而辅助转向。前轮前束即前轮在小车横向坐标轴上前端与后端的偏差形成的角度，简易来讲本次小车前轮调整为外八型。

2.4 差速机构

本组别使用的 I 车模为机械差速，理论上可以通过调节差速结构的松紧程度来实现差速，好的差速为：用手转动一侧的轮胎，另一侧的轮胎可以按反方向同速转动，电机启动后，固定住两侧轮胎，电机齿轮不会转动，此时的差速效果最优。但由于 I 车模是首次加入智能车竞赛车模家族，许多参数调整都没有往届经验参考，并且在实际调车发现差速结构优化效果并不理想，难以达到像 C 车模那样双电机差速的效果。

2.5 传感器安装

完全模型组允许使用 USB 摄像头，赛曙科技有包含一个原装摄像头，在早期使用中发现其性能并不能达到我们的要求，主要体现在其分辨率不够、帧率不够、视野角度不够等，后期选用了杰锐微通电子科技的 USB 摄像头，其具有体积小、重量轻、分辨率高、帧率高、视野广等优点，使用过程达到我们的要求。

另外在安装摄像头时发现原装的法兰座高度不够，导致其力臂太短，使得摄像头固定不够牢固，车模高速运动时车体抖动，若法兰座不能牢固锁定摄像头那将导致摄像头采集到的图像不符合我们的设想，故我们采用了力臂更长的法兰座。

2.6 车壳制作

完全模型组与其他组别明显区别于车体有外壳，套上原装外壳之后对平时调试产生极大的不方便，考虑到 Edgeboard 连接的网线、下位机 TC264 的烧录线、下位机电源开关、Edgeboard 电源开关、电机开关，并且在调车过程中发现舵机长时间运行会发烫容易损坏舵机，故后续单独给舵机加了个开关，这些人机交互的途径该如何通过外壳的影响，一开始选择自己设计一个车壳，通过在车的两侧开鸥翼门，使各类接口、开关能通过打开车门的方式连接，后续使用过程中发现自己设计的车壳受限于材料的影响，无法达到重量轻、强度又高的要求，且连接车门与车壳的连接件使用久了会有磨损，导致无法正常使用，于是弃用本方案。最后，在原装车壳的基础上进行优化，将车壳两侧底部的开孔扩大，将各类开关、接口引至此处，实现与外界人机交互。经验证，此方案基本满足调试需求。但其美观程度、风阻、重心高度等都不优秀，若后续队伍有想法，可在此基础上继续优化。

第三章 电路系统设计

综合考虑智能车的重心不左右倾斜以及各功能模块的大小，我们将电路分布在左右两块板上——右边一块为电源和驱动板，负责降压和驱动电机；左边一块为主板，负责连接所有外设。另外为了调试方便，我们还设计了一块调试板。

3.1 电源驱动板

官方要求使用的是电压为 11.1V 的锂电池，为了给各模块提供合适的工作电压，需要将 11.1V 分别降压为 6.5V、5V 和 3.3V。参考官方的原理图，我们采用了两个 TPS5430 来分别把 11.1V 降为 6.5V 和 5V（如图 3.1 所示），再使用两个 AMS1117-3.3 将 5V 将为 3.3V（如图 3.2 所示）。TPS5430 的输出电压是可调的，以图 3.1 的 5V 为例，具体计算公式如公式 1 所示。更具体的原理是 TPS5430 的 VSE 输出电压恒定为 1.221V，再通过简单基尔霍夫定律就可以得出公式 1 了。

$$Vout = (\frac{R42}{R41} + 1) \times 1.221 \qquad \text{公式 1}$$

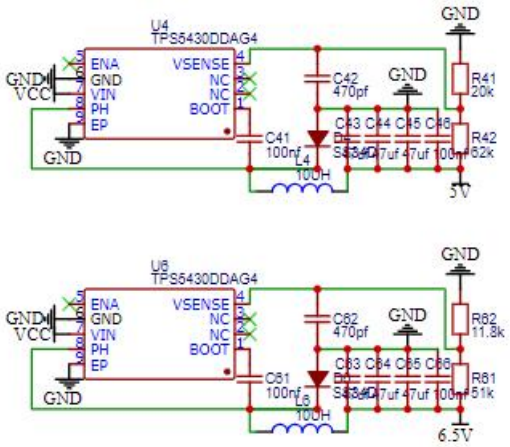

图 3.1 TPS5430 降压模块

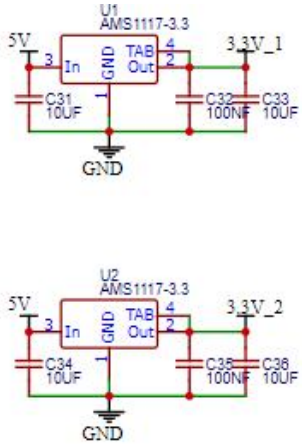

图 3.2 AMS1117-3.3 降压模块

因为比赛要求必须要戴上车壳，所以我们需要将调试使用的接口都引向车壳的开口处，包括：Egdeboard 的网络接口和 TC264D 的烧录接口。Egdeboard 的网络接口使用了两个网络口来进行引出。（如图 3.3 所示）

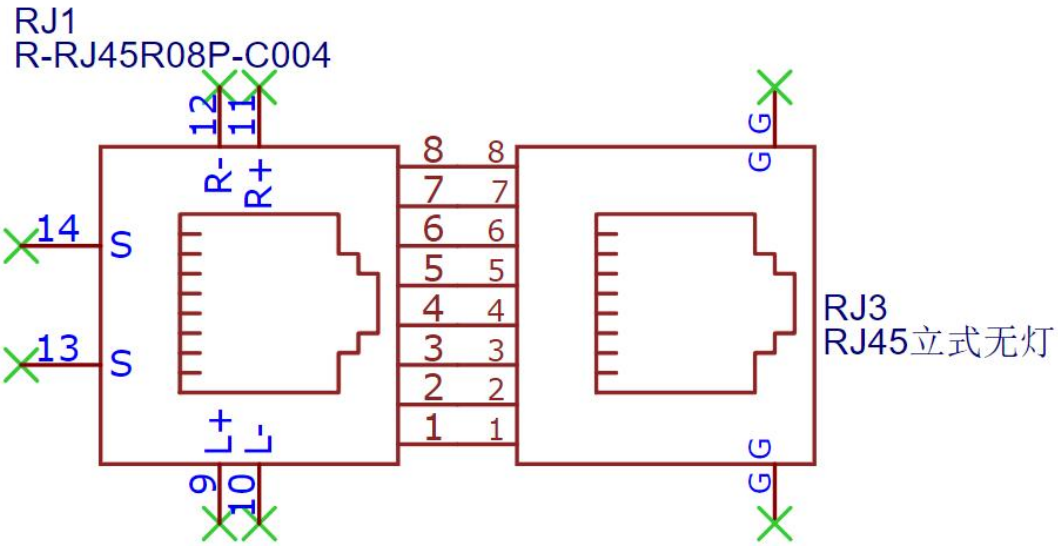

图 3.3 网线端口

驱动部分我们采用了集成半 H 桥的 BTS7960B，因为它的使用和焊接都比较简单，唯一的缺点就是有点小贵 7.5 元/个（如图 3.4 所示）。因为 BTS7960B 有一个使能口 INH，需要给高电平才能工作，而我们让它一直工作即可，所以直接给 3.3V 的高电平。IN 口为输入 PWM 信号，PWM 的占空比即为输出电压占电源电

压的比例，而最终的输出电压为两个输出电压的差值，一般只需要给一个 IN 口某个数值的占空比即可，另一个 IN 口给 0 占空比。若需电机反转则可将两个 PWM 信号互换即可。为了更好的控制速度，我们使用了电流环，信号放大器我们采用 了 INA293（如图 3.5 所示）。

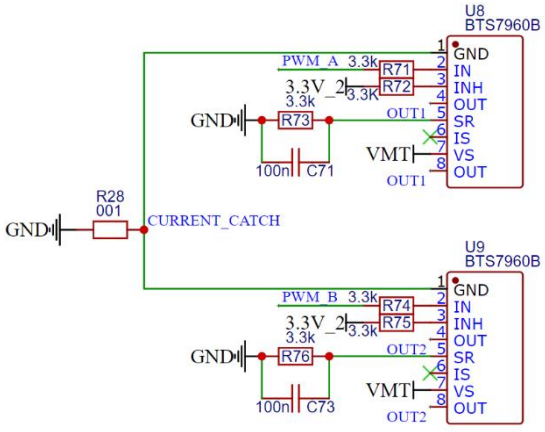

图 3.4 BTS7960B 驱动

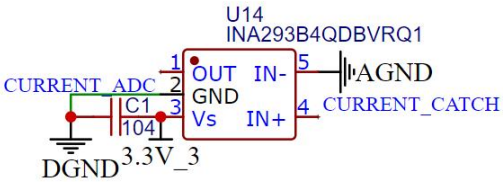

图 3.5 INA293 放大器

3.2 主板

主板需要承载核心板、舵机、编码器、无线串口、陀螺仪、板卡通信等接口（如图 3.6 所示），另外为了避免烧舵机，我们还加上了一个舵机电源开关，在跑完的时候将舵机电源关掉。第一次见到那么容易烧的舵机，开眼了。

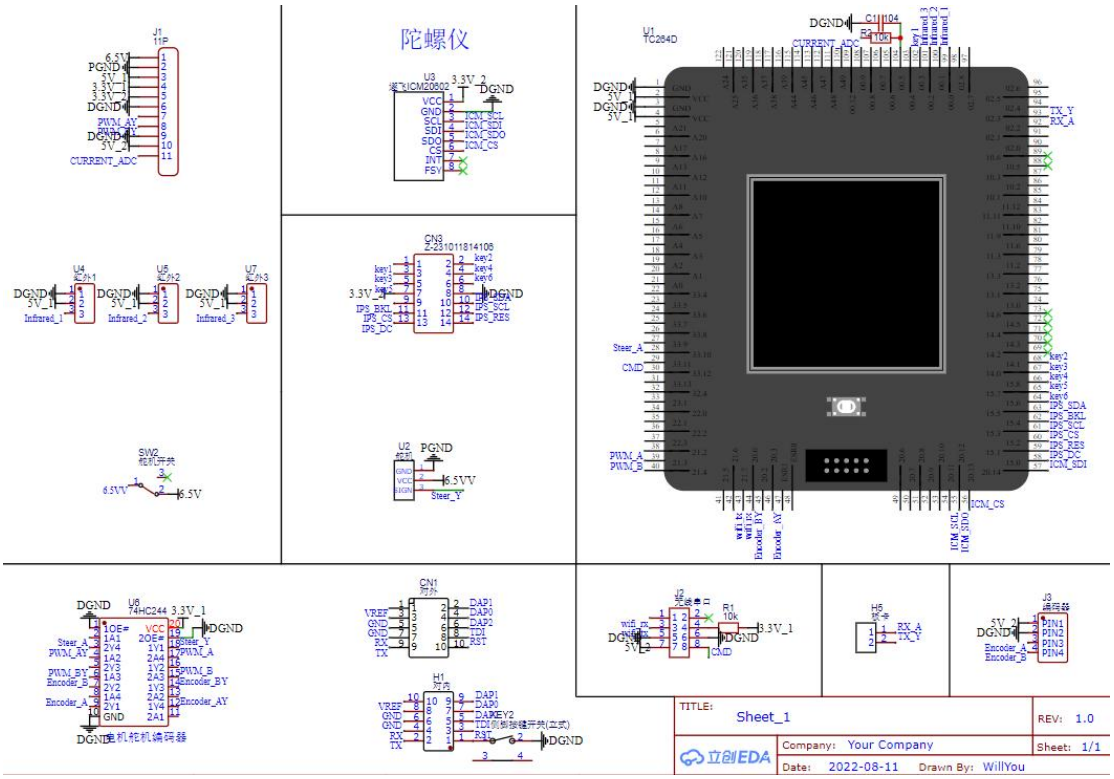

图 3.6 主板原理图

3.3 调试板

调试板上有屏幕和一个五向开关，之所以不将屏幕像别的组别一样放在车上，主要原因有：放不下了和减轻点重量。五向开关原来是可以上下左右拨动的。为了稳定屏幕不要上下摇晃，我们加了一个电容来作为屏幕支撑。（如图 3.7 所示）

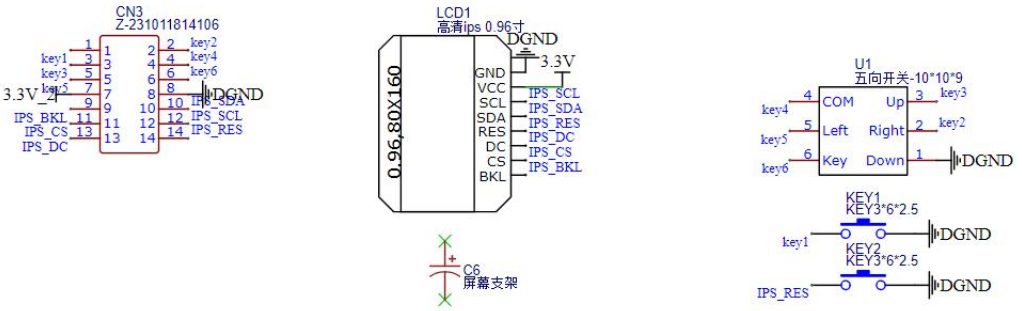

图 3.7 调试板原理图

第四章 软件开发设计

一辆小车想要成功地跑起来，其硬件是基础，而软件算法则是至关重要的“灵魂”。在本次智能车比赛中，整个软件系统包括如下的五个部分：初始化程序、图像采集处理、舵机控制、电机控制、元素识别模型部署。首先，初始化程序中进行各模块的初始化工作。然后，图像采集处理部分通过摄像头对赛道进行采集图片，在进行分析处理得到可用参数。接着，舵机与电机的控制依赖于采集处理到的数据，再根据偏差计算，应用 PID 调解与模糊控制算法，最终实现对给定的 PWM 值的调整。最后，为了实现对赛道上的特殊元素的处理，部署了相应的元素识别模型，从而可以完成特殊元素任务。

