第十七届全国大学生智能汽车竞赛

技 术 报 告

学　　校：新余学院

队伍名称：大大怪队

参赛队员：刘长清

余雄

吁凌洋

带队教师：陶秋香

傅思勇

目录

摘要

该智能车设计以百度的 edgeboard 为图像识别单元，通过 opencv 数字摄像头传感器采集赛道信息，将采集到的图像进行处理；通过 paddlepaddle 平台模型训练部署神经网络识别图像；另外 tc264 通过编码器监测速度，同时采用 PID 控制算法，控制智能车的舵机转向以及智能车的行驶速度闭环控制，使得智能车能够快速、稳定的行驶，从而实现自主寻迹以及控制的目的。

关键词： 智能车 PID 算法 edgeboard tc264 摄像头

第一章

引言

1.1 背景意义

全国大学生“恩智浦”杯智能汽车竞赛是以“立足培养、重在参与、鼓励探索、追求卓越”为宗旨，鼓励创新的一项科技竞赛活动。竞赛要求在规定的汽车模型平台上，使用指定公司的微控制器作为核心控制模块，通过增加道路检测传感器、电机驱动电路以及编写相应控制程序，制作完成一个能够自主识别道路以及相应任务的模型汽车。参赛队员的目标是模型汽车需要按照规则以最短时间完成赛道上相应元素的任务。

1.2 报告内容及安排

在此次比赛中，本组使用竞赛统一提供的竞赛车 I 车模，采用百度的 edgeboard 图像识别处理器，自主设计系统的控制方案。其中融合了摄像头的图像采集与处理、电机速度闭环控制以及识别，最终完成了一套能够自主识别行驶路线、进行图像识别的完整系统。

在这份报告中，我们主要阐述了整车系统的整体方案、机械结构、硬件设计、以及软件算法等方面，详细说明了我们设计过程中的思想与创新。在准备比赛的过程中，我们翻阅了大量的专业资料，反复的对各种算法模型的参数进行调试，每个队员都为本次智能车竞赛付出了艰苦的汗水

第二章

智能车方案设计

本章主要对模型车的设计制作的主要思路以及实现的技术方案概要说明，在后面的章节中将系统的分为机械结构、硬件电路、控制算法、图像识别算法等对智能车系统进行深入的介绍分析。

2.1 系统总体设计方案

根据竞赛规则相关规定，我们在智能车制作过程中，主要做了如下设计：

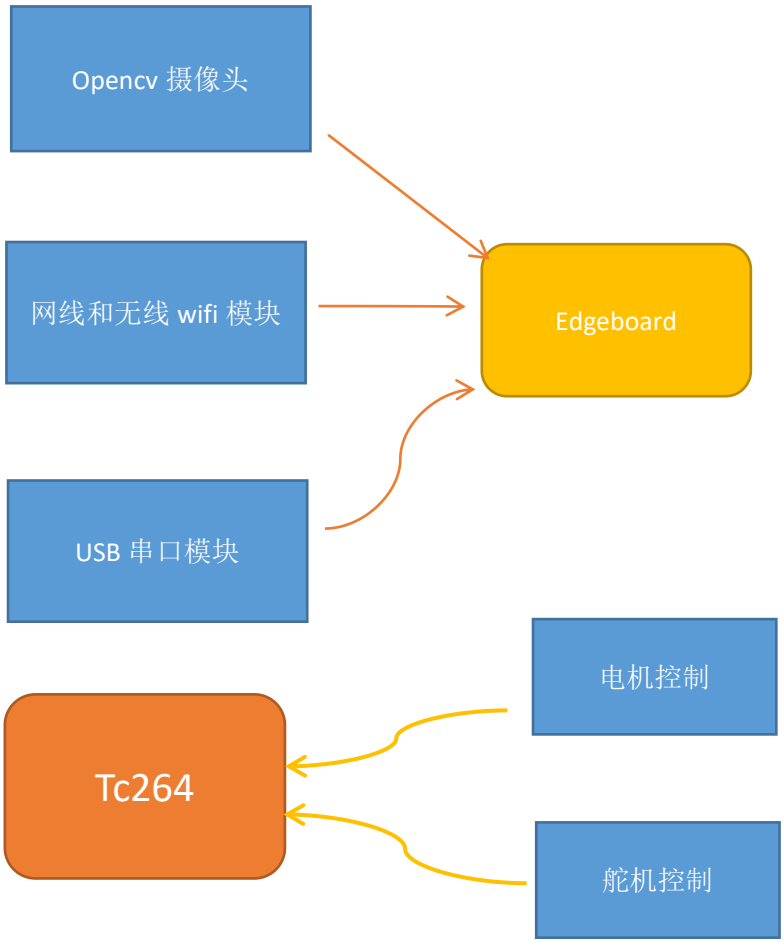
1、硬件平台的搭建：主要包括编码器的安装，摄像头的安装，舵机的安装，电机驱动安装等。