4.1 软件功能与框架

建立的程序工程是为了实现对小车机械结构与电路的结合，使其能够互相配合，合理地发挥作用，从而完成规定的任务。软件的主要功能具体包括：

（1）各部分初始化；

（2）图像的采集、处理；

（3）车模运行控制：方向控制、速度控制、路段的识别；

（4）特殊元素处理：模型的部署、车模运行调整；

（5）车模信息的参数设定：上位机监控、下位机的控制、参数设定。

程序上电运行后，便进行 edgeboard 开发板与单片机的初始化。初始化主要包括两方面。一部分是板卡和单片机对各应用模块的初始化，另一部分是应用程序中实现对车模控制的各参数的初始化。

4.2 经典 PID 算法控制策略

4.2.1 经典 PID 算法介绍

PID 控制是工程实际中应用最为广泛的调节器控制规律。问世至今 70 多年来，它以其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一。单位反馈的 PID 控制原理框图如图所示。

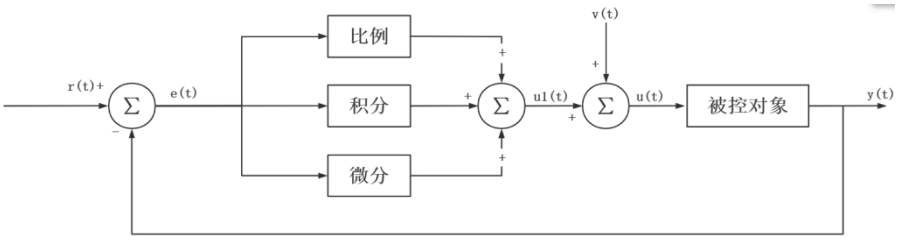

图 4.1 PID 控制原理

e(t) 代表理想输入与实际输出的误差，这个误差信号被送到控制器，控制器算出误差信号的积分值和微分值，并将它们与原误差信号进行线性组合，得到输出量

$$u = Kp * e + Ki * \int edt + Kd * \frac{de}{dt} \qquad \text{公式 2}$$

其中，kp、kd、ki 分别为比例系数、积分系数、微分系数。接着，u 被送到了执行机构，然后获得新的输出信号 u，这个新的输出信号 u 再次被送到感应器，得到新的误差信号 e(t)，这个过程就这样周而复始地进行。PID 各个参数作用基本介绍：比例调节（P）作用：是按比例反应系统的偏差，系统一旦出现了偏差，比例调节立即产生调节作用，以减少偏差。比例作用大，可以加快调节，减少误差，但是过大的比例，使系统的稳定性下降，甚至造成系统的不稳定。积分调节（I）作用：是使系统消除稳态误差，提高无差度。因为有误差，积分调节就进行，直至无差，积分调节停止，积分调节输出一常值。积分作用的强弱取决与积分时间常数 Ti，Ti 越小（注：Ki = 1/Ti），积分作用就越强。反之 Ti 大则积分作用弱，加入积分调节可使系统稳定性下降，动态响应变慢。积分作用常与另两种调节规律结合，组成 PI 调节器或 PID 调节器。微分调节（D）作用：微分作用反映系统偏差信号的变化率，具有预见性，能预见偏差变化的趋势，因此能产生超前的控制作用，在偏差还没有形成之前，已被微分调

节作用消除。因此，可以改善系统的动态性能。在微分时间选择合适情况下，可以减少超调，减少调节时间。微分作用对噪声干扰有放大作用，因此过强的加微分调节，对系统抗干扰不利。此外，微分反应的是变化率，而当输入没有变化时，微分作用输出为零。微分作用不能单独使用，需要与另外两种调节规律相结合，组成 PD 或 PID 控制器。

对连续系统中的积分项和微分项在计算机上的实现，是将上式转换成差分方程，由此实现数字 PID 调节器。

4.2.2 位置式 PID 算法

矩形数值积分代替上式中的积分项，对导数项用后向差分逼近，得到数字 PID 控制器的基本算式（位置算式）

$$u_n = Kp * e_n + \frac{1}{T_i} * \sum_{k=1}^{n} e_k T + T_d * \frac{e_n - e_{n-1}}{T} \quad \text{公式 3}$$

4.2.3 增量式 PID 算法

对位置式加以变换，可以得到 PID 算法的另一种实现形式（增量式）

$$\Delta u_n = u_n - u_{n-1} = Kp * \left[(e_n - e_{n-1}) + \frac{1}{T_i} * e_n + \frac{T_d}{T} * (e_n - 2e_{n-1} + e_{n-2}) \right] \quad \text{公式 4}$$

这种算法用来控制步进电机特别方便，对直流电机也可以采用，其实如果对控制有更高的要求或者干扰因素较多，我们可以对 PID 算法做各种改进，如用梯形法做数值积分以提高精度、将差分改成一阶数字滤波等等，在实际应用中，由于码盘采样得到的脉冲上升下降沿不够明显，使得速度采样出现不稳定和失真，但由于这些附加处理比较耗费代码的运行时间，出于代码效率和实际效果的比较，我们没有采用这些改进的方案，另外可以考虑加反向器来整波形得到较为理想的方波。

运用 PID 控制的关键是调整三个比例系数，即参数整定。

PID 整定的方法有两大类：一是理论计算整定法。它主要是依据系统的数学模型，经过理论计算确定控制器参数。由于智能车整个系统是机电高耦合的分布参数系统，并且要考虑赛道具体环境，要建立精确的智能车运动控制数学模型有一定难度，而且我们对车身机械结构经常进行不断修正，模型参数变化较频繁，可操作性不强；二是工程整定方法，它主要依赖工程经验，直接在控制系统的试验中进行，且方法简单，我们采用了这种方法，同时，我们先后实验了几种动态改变 PID 参数的控制方法。

4.3 车模控制策略

4.3.1 速度控制策略

在速度控制中，采用编码器传感器，实现闭环控制。在小车运行过程中，对于速度而言，对快速性要求较高，而对于准确性要求较低，但也不能产生太大的误差。基于这个原因，在速度控制上采用积分分离式的 PI 控制以及 BANGBANG 控制相结合，快速响应，同时减小静差。速度控制是通过脉宽调制技术（PWM）来实现的。计算所得值用于控制单片机相应管脚产生相应的波形（改变波形的占空比），将该 PWM 波形输入到电机控制芯片中，电机控制芯片将会产生相应的响应来改变电机两端的电压，从而实现变压调速。

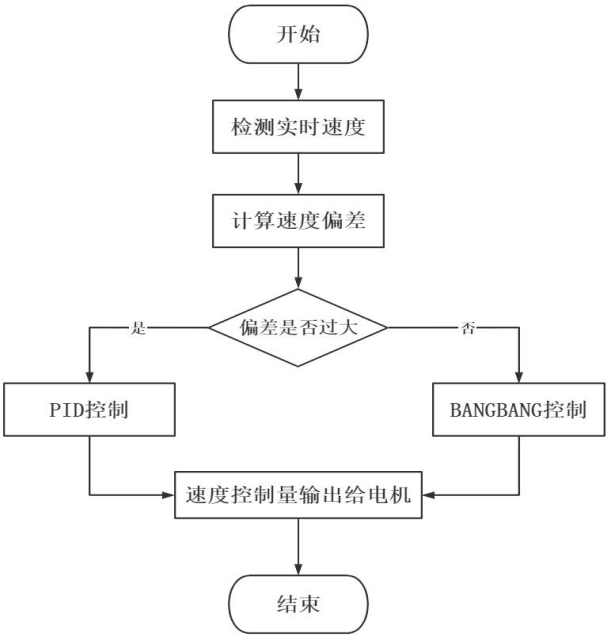

图 4.2 电机速度控制

4.3.2 舵机控制策略

对于舵机控制算法的探索，我们进行了大量的尝试，在早期调车时，我们使用的是动态 P 结合 D 的 PD 方向控制策略，但是由于经验缺乏以及与搜线算法不匹配，没有得到很好的效果。

而后我们尝试加入模糊控制算法，效果明显有了提升，模糊控制实质上是一种非线性控制，从属于智能控制的范畴。模糊控制的一大特点是既有系统化的理论，又有大量的实际应用背景。模糊控制的发展最初在西方遇到了较大的阻力；然而在东方尤其是日本，得到了迅速而广泛的推广应用。

近 20 多年来，模糊控制不论在理论上还是技术上都有了长足的进步，成为自动控制领域一个非常活跃而又硕果累累的分支。其典型应用涉及生产和生活的许多方面，例如在家用电器设备中有模糊洗衣机、空调、微波炉、吸尘器、照相机和摄录机等；在工业控制领域中有水净化处理、发酵过程、化学反应釜、水泥窑炉等；在专用系统和其它方面有地铁靠站停车、汽车驾驶、电梯、自动扶梯、蒸汽引擎以及机器人的模糊控制。

对于我们的智能车模型来说，模糊控制器的输入量为偏差值和偏差的变化量，模糊控制器进行偏差和偏差变化率的量化之后进行模糊化，而后进行解模糊输出一个特定的 PWM 值来控制舵机，模糊控制的规则表为调节模糊控制的一大难点，我们根据上位机来观察小车路径不好的地方的模糊输出值，而后来调节自己的模糊规则表，尽量使得小车能够对不同路段做出快速不同的响应。

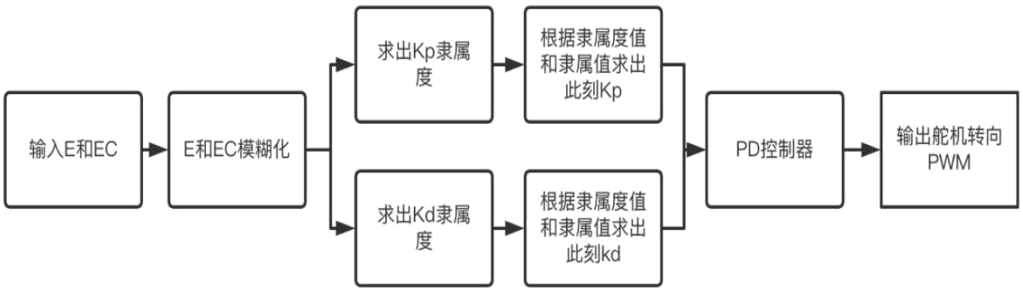

图 4.3 舵机打角控制

4.4 赛道识别思路解析

4.4.1 整体框架

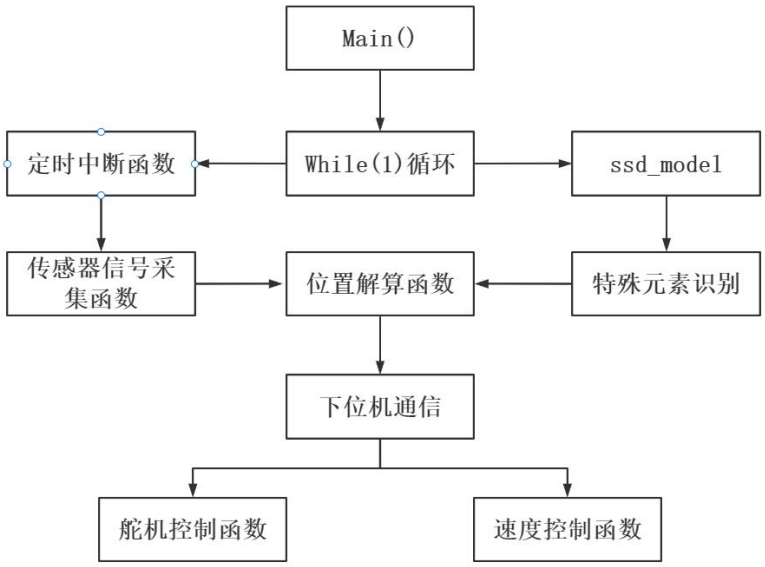

图 4.4 工程整体框架

4.4.2 位置解算

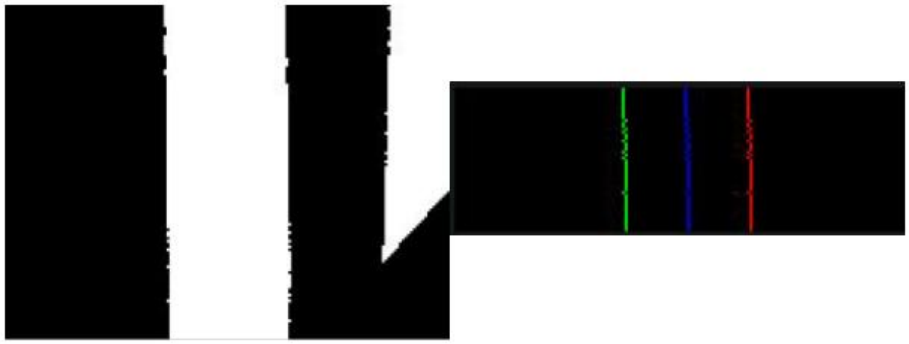

图 4.5 搜线示意图

我们首先将采集到的原始图像进行迭代搜线，从最底行中间开始向两边进行搜索，搜索到黑白边界即确定为边界，而后通过左右边界点的中点作为下一次搜索边界的起点，以此类推，通过迭代搜索完成整个搜线边界过程，而后再通过边界线与中线的位置关系来计算出不同的偏差，从而确定小车当前处于的位置。