2、软件平台的搭建：系统初始化模块，图像识别模块，控制模块，驱动模块，网线调试模块等组成，对车模进行调试。

3、控制算法的研究：通过舵机的方向环、电机的速度环, 二组控制系统结合控制车模在赛道上速度的控制以及实时的转向控制。

4、视觉识别算法的研究：数据集的制作，深度模型搭建，模型量化，模型部署。

2.2 系统总体方案设计图



第三章

智能车机械结构设计

3.1 车模信息

根据第十七届智能车竞赛规则对完全模型组的要求，使用 I 车模完成任务，细则中对车模尺寸没有加以限制，我们在车模重心稳定的情况下，安装了器件。

3.2 各大传感器的安装

3.2.1 编码器的安装

编码器直接安装在车模电机的孔位上，通过齿轮之间的传动，测量当前车轮转速。

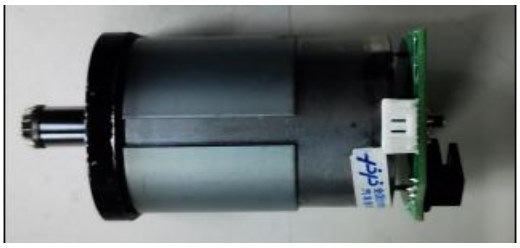


图 3.1 车模安装的一个编码器

3.2.2 摄像头的安装

我们车模的寻迹任务主要通过 opencv 数字摄像头实时采集赛道图像信息完成，摄像头的安装必须要稳定且合理。我们使用一根粗碳素杆用于安装摄像头，由于在车模运行时可能会出现抖动的情况，且摄像头的安装位置比较高，会导致粗碳素杆的晃动，影响所采集的图像质量，因此我们使用热熔胶辅助固定。