在这一过程中，为了更好地处理搜线过程中的离散点以及更加准确地解算当前小车所处状态，我们采取了图像的逆透视处理方法。这样对采集的原始图像做了灰度处理后，并进行空间投影，可以达到更好的效果。这一方法在后面的特殊地形中也加以运用。

4.4.3 其他特殊赛道判断

十字判断：

图 4.6 十字判断示意图

十字处的特征是比较明显的，首先是左右边都会有丢线，并且左右边线上都可以搜索到两个拐点，通过拐点进行补线即可稳定通过十字。

考虑到本次赛道无圆环和坡道，所以不做识别。

4.4.4 特殊元素识别

完全模型组别需要完成的特殊元素地形包括施工区、加油站、泛行区和车库。而在完成这些赛道的过程中，需要识别多个标志。所以在官方推荐的 Peddle 飞桨平台上应用 ssd-model 进行模型训练。之后将训练后的模型部署在 edgeboard 上，即可运用。

进行模型训练的样本集依赖于 labelImg 进行采集标记,再上传至平台即可。

第五章 辅助调试工具

5.1 开发调试工具

在整个开发调试过程中，采用 visual studio code 远程连接 edgeboard，远程连接后即可在电脑上编写程序于 edgeboard。

除此之外，下位机 TC264 编写程序使用的是 AURIX Development Studio（ADS），该软件是英飞凌推出在针对自家 AURIX 芯片的免费编译环境，软件使用无需 license，ADS 继承了编译器、调试器、iLLD 底层开发库等必备组件。

5.2 人机交互系统

5.2.1 ips 液晶屏

通过对多种 oled 和 LCD 进行对比和测试，最终选用 ips 全彩液晶屏来实现人机交互界面，这种屏幕具有如下优缺点。

图 5.1 IPS 屏幕实物图

优点：

（1）显示区域大。屏幕尺寸为 1.3 寸，显示区域为 23.4*23.4(mm)，因此能够同时显示多个变量，以及能够实现图像和变量同时查看；

（2）色彩丰富。可通过对颜色调整调整来实现在屏幕上观察各种类型赛道的各种标志位。

缺点：

（1）屏幕体积较大，对硬件电路板的设计要求较高；

（2）较为耗电，会过多的占用电池资源；

（3）传输数据可能花费较多单片机资源，我们使用示波器测量过时间，发现若在主线程中刷新，需要耗时数个 ms，这就会影响到程序运行时序问题。但这个问题是可以通过软件部分进行修改而避免的。

考虑到以上优缺点，最终我们采用通过外接屏幕的方式实现人机交互，需要调试时可通过外接借口连接上搭载屏幕的电路板，实现可以在屏幕上通过五向开关进行参数的调整。通过这种人机交互界面的构建，对于小车参数整定的效率大大提升。

5.2.2 无线串口模块及上位机调试

对于智能车在赛道中高速运行时的各种状况，我们无法依靠肉眼观测得出，无法详细了解车模运行时的各种状况，就很难对各个环节的参数进行精确地调整。为了解决问题，就需要单片机将智能车运行时的各种数据和信息反馈到上位机，这里我们使用的是逐飞科技的无线串口模块配合匿名上位机软件对智能车进行实时监控。

逐飞科技的无线串口模块具有低功耗、高速率、体积小的特点，完全符合我们对智能车电路精简的要求和对实时信号传输速率的要求。匿名上位机功能强大，可同时查看多个数据的波形满足我们同时对大量数据进行监测的要求。

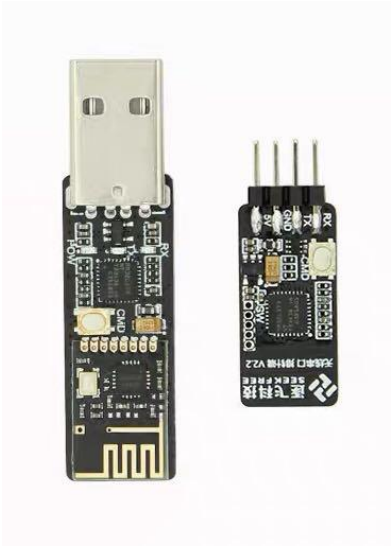

图 5.2 逐飞科技无线串口模块实物图

第六章 结论

6.1 车模主要技术参数

车模整体尺寸	长	28cm
	宽	19cm
	高	35cm
	重量	
传感器	六轴陀螺仪	1 个
	光电编码器	1 个
	杰锐微通 HX200D 摄像头	1 个
微处理器	Egdeboard	1 个
	TC264D	1 个
电机	CS-3120 舵机	1 个
	RS-555 直流电机	1 个
电池	格氏电池 2200mah 30C 3s	1 个
电路板	电源驱动板	1 个
	主板	1 个
无线通信	无线串口 2.4GHz	1 个

表 6.1 车模主要技术参数表

6.2 总结感想

感谢学校，给我们提供比赛场地，让我们不用为寻找场地而分神烦恼；感谢陈安老师和邓晓燕老师，为我们争取到学校的很多资源，包括场地和资金报销；感谢辛君诺、李子锋、刘炜斌和谢勇强学长为我们提供的大力支持；感谢所有成员，没有他们，华工赤霄队就无法进入国赛角逐。

准备比赛的一年有种梦回高三的感觉，身心疲惫时常有之，但这也告诉我们，虽然我们从高中毕业了，但是高三奋战高考的毅力并未消退，反而历久弥坚，生生不息。而且让我们感受到了大学的学习和高三的只有上课考试不一样，还有各种各样的比赛与合作，而比赛所需要的知识和课内挺不一样的，我们需要利用在课内学习所掌握的能力来学习比赛所需的知识并应用，感受到理论与实践之间的那条鸿沟，并努力跨过去。

从大学开始，每个人晋升的评价体系不再只是考试了，还有各种各样的竞赛，提供了多元化的发展途径，喜欢上课-作业-考试体系的同学可以继续保持，不喜欢的同学则可以参加比赛，展示自己的另一种风采，同样也会得到官方的认可。这与中小学时期的培训班式的竞赛还是挺不一样的，没有围墙。这与周围人的评价是不一样的，有证书。

天意终究难参，假若登顶成憾，与君同添青史几传，成败也当笑看。

参考文献

[1] 卓晴，黄开胜，邵贝贝．学做智能车 [M]．北京：北京航空航天大学出版社．2007

[2] 刘金琨.先进 PID 控制及其 MATLAB 仿真[M].北京.电子工业出版

[3] 贾翔羽,季厌庸,丁芳.前馈-改进 PID 算法在智能车控制上的应用

[4] 李仕伯,马旭,卓晴.基于磁场检测的寻线小车传感器布局研究[J].电子产品世界.2009

[5] 谭浩强著．C 程序设计．北京：清华大学出版社.2003

[6] 臧杰，阎岩．汽车构造[M] ．北京．机械工业出版社．2005

[7] 尹怡欣，陶永华．新型 PID 控制及其应用．北京：机械工业出版社.1998

[8] 张军．AVR 单片机应用系统开发典型实例．北京：中国电力出版社.2005

[9] 蔡述庭．“飞思卡尔”杯智能汽车竞赛设计与实践 [M]．北京：北京航空航天大学出版社.2012

[10] SUNPLUS. PID 调节控制做电机速度控制. 2006

[11] 飞思卡尔智能车----模糊 PID 算法通俗讲.2018

https://blog.csdn.net/weixin_36340979/article/details/79168052

[12] 张敏杰. 转弯差速在智能车上的运用[A]. 2011

[13]木南创智. PID 控制器开发笔记之二：积分分离 PID 控制器的实现. 2018

https://blog.csdn.net/foxclever/article/details/80274790

[14]努力挣扎的小菜菜. 鲁棒控制（棒棒控制原理）

https://blog.csdn.net/weixin_44513216/article/details/105635914

[15]☆Ronny 丶 . 边缘检测

https://www.cnblogs.com/ronny/p/4001910.html

附录

程序源代码

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <Camera.h>
#include <Car.h>
#include <ctime>
#include <Serial.h>
#include <string>
#include <Control.h>
#include <Detection.h>
#include <stdio.h>
#include <sstream>
using namespace std;
using namespace cv;

typedef union
 {
    char CharNum[4];
    float FloatNum;
 }DataUnion;
char data_buffer[2048]={0};

int main(int agrc,char **agrv)
{
    int ret = 0;
    clock_t startTime, endTime;

    /**************摄像头初始化 *************/
    VideoCapture camera(1);
   Camera image(camera);
   Camera& p_image = image;
   Car marscar;
   Car& p_car=marscar;
```

```
   /****模糊控制初始化******************/
   Fuzzy_init();
   /**************串口初始化 *************/
    DataUnion error;
   error.FloatNum=1.5;
   WzSerialportPlus serial;
serial.setReceiveCalback([&](char* data,int length)
{
      std::string responsePrefix = "received: ";
      std::string response(data,length);
      response = responsePrefix + response;
      serial.send((char*)response.c_str(),response.length());
}
);
   serial.open("/dev/ttyPS1",115200,1,8,'n');
   /*************目标检测初始化*********************/
  std::string system_config_path;
  std::string model_config_path;
  system_config_path = "../source/image.json";
  std::cout << "SystemConfig Path:" << system_config_path << std::endl;
  g_system_config = make_shared<SystemConfig>(system_config_path);
  model_config_path = g_system_config->model_config_path;
  std::cout << "ModelConfig Path:" << model_config_path << "\n";
  g_model_config = make_shared<ModelConfig>(g_system_config->model_config_path);
std::shared_ptr<CaptureInterface> capture = nullptr;
  capture =createCapture(g_system_config->input_type, g_system_config->input_path);   //创建
摄像头读取线程 队列形式
     if (capture == nullptr)
 {
    exit(-1);
   }
   ret = predictorInit();
if (ret != 0)
{
     std::cout << "Error!!! predictor init failed .\n";
     exit(-1);
    }
   std::vector<PredictResult> predict_results;
   std::shared_ptr<Timer> timer = make_shared<Timer>("Predict", 100);
```

```
  while (1)
  {
    FrameWrapper frame_wraper = capture->getFrame();
    image.o_frame=frame_wraper.frame;
    marscar.judge_stage(p_image);
    FuzzyPID(p_car);
    error.FloatNum=last_servo_out;
    for(int i=0;i<4;i++)
    {
      data_buffer[i]=error.CharNum[i];
    }
    data_buffer[4]=marscar.road;
    serial.send(data_buffer,5);
  }
  return 0;
}
using namespace std;
using namespace cv;
uint8_t use_num_left = 0;  //左边丢线数量
uint8_t use_num_right = 0;  //右边线丢线数量
uint8_t lost_num_left = 0;
uint8_t lost_num_right = 0;
uint8_t left_break;      //左边界不连续标志
uint8_t right_break;      //右边界不连续标志

uint8_t start_left = 0;  //有效顶行
uint8_t end_left = 0;    //有效底行
uint8_t start_right = 0;  //有效顶行
uint8_t end_right = 0;    //有效底行
uint8_t use_end = 0;
uint8_t use_start = 0;

uint8_t left_inflexion = 0;   //左边线拐点标志
uint8_t right_inflexion = 0;  //右边线拐点标志
int left_inflexion_row = 0; //左边线拐点行
int right_inflexion_row = 0; //右边线拐点行
double left_slope = 0.0;    //左边线斜率
double right_slope = 0.0;   //右边线斜率
```

```
uint8_t flag_time=0;

uint8_t FLOODED_slope=0;

std::vector<PredictResult> predict_results;
std::shared_ptr<Timer> timer = make_shared<Timer>("Predict", 100);

float derror_max=0.0;
uint8_t check_left1[120] = {
  49, 48, 47, 46, 46, 45, 44, 44, 43, 42, 41, 41, 40, 39, 39, 38, 37, 36, 36, 35, 34, 34, 33, 32, 31, 31,
 30, 29, 29, 28,
  27, 26, 26, 25, 24, 24, 23, 22, 21, 21, 20, 19, 19, 18, 17, 16, 16, 15, 14, 14, 13, 12, 11, 11, 10,  9,
 9,  8,  7,  6,
   6,  5,  4,  4,  3,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
   1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1
};  //逆透视边界

uint8_t check_right1[120] = {
  111, 112, 112, 113, 114, 114, 115, 116, 117, 117, 118, 119, 119, 120, 121, 122, 122, 123, 124, 1
24, 125, 126, 127, 127, 128, 129, 129, 130, 131, 131,
  132, 133, 134, 134, 135, 136, 136, 137, 138, 139, 139, 140, 141, 141, 142, 143, 143, 144, 145, 1
46, 146, 147, 148, 148, 149, 150, 151, 151, 152, 153,
  153, 154, 155, 156, 156, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158 ,1
58 ,158, 158, 158, 158, 158, 158, 158, 158, 158, 158,
  158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 1
58, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158
};  //逆透视边界

float absfloat(float x)
{
  if(x>0)
    return x;
  else
    return x*(-1);
}

uint8_t max(uint8_t a, uint8_t b)
{
```

```
    if (a >= b)
        return a;
    else
        return b;
}

uint8_t abstract(uint8_t a, uint8_t b)
{
    if (a > b)
        return a - b;
    else
        return b - a;
}
bool tu_line(uint8_t startline, uint8_t endline, uint8_t* arr)
{
    uint8_t mid=(startline+ endline)/2;

    if(((arr[startline] + arr[endline]) / 2) < arr[mid])
    {
        return true;
    }
    else
        return false;
}
bool judge_straight_circle(uint8_t startline, uint8_t endline, uint8_t* arr)
{
    uint8_t mid=(startline+ endline)/2;
    uint8_t x=abstract((arr[startline] + arr[endline]) / 2, arr[mid]);
    if (x < 2)
        return true;
    else
        return false;
}
bool judge_straight_cross(uint8_t startline, uint8_t endline, uint8_t* arr)
{
    uint8_t mid=(startline+ endline)/2;
    uint8_t x=abstract((arr[startline] + arr[endline]) / 2, arr[mid]);
    if (x < 3)
        return true;
```

```
    else
        return false;
}
int search_left_inflexion(uint8_t startline, uint8_t endline, uint8_t* arr)  //左边界单调递增
{
    int last = startline;
    int flag1 = 0;  //增加
    int flag2 = 0;  //增加
    int i = 0;
    int j = 0;
    for (i = startline; i <= endline; i++)
    {
        if (arr[i] > arr[last])  //如果单调递增  则继续
        {
            last = i;
            flag1 ++;
        }
        else if (arr[i] < arr[last])
        {
            break;   //跳出
        }//出现递减
    }
    for (j = i; j <= endline; j++)
    {
        if (arr[j] < arr[last])
        {
            last = j;
            flag2++;   //跳出
        }//出现递减
    }
    if (flag1>2 && flag2>3 )   //如果居中 则是拐点
    {

        return i-1;
    }
    else                //如果不是 返回无拐点
        return -1;

}
```

```
int search_right_inflexion(uint8_t startline, uint8_t endline, uint8_t* arr)  //左边界单调递增
{
   int last = startline;

   int flag1 = 0;  //增加
   int flag2 = 0;  //增加
   int i = 0;
   int j = 0;
   for ( i = startline; i <= endline; i++)
   {
      if (arr[i] < arr[last])  //如果单调递减  则继续
      {
         last = i;
         flag1 ++;
      }
      else if (arr[i] > arr[last])
      {
         break;//跳出
      }

   }
   for (j = i; j <= endline; j++)
   {
      if (arr[j] > arr[last])
      {

         last = j;
         flag2++;   //跳出
      }//出现递减
   }

   if (flag1 > 2 && flag2 > 3)   //如果居中 则是拐点
   {
      return i-1;
   }
   else                 //如果不是 返回无拐点
      return -1;

}
```

```
Car::Car()
{
    road='a';
    error_mode=2;
    road_mode = RoadMode::NORMAL;
    s_error =0;    //偏差
    s_slope=0;
    s_derror =0;   //偏差变化率
    speed=0;    //速度
    acceleration=0;  //加速度
    Lstraight_repair=false;   //直线补直开关
    Rstraight_repair=false;
    CROSS_OPEN=true;
    Lcross_repair=false;   //十字补线开关
    Rcross_repair=false;

    CONSTRUCTION_stage=0;  //记录施工区的阶段
    CONSTRUCTION_FLAG=0;
    FLOODED_stage=0;    //记录泛行区所处状态
    PEDESTRIAN_stage=0;
    FLOODED_FLAG=0;
    FLOODED_turn=0;
    STATION_flag=0;   //记录加油站所处状态
    STATION_stage=0;
    WHITEONE=0;
    WHITETWO=0;
    FINAL_flag=0;    //完赛标志
  PEDESTRIAN_OPEN=true;
}

void Car::get_useline(Camera& m_camera)
{
   start_left = 0;  //左边 有效顶行
   end_left = 0;   //有效底行
   start_right = 0;  //右边有效顶行
   end_right = 0;   //有效底行

   uint8_t top_tmp = 0;
   uint8_t but_tmp = 0;
```

```
// 循环条件是 r < n
while (top_tmp < 65)
{
  if (m_camera.lost_left[top_tmp] == 0)
  {
    // 满足，则不断扩大 r
    ++top_tmp;
  }
  else
  {
    // 不满足，则累加 r 和 l 更新为 r
    if ((top_tmp - but_tmp) > (start_left - end_left))
    {
      start_left = top_tmp;
      end_left = but_tmp;
    }

    ++top_tmp;
    but_tmp = top_tmp;
  }
}
if ((top_tmp - but_tmp) > (start_left - end_left))
{
  start_left = top_tmp;
  end_left = but_tmp;
}
if(start_left>1)
  start_left--; //退一行

top_tmp = 0;
but_tmp = 0;

//右边有效行寻找
while (top_tmp < 65)
{
  if (m_camera.lost_right[top_tmp] == 0)
  {
    // 满足，则不断扩大 r
```

```
            ++top_tmp;
        }
        else
        {
            // 不满足，则累加 r 和 l 更新为 r
            if ((top_tmp - but_tmp) > (start_right - end_right))
            {
                start_right = top_tmp;
                end_right = but_tmp;
            }

            ++top_tmp;
            but_tmp = top_tmp;
        }
    }
    if ((top_tmp - but_tmp) > (start_right - end_right))
    {
        start_right = top_tmp;
        end_right = but_tmp;
    }
    if(start_right>1)
        start_right--;
    use_num_left=start_left-end_left;
    use_num_right=start_right-end_right;
    use_end = end_left>end_right?end_left:end_right;  //底行取大
    use_start = start_left<start_right?start_left:start_right;  //顶行取小
}

void Car::get_miderror(Camera& m_camera)
{
    float error=0.0;
    int num=0;
    int width=44;
     memset(m_camera.centerline, 79, sizeof(m_camera.centerline));
    for (int i = 0; i < 60; i++)
    {
        if(m_camera.lost_left[i]==0 && m_camera.lost_right[i]==0)  //都不丢线 取中间
        {
            m_camera.centerline[i] =(m_camera.leftline[i]+m_camera.rightline[i])/2;
```

```
        error +=   m_camera.centerline[i] - 79;
        num++;
    }
    else if(m_camera.lost_left[i]==0 && m_camera.lost_right[i]==1)  //左边有效
    {
        if(m_camera.leftline[i]<65)
        {
        m_camera.centerline[i]=m_camera.leftline[i]+30;
        }
        else
        {
        // m_camera.centerline[i]=(m_camera.leftline[i]+159)/2;
        m_camera.centerline[i]=159;
        }
        error +=  m_camera.centerline[i] - 79;
        num++;
    }
    else if(m_camera.lost_left[i]==1 && m_camera.lost_right[i]==0)  //右边有效
    {
        if(m_camera.rightline[i]>94)
        {
            m_camera.centerline[i]=m_camera.rightline[i]-30;
        }
        else
        {
            m_camera.centerline[i]=0;
        }
        error +=  m_camera.centerline[i] - 79;
        num++;
    }
}
 /////偏差进行加权处理  远处权重大 前瞻更远  打角更提前
for (int i = 0; i < 40; i++)   //前 25 行权重 0.8
{
    float temp = (m_camera.centerline[i] - 80);
    if (temp != 0)
        num++;
    error += temp * 0.7; //0.7
}
```

```
  for (int i = 40; i < 60; i++)        //后 25 行权重 1.8
  {
     float temp = (m_camera.centerline[i] - 80);
     if (temp != 0)
        num++;
     error += temp * 1.5;//2
  }
     if(num!=0)
     error = error / num;
     else
     error=0.00001;
     s_error = error;   //计算偏差
      )/num;

     s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
     s_last_error=s_error;
}

void Car::get_miderror_sp(Camera& m_camera)
{
  float error=0.0;
  int num=0;
  int width=44;
   memset(m_camera.centerline, 79, sizeof(m_camera.centerline));
  for (int i = 0; i < 65; i++)
  {
    if(m_camera.lost_left[i]==0 )
    {
        m_camera.centerline[i]=m_camera.leftline[i]+22;
       error +=  m_camera.centerline[i] - 79;
       num++;
    }
    else if(m_camera.lost_left[i]==1 && m_camera.lost_right[i]==0)
    {
        m_camera.centerline[i]=50;
         error +=  m_camera.centerline[i] - 79;
       num++;
    }
  }
```

```
    if(num!=0)
    error = error / num;
    else
    error=0.00001;
    s_error = error;   //计算偏差
    //s_slope=((double)m_camera.centerline[use_start]- (double)m_camera.centerline[use_end]
)/num;

    s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
    s_last_error=s_error;
}
void Car::get_miderror_LL(Camera& m_camera)
{
  float error=0.0;
  int num=0;
  int width=44;
   memset(m_camera.centerline, 79, sizeof(m_camera.centerline));
  for (int i = 0; i < 65; i++)
  {
    if(m_camera.lost_left[i]==0 && m_camera.lost_right[i]==0)  //都不丢线 取中间
    {
      m_camera.centerline[i] =(m_camera.leftline[i]+m_camera.rightline[i])/2;
      error +=   m_camera.centerline[i] - 79;
      num++;
    }
    else if(m_camera.lost_left[i]==0 && m_camera.lost_right[i]==1)  //左边有效
    {
      m_camera.centerline[i]=(m_camera.leftline[i]+159)/2;
      error +=  m_camera.centerline[i] - 79;
      num++;
    }
    else if(m_camera.lost_left[i]==1 && m_camera.lost_right[i]==0)  //右边有效
    {
      m_camera.centerline[i]=(m_camera.rightline[i]+10)/2;
      error +=  m_camera.centerline[i] - 79;
      num++;
    }
  }
    if(num!=0)
```

```
    error = error / num;
    else
    error=0.00001;
    s_error = error;  //计算偏差
    //s_slope=((double)m_camera.centerline[use_start]- (double)m_camera.centerline[use_end]
)/num;

    s_derror = s_error - s_last_error;//计算本次 D 值  //计算偏差变化率
    s_last_error=s_error;

}
void Car::judge_stage(Camera& m_camera)  //判断当前所处道路状态
{
  if(road_mode==RoadMode::NORMAL)  //普通
      NORMAL_HANDLE(m_camera);
  // else if(road_mode==RoadMode::CIRCLE)  // 圆环
  //    CIRCLE_HANDLE(m_camera);
  else if(road_mode==RoadMode::CONSTRUCTION)  //施工区
    CONSTRUCTION_HANDLE(m_camera);
  else if(road_mode==RoadMode::FLOODED)     //三岔
    FLOODED_HANDLE(m_camera);
  else if(road_mode==RoadMode::STATION)   //入库
    STATION_HANDLE(m_camera);
  else if(road_mode==RoadMode::PEDESTRIAN_END)   //入库
    PEDESTRIAN_HANDLE_END(m_camera);
 /* else if(road_mode==RoadMode::PROHIBITORY)   //禁行障碍
    PROHIBITORY_HANDLE(m_camera);

  else if(road_mode==RoadMode::STOP)    //停车
    STOP_HANDLE(m_camera);        */

}

void Car::NORMAL_HANDLE(Camera& m_camera)   //普通情况下 巡线并识别标志
{
 int detection_result=0;
   predict_results = predict(m_camera.o_frame, timer);
   detection_result=DetectionResult(predict_results);
```

```
    if(detection_result==2)
     {
        road='f';
       road_mode=RoadMode::FLOODED;
       CROSS_OPEN=false;
      // CIRCLE_OPEN=true;
       PEDESTRIAN_OPEN=false;
       imwrite("FLOODED.jpg",m_camera.o_frame);
       FLOODED_stage=1;
      // cout<<"road_mode:三岔泛行区"<<endl;
      // cout<<clock()<<endl;
        return;
    }


   else
       road_mode=RoadMode::NORMAL;
          m_camera.image_processing();  //图像处理
   /***************停车扫描*********************/
     if(FINAL_flag==2)
  {
      int overturn=0;
      for(int i=40;i<120;i++)
      {
          if((int)m_camera.i_frame.at<uchar>(80, i) == 255 && (int)m_camera.i_frame.at<uchar>(8
0, i+1)==0)
                overturn++;
          else if((int)m_camera.i_frame.at<uchar>(80, i) == 0 && (int)m_camera.i_frame.at<uchar>(
80, i+1)==255)
                overturn++;
      }
      if(overturn>10)
      {
         road_mode=RoadMode::PEDESTRIAN_END;
         road='s';
         FINAL_flag=0;
         s_error=-35;
         s_derror=0;
         return;
```

```
        }
    }
    m_camera.PROCESS_GetLeftBoundary();
    m_camera.PROCESS_GetRightBoundary();
      m_camera.check_repair();     //逆透视边界修正
    get_useline(m_camera);       //获取有效行
    get_roadmode(m_camera);    //判断圆环和是否需要补线
    get_repair(m_camera);    // 十字补线和直线补直

    if(error_mode==1)
    get_miderror_sp(m_camera);  //出泛行区后的 V 型
    else
    get_miderror(m_camera);   //获取偏差  //普通 十字
     m_camera.draw_boundary_frame();
     m_camera.show_o_frame();
     m_camera.show_r_frame();
     m_camera.show_i_frame();
     m_camera.show_b_frame();

}
void Car::get_roadmode(Camera& m_camera)
{

    left_inflexion_row = -1;
    right_inflexion_row = -1;
    Lstraight_repair=false;
    Rstraight_repair=false;
    Lcross_repair=false;
    Rcross_repair=false;

    bool left_straight=false;
    bool left_inflexion2down=false;
    bool left_inflexion2up=false;

    bool right_straight=false;
    bool right_inflexion2down=false;
    bool right_inflexion2up=false;
    //左右边线极值点寻找
    left_inflexion_row= search_left_inflexion(end_left, start_left, m_camera.leftline);  //左拐点行
```