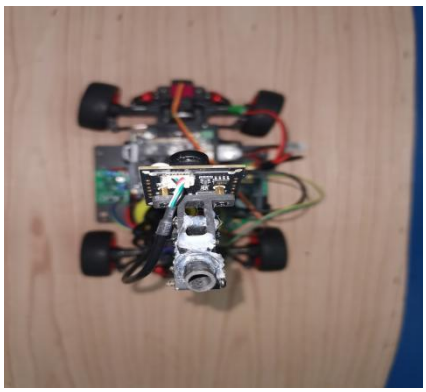


图 3.2 车模安装的摄像头

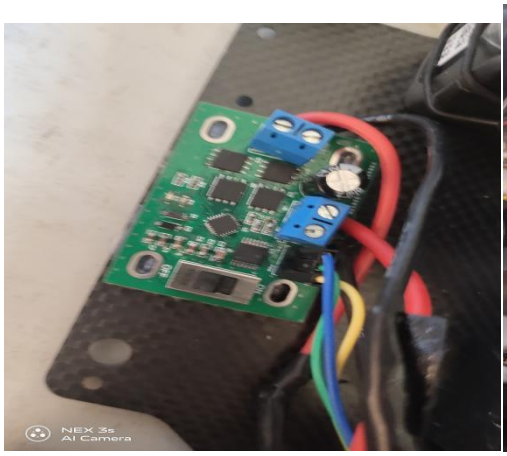
3.2.3 车壳的安装

在我们的完全模型组的规则中我们必须使用车壳来完成比赛，所以我们使用了官方推荐的车壳，并且做了一些改变，在旁边开了个车门方便网线的插入。

图 3.3 车壳的安装

3.4 PCB 板的安装

我们使用到的 PCB 板主要有二个，主板、驱动，在布局是，由于考虑到安装车壳的问题，在安装时把 PCB 板尽量迎合车模的大小，以此能够安装好车壳。



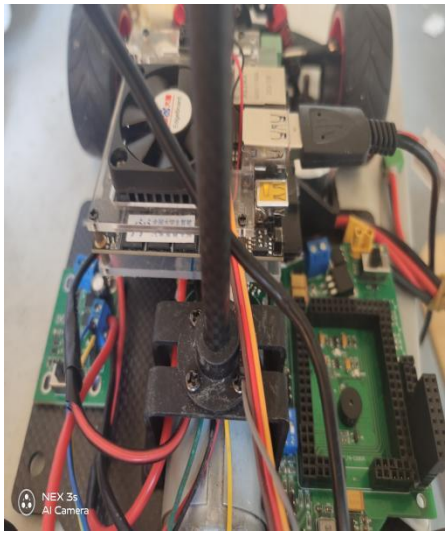


图 3.4 PCB 安装

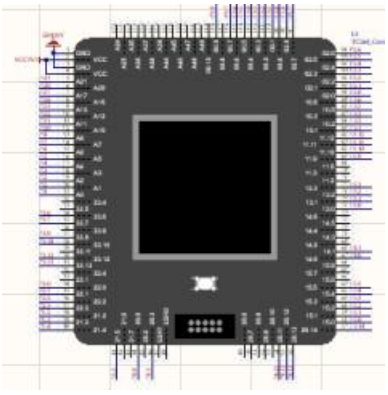
第四章

智能车硬件设计

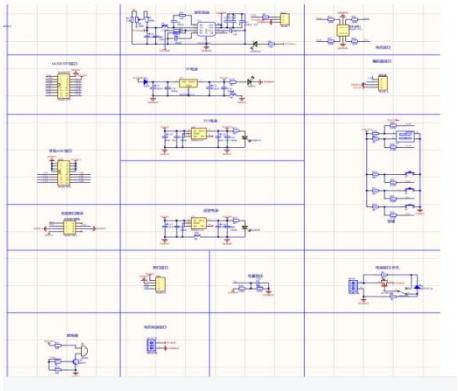
智能车硬件电路部分主要的包括：主控板模块、驱动模块以及其他周边调试模块。各模块的总体设计原则是：紧凑易于拆换、稳定可靠。但根据各模块的不同，又有不同的设计要求，本章对各个模块的设计进行详细描述。

4.1 运动控制控模块

tc264 核心板



tc264 主板原理图



4.2 主控板电源管理模块

硬件电路的电源由 12.6V 的蓄电池提供，而由于电路中不同模块的工作电压和电流容量各不相同，所以需要将电池电压转换成各个模块所需电压。主控板采用了外设与 MCU 分开供电的方案，MCU 由单独一片 SPX2940-5.0 供电。因此，在电池电压以及 5V 供电正常的情况下，外设用电的变化不对最小系统的供电产生影响，保证了其电源的稳定可靠。