```
left_straight = judge_straight_cross(end_left, start_left, m_camera.leftline);  //左边直不直

right_inflexion_row=search_right_inflexion(end_right, start_right, m_camera.rightline);  //右拐点行
right_straight = judge_straight_circle(end_right, start_right, m_camera.rightline);    //右边线直不直

    bool line_ot=false;
   if(left_inflexion_row!=-1)
    {
        left_inflexion2down = judge_straight_cross(end_left, left_inflexion_row, m_camera.leftline);  //拐点往下直不直
        left_inflexion2up = judge_straight_cross(left_inflexion_row, start_left, m_camera.leftline);  //拐点往上直不直
    }
    if(right_inflexion_row!=-1)
    {
    right_inflexion2down = judge_straight_circle(end_right, right_inflexion_row, m_camera.rightline);  //拐点往上直不直
     right_inflexion2up = judge_straight_circle(right_inflexion_row, start_right, m_camera.rightline); //拐点往下直不直
    line_ot=tu_line(end_right, right_inflexion_row, m_camera.rightline);
     //cout<<line_ot<<endl;
    }
    if(CONSTRUCTION_FLAG==1 && right_straight)
    {
     CONSTRUCTION_FLAG=0;
     road_mode=RoadMode::CONSTRUCTION;
     road='c';
     error_mode=0;
     imwrite("CONSTRUCTION.jpg",m_camera.o_frame);
     CONSTRUCTION_stage=1;
     //CIRCLE_OPEN=false;
     return;
    }
    else
    {
      CONSTRUCTION_FLAG=0;
    }
```

```
    if (left_straight && left_inflexion_row == -1 && use_num_left>60)  //左边直右边乱七八糟
    {
        Lstraight_repair=true;
    }
     if(right_straight&&right_inflexion_row==-1  && use_num_right>60)     //右边直左边乱七八
糟
    {
       Rstraight_repair=true;
    }

     if( CROSS_OPEN==true )
     {
        if( left_inflexion2down && left_inflexion2up && end_left==0)
        {
              Lcross_repair=true;
        }

        if( right_inflexion2down && right_inflexion2up&& end_right==0)
        {
           Rcross_repair=true;
        }
     }
}

void Car::get_repair(Camera& m_camera)
{
   if(Lstraight_repair==true)
     {
        int temp;
        for(int i=0;i<65;i++)
        {
           m_camera.lost_right[i]=0;
           temp=m_camera.leftline[i]+44;
           if(temp>159)
              temp=159;
           m_camera.rightline[i]= temp;
        }
     }
```

```
  if( Rstraight_repair==true)
  {
     int temp;
        for(int i=0;i<65;i++)
        {
           m_camera.lost_left[i]=0;
           temp=m_camera.rightline[i]-44;
           if(temp<0)
              temp=0;
           m_camera.leftline[i]= temp;
        }
  }
  if(Lcross_repair==true)
  {
     float slope=((float)m_camera.leftline[left_inflexion_row]-(float)m_camera.leftline[0])/left_in
flexion_row;
      for(int i=0;i<=64;i++)
      {
         m_camera.lost_left[i]=0;
          m_camera.lost_right[i]=0;
          m_camera.leftline[i]=m_camera.leftline[left_inflexion_row]+round((i-left_inflexion_row)*
slope);
         m_camera.rightline[i]=m_camera.leftline[i]+52;
      }
  }
  if(Rcross_repair==true)
  {
       float slope=((float)m_camera.rightline[right_inflexion_row]-(float)m_camera.rightline[0])/ri
ght_inflexion_row;
      for(int i=0;i<=64;i++)
      {
         m_camera.lost_left[i]=0;
         m_camera.lost_right[i]=0;
          m_camera.rightline[i]=m_camera.rightline[right_inflexion_row]+round((i-right_inflexion_r
ow)*slope);
         m_camera.leftline[i]=m_camera.rightline[i]-52;
      }
  }
  if(PEDESTRIAN_OPEN==true)
```

```
  {
  int overturn=0;
  for(int i=10;i<45;i++)
   {
       if((int)m_camera.i_frame.at<uchar>(119-i, 76) == 255 && (int)m_camera.i_frame.at<uchar
>(119-i-1, 76)==0)
           overturn++;
       else if((int)m_camera.i_frame.at<uchar>(119-i, 76) == 0 && (int)m_camera.i_frame.at<uch
ar>(119-i-1, 76)==255)
           overturn++;
   }
  if(overturn>5)
  {
     //cout<<clock()<<endl;
      for(int i=0;i<=64;i++)
     {
        m_camera.lost_left[i]=0;
        m_camera.lost_right[i]=0;
        m_camera.rightline[i]=90;
        m_camera.leftline[i]=0;
     }
  }
  }
}
void Car::CONSTRUCTION_HANDLE(Camera& m_camera)  //施工区处理
{
  if(CONSTRUCTION_stage==1)  //入施工区之前正常巡线
  {
     m_camera.image_processing();  //图像处理
     m_camera.red_detection();
     m_camera.PROCESS_GetLeftBoundary();
     m_camera.PROCESS_GetRightBoundary();
     m_camera.check_repair();     //逆透视边界修正
     get_useline(m_camera);       //获取有效行
      /* m_camera.draw_boundary_frame();
     m_camera.show_b_frame();
     m_camera.show_i_frame(); */
     int num = 0;
     int x=0;
```

```
      for (int i = 65; i > 60; i--)  //晚进就是小
          {
               if ((int)m_camera.z_frame.at<uchar>(119 - i, (m_camera.leftline[i] - 21)) == 255)  //
看到锥桶
               {
                  num++;
                  break;
               }
          }
     if (num > 0)
     {
        CONSTRUCTION_stage=2;
        s_last_error=0;
              return;
     }

          get_miderror(m_camera);   //获取偏差


  }
   if(CONSTRUCTION_stage==2)
  {
     m_camera.image_Construction();  //图像处理
     int num=0;
     for(int i=49;i<112;i++)
     {
          if ((int)m_camera.i_frame.at<uchar>(119, i) == 0 )  //扫描车头黑色数 大于一定数量进
入施工区
               num++;
     }
     if(num>45)
     {

        CONSTRUCTION_stage=3;

        return;
     }
     m_camera.search_single();
```

```
    CONSTRUCTION_inerror(m_camera);

}
 if(CONSTRUCTION_stage==3)
{
    m_camera.image_Construction();  //图像处理
     int num=0;
    for(int i=0;i<60;i++)
    {
       if((int)m_camera.i_frame.at<uchar>(119-i, 79) == 255)
          num++;
    }
     if(num>5)
    {
       CONSTRUCTION_stage=4;
       s_error=-42;
       s_derror=-9;
      // road_mode=RoadMode::NORMAL;
       //s_last_error=30;
      // imwrite("outc.jpg",m_camera.r_frame);
       //cout<<"1"<<endl;
       //CONSTRUCTION_stage=0;
       return;
    }
     m_camera.red_detection();
    CONSTRUCTION_repair(m_camera);
    CONSTRUCTION_outerror(m_camera);
}
 if(CONSTRUCTION_stage==4)
{
    m_camera.image_Construction();
    int num=0;
    for(int i=43;i<112;i++)
    {
       if((int)m_camera.i_frame.at<uchar>(119, i) == 255)
          num++;
    }
    if(num>40)
    {
```

```
            road_mode=RoadMode::NORMAL;
            road='l';
            s_error=-42;
            s_derror=-6;
          //  imwrite("returnnomal.jpg",m_camera.r_frame);
            //cout<<"1"<<endl;
            CONSTRUCTION_stage=0;
            return;
        }
        m_camera.search_single();
        float error=0;
         for(int i=0;i<65;i++)
        {
       // m_camera.centerline[i]=m_camera.leftline[i]+28;
        error +=  m_camera.leftline[i] - 140;
        }
        error=error/65;
        s_error = error;   //计算偏差
        s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
        if(s_derror>9)
        s_derror=9;
        if(s_derror<-9)
        s_derror=-9;
        s_last_error=s_error;
    }

}
void Car::CONSTRUCTION_repair(Camera& m_camera)
{
    m_camera.search_single_out();
     m_camera.search_outcono();
    float slope=((float)m_camera.outcono[1]-(float)79)/(119-m_camera.outcono[0]);
    for(int i=0;i<=(64-m_camera.outcono[0]);i++)
     {
        //m_camera.leftline[i]=m_camera.leftline[left_inflexion_row]+round((i-left_inflexion_row)
*slope);
        m_camera.centerline[i]=72+round(i*slope);
     }
```

```
    for(int i=(64-m_camera.outcono[0]);i<65;i++)
    {
       m_camera.centerline[i]=132;   //135
    }


}

void Car::CONSTRUCTION_inerror(Camera& m_camera)
{
   float error=0;
   for(int i=0;i<65;i++)
   {
      if(m_camera.leftline[i]>120)
         m_camera.centerline[i]=120;
      m_camera.centerline[i]=m_camera.leftline[i]-35;

     error +=  m_camera.centerline[i] - 79;
   }
   error=error/65;
   s_error = error;   //计算偏差
   s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
   if(s_derror>9)
      s_derror=9;
   if(s_derror<-9)
      s_derror=-9;
   s_last_error=s_error;
   /* float error=0;
   for(int i=0;i<65;i++)
   {
     m_camera.centerline[i]=m_camera.Barricade[i]+22;
     error +=  m_camera.centerline[i] - 79;
   }
   error=error/65;
   s_error = error;   //计算偏差
   s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
   s_last_error=s_error; */
}
```

```
void Car::CONSTRUCTION_outerror(Camera& m_camera)
{

   float error=0;
   for(int i=0;i<65;i++)
   {
      /* if(m_camera.leftline[i]<79)
         m_camera.leftline[i]=120;
      if(m_camera.leftline[i]>125)
         m_camera.leftline[i]=125; */

     // m_camera.centerline[i]=m_camera.leftline[i]+28;
      error +=  m_camera.centerline[i] - 79;
   }
   error=error/65;
   s_error = error;   //计算偏差
   s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
   if(s_derror>9)
      s_derror=9;
   if(s_derror<-9)
      s_derror=-9;
   s_last_error=s_error;
   /* float error=0;
   for(int i=0;i<65;i++)
   {
     m_camera.centerline[i]=m_camera.Barricade[i]+22;
     error +=  m_camera.centerline[i] - 79;
   }
   error=error/65;
   s_error = error;   //计算偏差
   s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
   s_last_error=s_error; */
}
void Car::FLOODED_HANDLE(Camera& m_camera)  //泛行区处理
{

    if(FLOODED_stage==1)  //直走
   {
      m_camera.image_processing();  //图像处理
```

```
    int num=0;
    for(int i=49;i<111;i++)
    {
        if((int)m_camera.i_frame.at<uchar>(119, i) == 0)
            num++;
    }
    if(num>36)
    {
        imwrite("floodin.jpg",m_camera.r_frame);
        FLOODED_stage=2;
       // road='f';
       //  cout<<" FLOODED_stage=2"<<endl;
       // cout<<clock()<<endl;
        return;
    }
     m_camera.PROCESS_GetLeftBoundary();
    m_camera.PROCESS_GetRightBoundary();
     FLOODED_mid(m_camera);   //获取偏差
     /*  m_camera.draw_boundary_frame();
    m_camera.show_b_frame();
    m_camera.show_i_frame(); */
}
else if(FLOODED_stage==2)  //泛行区内贴左边走
{
    m_camera.image_processing();
 //   m_camera.show_i_frame();
     int num1=0;
     int num2=0;
     for (int i = 40; i < 65; i++)
    {
        if ((int)m_camera.i_frame.at<uchar>(119 - i, check_right1[i]) == 255)
            num1++;
    }
    for(int i=70;i<135;i++)
    {
        if((int)m_camera.i_frame.at<uchar>(0, i)==255)
            num2++;
    }
```

```
    /* cout<<"num1:"<<num1<<endl;
    cout<<"num2"<<num2<<endl; */
    if (num1 >13 && num2>25)
    {
       imwrite("outflood.jpg",m_camera.i_frame);
      FLOODED_stage=3;
     //  cout<<"FLOODED_stage=3"<<endl;
      //  cout<<clock()<<endl;
       return;
    }
     m_camera.search_single_right();
  /*   m_camera.show_r_frame();
    m_camera.show_i_frame();

    m_camera.draw_boundary_frame();
    m_camera.show_b_frame();   */

    FLOODED_error(m_camera);
}
else if(FLOODED_stage==3)   //固定打左
{
    m_camera.image_processing();
    int num=0;
    for(int i=49;i<112;i++)
    {
       if((int)m_camera.i_frame.at<uchar>(119, i) == 255)  //扫车头白色数
          num++;
    }
    if(num>50)
    {
       s_last_error=0;
        //imwrite("floodend.jpg",m_camera.i_frame);
       road_mode=RoadMode::NORMAL;
        road='a';
       error_mode=1;
       //cout<<"out_flooded"<<endl;
       FLOODED_stage=0;
       return;
```