4.2.1 5V 稳压电路

SPX2940-5.0 稳压电路如图 4.1 所示。

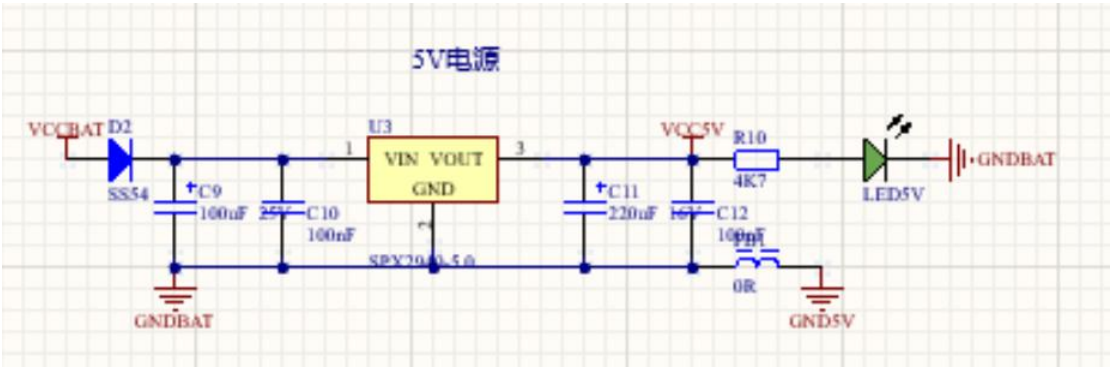


图 4.1 SPX2940-5.0

4.2.2 舵机可调稳压电路

SY8205FCC 可调稳压舵机电源如图 4.2.所示。

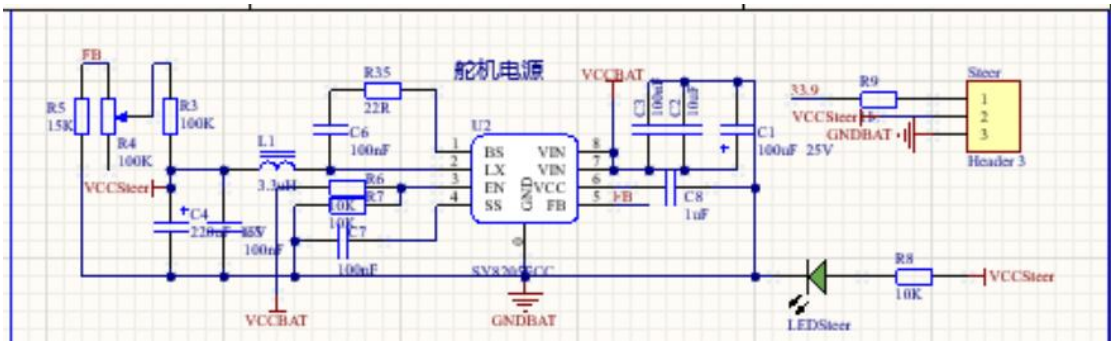


图 4.2 SY8205FCC

4.3 电机驱动电路

电机控制从控制方法上可以分为开环控制和闭环控制两种。开环控制在用法上比较简单。通常情况下的电机转速只需考虑输出，没有反馈，但是其缺点是速度控制的精度比较低，不能适应不同的赛道环境。另外一种闭环控制，电机的速度控制信号输出由需要的速度和电机的实际转速二者决定，即需要对电机的实际转速进行采集和反馈。这种做法的好处是控制精度比较高，对赛道的适应性会好很多的闭环控制，是通过软件的自动控制算法实现的，需要将电机的转速反馈给 tc264，通过软件上的自动控制算法，由需要的速度同实际的速度的偏差， 给出纠正值，达到对速度的稳定控制。这种方案在电机实际速度的采集上使用了编码器。编码器一个时间周期内输出的脉冲数表征了电机的实际转速 在电机驱动上，我们 MOS 管作为分立元件搭建了 H 桥驱动电路，如图 4.3。通过逻辑设计，可以让电机处于多种模式下工作，经过赛道试验，电机的加减速效果好，满足要求。

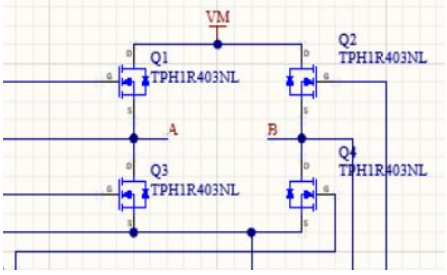


图 4.4 H 桥电机驱动电路

第五章

控制软件系统设计

我们制作的智能车系统采用 opencv 数字摄像头进行赛道元素的识别，对采集到的图像数据进行处理便是整个转向控制系统的核心内容，需要足够健壮的算法让车模能够按照规则要求运行不同的元素。在智能车的转向控制与速度控制上，我们使用传统的 PID 控制算法，在舵机转向上采用 PD 算法进行控制，电机速度采用 PI 算法进行控制。经过不断的调试，达到了比较理想的运行效果。

5.1 赛道边界提取及处理

5.1.1 原始图像的特征

摄像头的镜头选用 60 度无畸变免驱动镜头，在保证足够的广角下，同时采集到的图像不会发生畸变，程序上可以减少图像去畸变的运行时间。在保证图像信息完整的情况下，我们选择尽可能的缩小图像的尺寸，以减小图像处理的时间。经过粗略调试后，采集的图像尺寸我们设置为 120*80，即宽 180 个像素，高 80 个像素，此时的图像比较完整，且足以用于路径的规划与赛道元素识别，原始图像如图 5.1~5.6 所示。