```
      }
      s_error=-5;
      s_derror=0;
   }
}
void Car::FLOODED_mid(Camera& m_camera)
{
   float error=0.0;
    memset(m_camera.centerline, 79, sizeof(m_camera.centerline));
   for (int i = 0; i < 65; i++)
   {
      if(m_camera.leftline[i]>70)
         m_camera.leftline[i]=0;
      if(m_camera.rightline[i]<88)
         m_camera.rightline[i]=159;
      m_camera.centerline[i]=(m_camera.leftline[i]+m_camera.rightline[i])/2;
      error +=  m_camera.centerline[i]-79;
   }
      error = error / 65;
      s_error = error-10;   //计算偏差
      //s_slope=((double)m_camera.centerline[use_start]- (double)m_camera.centerline[use_end]
 )/num;

      s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
      s_last_error=s_error;
}
void Car::FLOODED_error(Camera& m_camera)
{
   float error=0;
   int temp=0;
   for(int i=0;i<65;i++)
   {
     if(m_camera.rightline[i]<30)
      m_camera.rightline[i]=30;
      if(m_camera.rightline[i]>80)
       m_camera.rightline[i]=80;
      temp=m_camera.rightline[i]+28;
      if(temp>159)
         temp=159;
```

```
      m_camera.centerline[i]=temp;
      error +=  m_camera.centerline[i] - 79;
    }
    error=error/65;
    s_error = error;   //计算偏差
    s_derror = s_error - s_last_error;//计算本次 D 值   //计算偏差变化率
    if(s_derror>9)
       s_derror=9;
    if(s_derror<-9)
       s_derror=-9;
    s_last_error=s_error;
}
void Car::PEDESTRIAN_HANDLE_BEGIN(Camera& m_camera) {
    if(PEDESTRIAN_stage==0)
    {
       m_camera.image_processing();
       int overturn=0;
       for(int i=10;i<45;i++)
       {
          if((int)m_camera.i_frame.at<uchar>(119-i, 76) == 255 && (int)m_camera.i_frame.at<ucha
r>(119-i-1, 76)==0)
             overturn++;
          else if((int)m_camera.i_frame.at<uchar>(119-i, 76) == 0 && (int)m_camera.i_frame.at<uc
har>(119-i-1, 76)==255)
             overturn++;
       }
       if(overturn>10)
       {
          PEDESTRIAN_stage=1;
          return;
       }
       s_error=0;
       s_derror=0;
      //cout<<clock()<<endl;
    }
    else if( PEDESTRIAN_stage==1)
    {
       m_camera.image_processing();
       int white_num=0;
```

```
        for(int i=10;i<35;i++)
        {
          if((int)m_camera.i_frame.at<uchar>(119-i, 76) == 255)
             white_num++;
        }
        //cout<<white_num<<endl;
        if(white_num>20)
        {
           s_error=-24;
           s_derror=0;
           road_mode=RoadMode::NORMAL;
           PEDESTRIAN_stage=0;
           return;
        }
       s_error=-27;
       s_derror=0;
       //cout<<clock()<<endl;
    }
}
void Car::PEDESTRIAN_HANDLE_END(Camera& m_camera)
{
       s_error=-35;
       s_derror=-3;
       road='s';
       FINAL_flag=0;
}
void Car::STATION_HANDLE(Camera& m_camera)
{
    if(STATION_stage==1)     {
       m_camera.image_processing();  //图像处理
       m_camera.red_detection();
       m_camera.PROCESS_GetLeftBoundary();
       m_camera.PROCESS_GetRightBoundary();
       m_camera.check_repair();     //逆透视边界修正
       get_useline(m_camera);       //获取有效行
        /* m_camera.draw_boundary_frame();
       m_camera.show_b_frame();
       m_camera.show_i_frame(); */
       int num = 0;
```

```
    int x=0;
     for (int i = 65; i > 60; i--)   //晚进就是小
         {
              if ((int)m_camera.z_frame.at<uchar>(119 - i, (m_camera.leftline[i] - 21)) == 255)   //看到锥桶
              {
                imwrite("intostation.jpg",m_camera.o_frame);
                num++;
                break;
              }
         }
    if (num > 0)
    {
      STATION_stage=2;
      s_last_error=0;
     //  cout<<"CONSTRUCTION_stage=2"<<endl;
      return;
    }

    //m_camera.Get_boundary();     //获取边界
     get_miderror(m_camera);   //获取偏差


  }
   if(STATION_stage==2)
  {
    m_camera.image_Construction();  //图像处理
    int num=0;
    for(int i=49;i<112;i++)
    {
        if ((int)m_camera.i_frame.at<uchar>(119, i) == 0 )   //扫描车头黑色数 大于一定数量进入施工区

              num++;
    }
    if(num>45)
    {

      STATION_stage=3;
```

```
        imwrite("black.jpg",m_camera.o_frame);
        //cout<<"CONSTRUCTION_stage=3"<<endl;
        return;
    }
    m_camera.search_single();
    CONSTRUCTION_inerror(m_camera);

}
 if(STATION_stage==3)
{
    WHITETWO=1;
    if(WHITETWO==1)
    {
        m_camera.image_Construction();  //图像处理
        int num=0;
        for(int i=80;i<96;i++)
        {
            if((int)m_camera.z_frame.at<uchar>(80, i-15) == 255)
                num++;
        }
        if(num>5)
        {
            imwrite("twoout.jpg",m_camera.o_frame);
            STATION_stage=4;
            s_error=-42;
            s_derror=-9;

             return;
        }
    }
    m_camera.red_detection();
    CONSTRUCTION_repair(m_camera);
    CONSTRUCTION_outerror(m_camera);
}
 if(STATION_stage==4)
{
    m_camera.image_Construction();
    int num=0;
    for(int i=43;i<112;i++)
```

```
    {
      if((int)m_camera.i_frame.at<uchar>(119, i) == 255)
        num++;
    }
    if(num>40)
    {

       road_mode=RoadMode::NORMAL;
       road='l';
       s_error=-42;
       s_derror=-6;
     //  imwrite("returnnomal.jpg",m_camera.r_frame);
      //cout<<"1"<<endl;
      STATION_stage=0;
      return;
    }
    m_camera.search_single();
    float error=0;
     for(int i=0;i<65;i++)
    {
   // m_camera.centerline[i]=m_camera.leftline[i]+28;
      error +=  m_camera.leftline[i] - 140;
    }
    error=error/65;
    s_error = error;  //计算偏差
    s_derror = s_error - s_last_error;//计算本次 D 值  //计算偏差变化率
    if(s_derror>9)
    s_derror=9;
    if(s_derror<-9)
    s_derror=-9;
    s_last_error=s_error;
  }
}
using namespace std;

struct Fuzzy Fz;
float temp_dif=0;
float d_temp_dif=0;
float last_servo_out=0;
```

```
float steering_KP=5;
float basic_KP=0;  //基础 P 值
float steering_KI=0;
float steering_KD=0;
float basic_KD=0;     //基础 D 值
float Bias=0;
float Integral_bias=0;
float Last_Bias=0;



float abc(float a)
{
  if(a>0)
  {
    return a;
  }
  else
    return a*(-1);
}

const float fuzzyRuleKp[7][7] =
{

    {NL1,NB1,NM1,NB1,NM1,NS1,ZO1},
    {NB1,NM1,NS1,NM1,ZO1,ZO1,NS1},
    {NM1,NS1,NS1,NS1,ZO1,ZO1,NS1},
    {ZO1,ZO1,ZO1,ZO1,ZO1,ZO1,ZO1},
    {PS1,ZO1,ZO1,PS1,PS1,PS1,PM1},
    {PS1,ZO1,ZO1,PM1,PS1,PM1,PB1},
    {ZO1,PS1,PM1,PB1,PM1,PB1,PL1},


};//dKp 模糊规则表

const float fuzzyRuleKi[7][7]=
{
    {NB2,NB2,NM2,NM2,NS2,ZO2,ZO2},
    {NB2,NB2,NM2,NS2,NS2,ZO2,ZO2},
```

```
    {NB2,NM2,NS2,NS2,ZO2,PS2,PS2},
    {NM2,NM2,NS2,ZO2,PS2,PM2,PM2},
    {NM2,NS2,ZO2,PS2,PS2,PM2,PB2},
    {ZO2,ZO2,PS2,PS2,PM2,PB2,PB2},
    {ZO2,ZO2,PS2,PM2,PM2,PB2,PB2},


};//dKi 模糊规则表

const float fuzzyRuleKd[7][7]={

   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},
   {NB3,NM3,NS3,ZO3,PS3,PM3,PB3},

};//dKd 模糊规则表



float min(float x,float y)
{
  float z;
  if(x<y) z=x;
  else z=y;
  return (z);
}

/**********输出最大值*****************/
float max(float x,float y)
{
  float z;
  if(x>y) z=x;
  else z=y;
  return (z);
}
```

```
void Fuzzy_init(void)//模糊控制初始化函数
{
   Fz.Pos=0;//定义论域
   Fz.Dpos=0;

   Fz.Pos_Rule_Value[0]=Pos_ZO;//定义偏差模糊值
   Fz.Pos_Rule_Value[1]=Pos_ZO;

   Fz.Pos_U_Value[0]=1;//定义偏差隶属度
   Fz.Pos_U_Value[1]=0;

   Fz.Dpos_Rule_Value[0]=Dpos_ZO;//定义偏差变化率模糊值
   Fz.Dpos_Rule_Value[1]=Dpos_ZO;

   Fz.Dpos_U_Value[0]=1;//定义偏差变化率隶属度
   Fz.Dpos_U_Value[1]=0;

   Fz.Servo_out = 0;//舵机输出值
   Fz.Speed_out = 0;//电机输出值
}

void Fuzzy_Pos_input(float s_error)    //偏差量化函数
{

 temp_dif=s_error;
  if(temp_dif>=42)
 {
     temp_dif =42;
 }
  if(temp_dif<=-42)
  {
     temp_dif = -42;
  }
   Fz.Pos =  (float)temp_dif/14+3 ;   // Pos[0 6]
}
void Fuzzy_Dpos_input(float s_derror)  //变化值偏差量化
{

   //变化量偏差量化为[0,6]
```

```
  d_temp_dif =s_derror;
 if(d_temp_dif>9)
  {
    d_temp_dif = 9;//确定最大 EC=6
  }
  if(d_temp_dif<-9)
  {
    d_temp_dif =-9;//确定最小 EC=0
  }
  Fz.Dpos = d_temp_dif/3+3;


}
void Pos_fuzzy(void)
{
   int i=0,j=0;
   float f1,f2;
   for(j=0;j<2;j++)
   {
    Fz.Pos_Rule_Value[j]=0;
   }//先将之前运算的隶属度变为 0

   if(Fz.Pos < Pos_NB_R)      //论域 1   01 之间 模糊值为 1 隶属度为
   {
    Fz.Pos_Rule_Value[i] = Pos_NB;  //确定模糊值

    f2 = (Pos_NB_R - Fz.Pos);
    f1 = f2;

    Fz.Pos_U_Value[i] = min(f1,f2);  //计算隶属度

    i++;
   }

   if((Fz.Pos>=Pos_NM_L)&&(Fz.Pos<Pos_NM_R))//论域 2
   {
    Fz.Pos_Rule_Value[i]=Pos_NM;

    f1 = (Fz.Pos - Pos_NM_L);
```

```
  f2 = (Pos_NM_R - Fz.Pos);

  Fz.Pos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Pos>=Pos_NS_L)&&(Fz.Pos<Pos_NS_R))//论域 3
{
  Fz.Pos_Rule_Value[i]=Pos_NS;

  f1 = (Fz.Pos - Pos_NS_L);
  f2 = (Pos_NS_R - Fz.Pos);

  Fz.Pos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Pos>=Pos_ZO_L)&&(Fz.Pos<Pos_ZO_R))//论域 4
{
  Fz.Pos_Rule_Value[i]=Pos_ZO ;

  f1 = (Fz.Pos - Pos_ZO_L);
  f2 = (Pos_ZO_R - Fz.Pos);

  Fz.Pos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Pos>=Pos_PS_L)&&(Fz.Pos<Pos_PS_R))//论域 5
{
  Fz.Pos_Rule_Value[i]=Pos_PS;

  f1 = (Fz.Pos - Pos_PS_L);
  f2 = (Pos_PS_R - Fz.Pos);

  Fz.Pos_U_Value[i]=min(f1,f2);
```

```
    i++;
  }

  if((Fz.Pos>=Pos_PM_L)&&(Fz.Pos<Pos_PM_R))//论域 6
  {
    Fz.Pos_Rule_Value[i]=Pos_PM;

    f1 = (Fz.Pos - Pos_PM_L);
    f2 = (Pos_PM_R - Fz.Pos);

    Fz.Pos_U_Value[i]=min(f1,f2);

    i++;
  }

  if(Fz.Pos >= Pos_PB_L)          //论域 7
  {
    Fz.Pos_Rule_Value[i]=Pos_PB;

    f1 = (Fz.Pos - Pos_PB_L);
    f2 = f1;

    Fz.Pos_U_Value[i]=min(f1,f2);

    i++;
  }

  if(i == 1)
  {
     Fz.Pos_U_Value[1]=0;
  }
}

void Dpos_fuzzy(void)
{
  int i=0,j=0;
  float f1=0,f2=0;
  for(j=0;j<2;j++)
```

```
{
  Fz.Dpos_Rule_Value[j]=0;
}

if(Fz.Dpos < Dpos_NB_R)       //论域 1
{
  Fz.Dpos_Rule_Value[i] = Dpos_NB;  //?????

  f2 = (Dpos_NB_R - Fz.Dpos);
  f1 = f2;

  Fz.Dpos_U_Value[i] = min(f1,f2);

  i++;
}

if((Fz.Dpos>=Dpos_NM_L)&&(Fz.Dpos<Dpos_NM_R))//论域 2
{
  Fz.Dpos_Rule_Value[i]=Dpos_NM ;

  f1 = (Fz.Dpos - Dpos_NM_L);
  f2 = (Dpos_NM_R - Fz.Dpos);

  Fz.Dpos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Dpos>=Dpos_NS_L)&&(Fz.Dpos<Dpos_NS_R))//论域 3
{
  Fz.Dpos_Rule_Value[i]=Dpos_NS ;

  f1 = (Fz.Dpos - Dpos_NS_L);
  f2 = (Dpos_NS_R - Fz.Dpos);

  Fz.Dpos_U_Value[i]=min(f1,f2);

  i++;
}
```

```
if((Fz.Dpos>=Dpos_ZO_L)&&(Fz.Dpos<Dpos_ZO_R))//论域 4
{
  Fz.Dpos_Rule_Value[i]=Dpos_ZO;

  f1 = (Fz.Dpos - Dpos_ZO_L);
  f2 = (Dpos_ZO_R - Fz.Dpos);

  Fz.Dpos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Dpos>=Dpos_PS_L)&&(Fz.Dpos<Dpos_PS_R))//论域 5
{
  Fz.Dpos_Rule_Value[i]=Dpos_PS ;

  f1 = (Fz.Dpos - Dpos_PS_L);
  f2 = (Dpos_PS_R - Fz.Dpos);

  Fz.Dpos_U_Value[i]=min(f1,f2);

  i++;
}

if((Fz.Dpos>=Dpos_PM_L)&&(Fz.Dpos<Dpos_PM_R))//论域 6
{
  Fz.Dpos_Rule_Value[i]=Dpos_PM ;

  f1 = (Fz.Dpos - Dpos_PM_L);
  f2 = (Dpos_PM_R - Fz.Dpos);

  Fz.Dpos_U_Value[i]=min(f1,f2);

  i++;
}

if(Fz.Dpos >= Dpos_PB_L)          //论域 7
{
```

```
    Fz.Dpos_Rule_Value[i]=Dpos_PB ;

    f1 = (Fz.Dpos - Dpos_PB_L);
    f2 = f1;

    Fz.Dpos_U_Value[i]=min(f1,f2);

    i++;
   }

   if(i==1)
   {
     Fz.Dpos_U_Value[1]=0;
   }
}

float   Fuzzy_Kp(Car& m_car)
{
    uint8_t i,j;
    Fuzzy_Pos_input(m_car.s_error);  //对检测到的偏差量化
    Fuzzy_Dpos_input(m_car.s_derror);  //对计算出的偏差变化率量化

    Pos_fuzzy();//E 模糊化，得模糊值与隶属度
    Dpos_fuzzy();//EC 模糊化，得模糊值与隶属度

    Fz.P_Rule[0]=fuzzyRuleKp[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[0]];
    Fz.P_Rule[1]=fuzzyRuleKp[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[0]];
    Fz.P_Rule[2]=fuzzyRuleKp[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[1]];
    Fz.P_Rule[3]=fuzzyRuleKp[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[1]];

    Fz.P_U[0] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[0]);
    Fz.P_U[1] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[1]);
    Fz.P_U[2] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[0]);
    Fz.P_U[3] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[1]);

    for(i=0;i<3;i++)
    {
     for(j=i+1;j<4;j++)
     {
```

```
      if(Fz.P_Rule[i] == Fz.P_Rule[j])
      {
        if(Fz.P_U[i] > Fz.P_U[j])
        {
          Fz.P_U[j]=0;
        }
        else
        {
          Fz.P_U[i]=0;
        }
      }
    }
   }
   Fz.P_out_u=0;
    Fz.P_out=0;
   /*最大最小法解模糊*/
   /*加权平均法解模糊？ */
    for(i=0;i<4;i++)
    {
      Fz.P_out   += (float)Fz.P_Rule[i]*Fz.P_U[i];
      Fz.P_out_u += (float)(Fz.P_U[i]);
    }

    Fz.P_out =Fz.P_out / Fz.P_out_u;

      return Fz.P_out;
}
float   Fuzzy_Ki(Car& m_car)
{
   uint8_t i,j;
   Fuzzy_Pos_input(m_car.s_error);  //对检测到的偏差量化
   Fuzzy_Dpos_input(m_car.s_derror);  //对计算出的偏差变化率量化

   Pos_fuzzy();//E 模糊化，得模糊值与隶属度
   Dpos_fuzzy();//EC 模糊化，得模糊值与隶属度

   Fz.I_Rule[0]=fuzzyRuleKi[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[0]];
   Fz.I_Rule[1]=fuzzyRuleKi[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[0]];
   Fz.I_Rule[2]=fuzzyRuleKi[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[1]];
```

```
Fz.I_Rule[3]=fuzzyRuleKi[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[1]];



Fz.I_U[0] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[0]);
Fz.I_U[1] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[1]);
Fz.I_U[2] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[0]);
Fz.I_U[3] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[1]);

for(i=0;i<3;i++)
{
 for(j=i+1;j<4;j++)
 {
  if(Fz.I_Rule[i] == Fz.I_Rule[j])
  {
    if(Fz.I_U[i] > Fz.I_U[j])
    {
      Fz.I_U[j]=0;
    }
    else
    {
      Fz.I_U[i]=0;
    }
  }

 }
}
Fz.I_out_u=0;
 Fz.I_out=0;
/*最大最小法解模糊*/
/*加权平均法解模糊？ */
 for(i=0;i<4;i++)
 {
  Fz.I_out   += (float)Fz.I_Rule[i]*Fz.I_U[i];
  Fz.I_out_u += (float)(Fz.I_U[i]);
 }

 Fz.I_out =Fz.I_out / Fz.I_out_u;
```

```
    return Fz.I_out;
}
float  Fuzzy_Kd(Car& m_car)
{
  uint8_t i,j;
  Fuzzy_Pos_input(m_car.s_error); //对检测到的偏差量化
  Fuzzy_Dpos_input(m_car.s_derror); //对计算出的偏差变化率量化

  Pos_fuzzy();//E 模糊化，得模糊值与隶属度
  Dpos_fuzzy();//EC 模糊化，得模糊值与隶属度

  Fz.D_Rule[0]=fuzzyRuleKd[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[0]];
  Fz.D_Rule[1]=fuzzyRuleKd[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[0]];
  Fz.D_Rule[2]=fuzzyRuleKd[Fz.Pos_Rule_Value[0]][Fz.Dpos_Rule_Value[1]];
  Fz.D_Rule[3]=fuzzyRuleKd[Fz.Pos_Rule_Value[1]][Fz.Dpos_Rule_Value[1]];

  Fz.D_U[0] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[0]);
  Fz.D_U[1] = min(Fz.Dpos_U_Value[0],Fz.Pos_U_Value[1]);
  Fz.D_U[2] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[0]);
  Fz.D_U[3] = min(Fz.Dpos_U_Value[1],Fz.Pos_U_Value[1]);

  for(i=0;i<3;i++)
  {
   for(j=i+1;j<4;j++)
   {
    if(Fz.D_Rule[i] == Fz.D_Rule[j])
    {
      if(Fz.D_U[i] > Fz.D_U[j])
      {
        Fz.D_U[j]=0;
      }
      else
      {
        Fz.D_U[i]=0;
      }
    }
   }
  }
  Fz.D_out_u=0;
```

```
    Fz.D_out=0;
    for(i=0;i<4;i++)
    {
     Fz.D_out   += (float)Fz.D_Rule[i]*Fz.D_U[i];
     Fz.D_out_u += (float)(Fz.D_U[i]);
    }

    Fz.D_out =Fz.D_out / Fz.D_out_u;

     return Fz.D_out;
}

void PID_steering_out(float s_error,float s_derror)
{

  if(s_derror>12)
  {
     s_derror=12;
  }
  if(s_derror<-12)
   {
     s_derror=-12;
   }
   last_servo_out=steering_KP*s_error+steering_KD*s_derror;
   if(last_servo_out>165)          //输出限幅
     last_servo_out=165;
   if(last_servo_out<-165)
     last_servo_out=-165;
}
 void FuzzyPID(Car& m_car)
 {
   //  cout<<"***running_parameter***"<<endl;
    steering_KP=Fuzzy_Kp(m_car);
    steering_KD=Fuzzy_Kd(m_car);
   // steering_KD=5;
    PID_steering_out(m_car.s_error,m_car.s_derror);
 }
using namespace std;
using namespace cv;
```

```
const int white = 255;
const int black = 0;

unsigned char noise_flag = 0;
unsigned char tmp_row = 0; //行计数器
unsigned char tmp_col = 0;  //列计数器
unsigned char tmp_boundary_left = 80; //左边界暂存值
unsigned char tmp_boundary_right = 80; //右边界暂存值

extern uint8_t lost_num_left;
extern uint8_t lost_num_right;



int distance(unsigned char a, unsigned char b)
{
   if (a > b)
      return a - b;
   else
      return b - a;
}
uint8_t check_left[120] = {
   49, 48, 47, 46, 46, 45, 44, 44, 43, 42, 41, 41, 40, 39, 39, 38, 37, 36, 36, 35, 34, 34, 33, 32, 31, 31,
 30, 29, 29, 28,
   27, 26, 26, 25, 24, 24, 23, 22, 21, 21, 20, 19, 19, 18, 17, 16, 16, 15, 14, 14, 13, 12, 11, 11, 10,  9,
 9,  8,  7,  6,
    6,  5,  4,  4,  3,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
    1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1
};  //逆透视边界

uint8_t check_right[120] = {
   111, 112, 112, 113, 114, 114, 115, 116, 117, 117, 118, 119, 119, 120, 121, 122, 122, 123, 124, 1
24, 125, 126, 127, 127, 128, 129, 129, 130, 131, 131,
   132, 133, 134, 134, 135, 136, 136, 137, 138, 139, 139, 140, 141, 141, 142, 143, 143, 144, 145, 1
46, 146, 147, 148, 148, 149, 150, 151, 151, 152, 153,
   153, 154, 155, 156, 156, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158 ,1
58 ,158, 158, 158, 158, 158, 158, 158, 158, 158, 158,
   158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 1
```

```
58, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158
};  //逆透视边界

Camera::Camera(VideoCapture video)
{
   m_video = video;

   m_video.set(CAP_PROP_FOURCC, VideoWriter::fourcc('M', 'J', 'P', 'G'));


   m_video.set(3, 640);  //设置摄像头的分辨率
   m_video.set(4, 480);

   weight = 160;    //图像宽度
   height = 120;    //图像高度
   rotation = get_perspective_mat();
   outcono[0]=0;
   outcono[1]=0;
}

void Camera::read_frame(void)
{
   m_video >> o_frame;

}
void Camera::create_video(const string& file)
{
   //read_frame();

   bool isColor = (r_frame.type() == CV_8UC3);
   int codec = VideoWriter::fourcc('M', 'J', 'P', 'G');
   double fps = 30.0;
   string filename = file;
   m_avi.open(filename, codec, fps,r_frame.size(), isColor);
   if (!m_avi.isOpened())
   {
      cout << "创建视频流失败" << endl;
   }
}
```

```
void Camera::save_avi()
{
   m_avi.write(r_frame);
}
void Camera::show_o_frame(void)
{

   // imshow("orignal", o_frame);
   // waitKey(1);
}

void Camera::show_r_frame(void)
{

   imshow("resize", r_frame);
   waitKey(1);

}
void Camera::show_b_frame(void)
{

   imshow("boundary", b_frame);
   waitKey(1);
}
void Camera::show_i_frame(void)
{
   imshow("Inverse perspective", i_frame);
   waitKey(1);
}
void Camera::save_frame()
{

}
Mat Camera::get_perspective_mat()
{
   float offset_x = 80;
   float offset_y = 124;
   Mat rotation;
   Point2f src_point[4];
```

```
    Point2f dst_point[4];
    src_point[0] = Point2f(49, 74);
    src_point[1] = Point2f(111, 74);
    src_point[2] = Point2f(33, 102);
    src_point[3] = Point2f(127, 102);

    dst_point[0] = Point2f(-21 + offset_x, -30 + offset_y);
    dst_point[1] = Point2f(21 + offset_x, -30 + offset_y);
    dst_point[2] = Point2f(-21 + offset_x, -10 + offset_y);   //29.5    54.5      20.5
    dst_point[3] = Point2f(21 + offset_x, -10 + offset_y);

    rotation = getPerspectiveTransform(src_point, dst_point);
    return rotation;
}
void Camera::image_processing()
{
    Mat temp;
    resize(o_frame, r_frame, Size(160, 120), 0, 0, INTER_AREA); //裁剪尺寸
    warpPerspective(r_frame, temp, rotation, r_frame.size(), cv::INTER_LINEAR);  //逆透视变换
    r_frame = temp;
    cvtColor(r_frame, i_frame, COLOR_BGR2GRAY);  //灰度化
    threshold(i_frame, i_frame, 150, 255, THRESH_BINARY); //二值化
}
void Camera::image_Construction()
{
    Mat temp;
    resize(o_frame, r_frame, Size(160, 120), 0, 0, INTER_AREA); //裁剪尺寸
    warpPerspective(r_frame, temp, rotation, r_frame.size(), cv::INTER_LINEAR);  //逆透视变换
    r_frame = temp;
    cvtColor(r_frame, i_frame, COLOR_BGR2GRAY);  //灰度化
    threshold(i_frame, i_frame, 170, 255, THRESH_BINARY); //二值化
}
void Camera::red_detection()
{
    Mat imgHSV;
    Mat temp;
    Mat temp1=r_frame;
    GaussianBlur(temp1, temp1, Size(7, 7), 0, 0);
    vector<Mat> hsvSplit;  //创建向量容器，存放 HSV 的三通道数据
```

```
  cvtColor(temp1, imgHSV, COLOR_BGR2HSV); //Convert the captured frame from BGR to HSV
  split(imgHSV, hsvSplit);          //分类原图像的 HSV 三通道
  equalizeHist(hsvSplit[2], hsvSplit[2]);   //对 HSV 的亮度通道进行直方图均衡
  merge(hsvSplit, imgHSV);              //合并三种通道
  Mat ce1;
  Mat ce2;
  inRange(imgHSV, Scalar(156, 43, 46), Scalar(180, 255, 255), ce1); //红色
  inRange(imgHSV, Scalar(0, 43, 46), Scalar(3, 255, 255), ce2); //红色
  add(ce1, ce2, temp, Mat());
  //  imshow("0-10", ce1);
   // imshow("156-180", ce2);
  //  imshow("add 后", z_frame);
  // waitKey(1);
  Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));
  dilate(temp, z_frame, element);
  /*   imshow("add 后", z_frame);
  waitKey(1);   */

}

void Camera::draw_boundary_frame(void)
{
  b_frame = Mat::zeros(50, 160, CV_8UC3);
  b_frame = Scalar(0, 0, 0);
  for (int i = 0; i < 50; i++)
  {

    if(lost_left[i]==0)
    {
    b_frame.at<Vec3b>(49 - i, leftline[i])[0] = 0; // blue
    b_frame.at<Vec3b>(49 - i, leftline[i])[1] = 255; // green
    b_frame.at<Vec3b>(49 - i, leftline[i])[2] = 0; // red
    }
    else
    {
    b_frame.at<Vec3b>(49 - i, leftline[i])[0] = 200; // blue
    b_frame.at<Vec3b>(49 - i, leftline[i])[1] = 125; // green
    b_frame.at<Vec3b>(49 - i, leftline[i])[2] = 0; // red
    }
```

```
    if(lost_right[i]==0)
    {
    b_frame.at<Vec3b>(49 - i, rightline[i])[0] = 0; // blue
    b_frame.at<Vec3b>(49 - i, rightline[i])[1] = 0; // green
    b_frame.at<Vec3b>(49 - i, rightline[i])[2] = 255; // red
    }
    else
    {
    b_frame.at<Vec3b>(49 - i, rightline[i])[0] = 200; // blue
    b_frame.at<Vec3b>(49 - i, rightline[i])[1] = 0; // green
    b_frame.at<Vec3b>(49 - i, rightline[i])[2] = 125; // red
    }
    if(lost_left[i]==0 || lost_right[i]==0)
    {
    b_frame.at<Vec3b>(49 - i, centerline[i])[0] = 255; // blue
    b_frame.at<Vec3b>(49 - i, centerline[i])[1] = 0; // green
    b_frame.at<Vec3b>(49 - i, centerline[i])[2] = 0; // red
    }
  }

}

/*******************搜左边线***********************************************/
void Camera::PROCESS_GetLeft_White(void)   //如果近行上一点是白色，说明近行的边界在左侧
{

  while ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == white)
  {
    if (tmp_boundary_left ==1)
    {

      break;
      //向左找到第一个黑点
    }
    else tmp_boundary_left--;
  }
  noise_flag = 0;
  if (tmp_boundary_left > 3)
```

```
  {
    if (((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left - 1) != white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left - 2) != white) &&
       ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left - 3) != white))     //如果黑点左边三点仍然是黑点，则为边界
    {
      tmp_boundary_left++;                                          //定位到白点
    }
    else
    {                                                  //噪声则重来
      tmp_boundary_left -= 3;
      noise_flag = 1;

    }
  }
}

void Camera::PROCESS_GetLeft_Black(void)  //如果近边上一点为黑色，说明边界在右侧，则向右搜索
{
  while ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) != white)
  {
    if (tmp_boundary_left ==158)
    {

      break;
      //搜到第 158 仍然为黑，则退出
    }
    else tmp_boundary_left++;
  }
  noise_flag = 0;
  if (tmp_boundary_left < 156)
  {
    if (((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left + 1) == white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left + 2) == white) &&
       ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left + 3) == white)) //白点右则三点仍为白色，符合
    {
      noise_flag = 0;
```

```
    }
    else
    {
      tmp_boundary_left += 3;
      noise_flag = 1;                                    //噪声则重来
    }
  }

}

void Camera::PROCESS_GetLeftBoundary(void)  //从最后一行开始往上，逐行搜索左边界，每一行搜索在前一行边界点的基础上进行。
{
  tmp_boundary_left = 80;   //左边线搜线暂存值居中
  memset(lost_left, 0, sizeof(lost_left)); //丢线位图清零
   lost_num_left = 0;
  for (tmp_row = 0; tmp_row <80; tmp_row++)
  {
    noise_flag = 1;
   // unsigned last_left = tmp_boundary_left;
    if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == white)
    {
      while (noise_flag == 1) PROCESS_GetLeft_White();          //近边上一点为白色
    }
    else if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == black)
    {
      while (noise_flag == 1) PROCESS_GetLeft_Black();          //近边上一点为黑色
    }

      leftline[tmp_row] = tmp_boundary_left;

  }

  tmp_boundary_left = 80;                                    //Î³ÏÂÒ»ÐÈÑÏß×ö×¼±
¸
}

/**********************搜右边线*****************************/
void Camera::PROCESS_GetRight_White(void)  //如果近行上一点是白色，说明近行的边界在
```

```
右侧
{

  while ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) == white)
  {
    if (tmp_boundary_right == 158)
    {

      break;//近边上一点为白色，则向右搜索，找到黑点为止
    }
    else tmp_boundary_right++;
  }
  noise_flag = 0;
  if (tmp_boundary_right < 156)
  {
    if (((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right + 1) != white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right + 2) != white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right + 3) != white))    //黑点右侧
三点仍为黑点
    {
      tmp_boundary_right--; //定位到白点
    }
    else
    {
      tmp_boundary_right += 3;
      noise_flag = 1;
    }
  }
}
void Camera::PROCESS_GetRight_Black(void)  //如果近行上一点是黑色，说明近行的边界在左
侧
{
  while ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) != white)
  {
    if (tmp_boundary_right == 1)
    {

      break;
    }
```

```
    else tmp_boundary_right--;
  }
  noise_flag = 0;
  if (tmp_boundary_right > 3)
  {
    if (((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right - 1) == white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right - 2) == white) &&
      ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right - 3) == white))    //
    {
      noise_flag = 0;
    }
    else {
      tmp_boundary_right -= 3;
      noise_flag = 1;
    }
  }
}
void Camera::PROCESS_GetRightBoundary(void)
{
  tmp_boundary_right = 80;    //搜线暂存值居中
  memset(lost_right, 0, sizeof(lost_right));  //丢线位图清零
  lost_num_right = 0;

  for (tmp_row = 0; tmp_row <80; tmp_row++)
  {
    noise_flag = 1;
    //unsigned last_right = tmp_boundary_right; // 存储上一行边界的值
    if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) == white)
    {
      while (noise_flag == 1) PROCESS_GetRight_White();
    }
    else if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) == black)
    {
      while (noise_flag == 1) PROCESS_GetRight_Black();
    }


     rightline[tmp_row] = tmp_boundary_right;
```

```
    }
    tmp_boundary_right = 80;
}
void Camera::Get_boundary(void)
{
    uint8_t i;
    memset(lost_left, 1, sizeof(lost_left));
    memset(lost_right, 1, sizeof(lost_right));
    memset(leftline,0,sizeof(leftline));
    memset(rightline,159,sizeof(rightline));
    centerline[0] = Get_one_boundary_after_start(119, 79);//第一行的起始行为 58 起始点为 40
    uint8_t temp = centerline[0];
  for (i = 1; i < 80; i++)
    {
        centerline[i] = Get_one_boundary_after_start(119-i, temp);
        temp = centerline[i];
    }
}
uint8_t Camera::Get_one_boundary_after_start(uint8_t now_row, uint8_t last_middle_point)
{
   //向左搜线 起始点为上次的中点 但当起始点为黑点时，说明此时搜线起点不在跑道上，此时起始点仍然选为 40 列处
    uint8_t j;
    uint8_t mid_point;
    //  flag1=2;
    if ((int)i_frame.at<uchar>(now_row, last_middle_point) == white)//如果上一次的起始点在跑道上
    {
        for (j = last_middle_point; j > 3; j--)//从中间开始向左搜黑点
        {
            /* if (j <= check_left[119 - now_row])
            {
                leftline[119 - now_row] = check_left[119 - now_row];
                lost_left[119 - now_row] = 0;
                break;
            } */
            if ((int)i_frame.at<uchar>(now_row, j) == white && (int)i_frame.at<uchar>(now_row, j - 1) == black && (int)i_frame.at<uchar>(now_row, j - 2) == black && (int)i_frame.at<uchar>(now_row, j - 3) == black)//搜到的黑色点往左连续 3 个点都为黑色时才判定为左边线
```

```
        {

            leftline[119 - now_row] = j;//得到本次行的黑色赛道线点
            lost_left[119 - now_row] = 0;     //表明本次行成功搜到边线
            break;
        }

    }
  }
  else
  {
    leftline[119 - now_row] = 0;
  }
  //向右搜线 起始点为上次的中点
  if ((int)i_frame.at<uchar>(now_row, last_middle_point) == white)
  {
    for (j = last_middle_point; j < 156; j++)//从中间开始搜黑点
    {

        if ((int)i_frame.at<uchar>(now_row, j) == white && (int)i_frame.at<uchar>(now_row, j + 1)
 == black && (int)i_frame.at<uchar>(now_row, j + 2) == black && (int)i_frame.at<uchar>(now_ro
w, j + 3) == black)
        {

            rightline[119 - now_row] = j;
            lost_right[119 - now_row] = 0;                //记录成功搜到边线的行
            break;
        }

    }
  }
  else
  {
    rightline[119 - now_row] = 159;

  }
  mid_point = (leftline[119 - now_row] + rightline[119 - now_row]) / 2;
  return mid_point;
}
```

```
void Camera::check_repair(void)
{
  for (int i = 0; i < 50; i++)
   {
      if (leftline[i] <= check_left[i]+1)
      {
         leftline[i] = 0;
         lost_left[i] = 1;
      }
      if (rightline[i] >= check_right[i]-1)
      {
         rightline[i] = 159;
         lost_right[i] = 1;
      }
      if (leftline[i] > rightline[i])  //错误 1：左边界大于右边界 边界修正
      {
         leftline[i] = 0;
         rightline[i] = 159;
         lost_left[i] = 1;
         lost_right[i] = 1;
      }
   }
}
void Camera::search_single()  //搜索单边界
{

    tmp_boundary_left = 80;   //左边线搜线暂存值居中
   memset(lost_left, 0, sizeof(lost_left)); //丢线位图清零
    lost_num_left = 0;
   for (tmp_row = 0; tmp_row <80; tmp_row++)
   {
      noise_flag = 1;
     // unsigned last_left = tmp_boundary_left;
      if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == white)
      {
         while (noise_flag == 1) PROCESS_GetLeft_White();                    //近边上一点为白色
      }
      else if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == black)
      {
```

```
        while (noise_flag == 1) PROCESS_GetLeft_Black();                    //近边上一点为黑色
      }
        if(tmp_boundary_left>155)
        {
        leftline[tmp_row] = check_right[tmp_row];
        lost_left[tmp_row]=1;
        tmp_boundary_left=80;
        }
        else
          leftline[tmp_row] = tmp_boundary_left;
   }

   tmp_boundary_left = 80;
}
void Camera::search_single_out()  //搜索单边界
{
    tmp_boundary_left = 80;   //左边线搜线暂存值居中
   memset(lost_left, 0, sizeof(lost_left)); //丢线位图清零
    lost_num_left = 0;
   for (tmp_row = 55; tmp_row <= 119; tmp_row++)
   {
      noise_flag = 1;
      if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == white)
      {
        while (noise_flag == 1) PROCESS_GetLeft_White();                    //近边上一点为白色
      }
      else if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_left) == black)
      {
        while (noise_flag == 1) PROCESS_GetLeft_Black();                    //近边上一点为黑色
      }
        if(tmp_boundary_left>155 || tmp_boundary_left<79)
        {
        lost_num_left++;
        leftline[tmp_row-55] = 159;
        lost_left[tmp_row-55]=1;
        tmp_boundary_left=80;
        }
        else
          leftline[tmp_row-55] = tmp_boundary_left;
```

```
    }
    tmp_boundary_left = 80;
}
void Camera::search_outcono()
{
     outcono[0]=50;
    outcono[1]=159;
    for(int i=0;i<26;i++)
    {
        if((int)z_frame.at<uchar>(i, leftline[49-i]-4) == 255 )
        {
            outcono[0]=i;
            outcono[1]=leftline[i]-4;
     break;
        }
    }
}
void Camera::search_single_right()  //搜索单边界
{
     tmp_boundary_right = 80;   //左边线搜线暂存值居中
    memset(lost_right, 0, sizeof(lost_right)); //丢线位图清零
      lost_num_right = 0;
    for (tmp_row = 0; tmp_row <80; tmp_row++)
    {
        noise_flag = 1;
        if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) == white)
        {
            while (noise_flag == 1) PROCESS_GetRight_White();                  //近边上一点为白色
        }
        else if ((int)i_frame.at<uchar>(119 - tmp_row, tmp_boundary_right) == black)
        {
            while (noise_flag == 1) PROCESS_GetRight_Black();                  //近边上一点为黑色
        }
            if(tmp_boundary_right<3)
            {
            rightline[tmp_row] = tmp_boundary_right;
            tmp_boundary_right=80;
            }
            else
```

```
            rightline[tmp_row] = tmp_boundary_right;
    }
    tmp_boundary_left = 80;
}
```