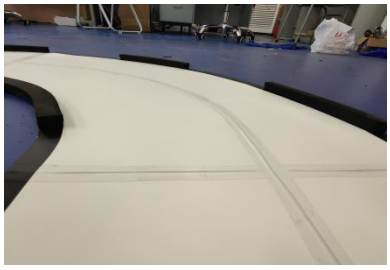


图 5.1 普通弯道

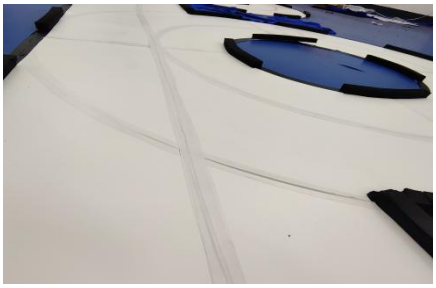


图 5.2 圆环

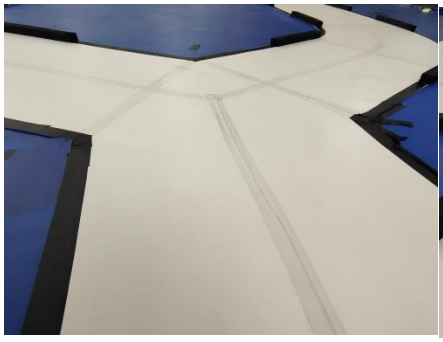


图 5.3 三岔路口

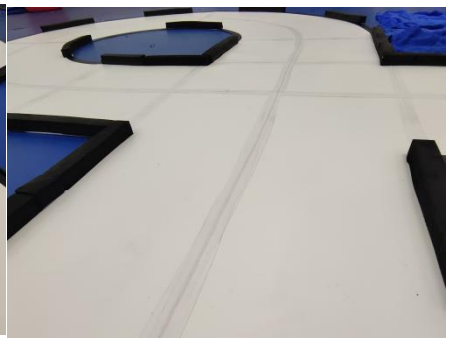


图 5.4 十字路口

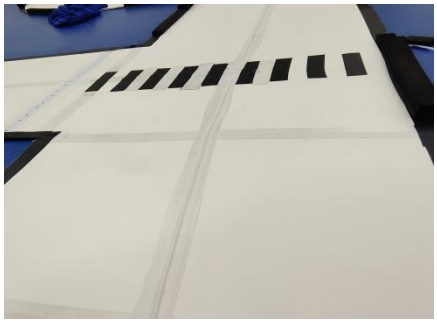


图 5.6 车库

5.1.2 图像二值化

我们将 opencv 数字摄像头采集到的彩色图像，先通过灰度转化再通过大津法进行图像的动态二值化处理。当白天光线较强时，我们采取调节摄像头的曝光以适应当前环境的光线影响，图像二值化的稳定是提取赛道特征的决定性因素。如图 5.7 转灰度二值化部分代码。

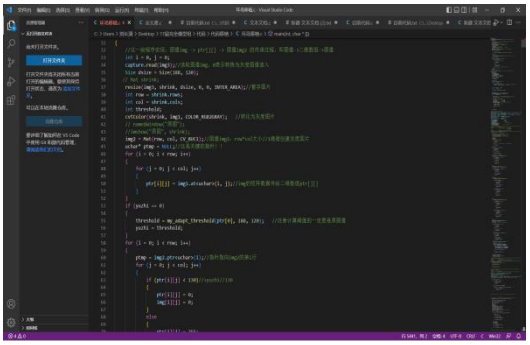


图 5.7 部分代码

5.1.3 图像边界提取

边界提取的基本思想如下：

（1）由近到远遍历每一行的图像信息，从而确定图像的左右边界，首先将最近的一行图像信息从左至右完整遍历一次确定出右边界，再由右至左遍历出左边界。

（2）在确定好首行的边界后，对左右边界取平均作为中点。后继的每一行图像信息都将以上一行的中点作为当前行的边界遍历起始点，分别向两边遍历图像，从而确定出左右边界，直至判断出当前行为截止行为止。

（3）对于某些特殊元素来说，单单利用左右边界的信息在某些情况可能会出现元素之间的误判现象，例如十字与三岔路口。于是我们还做了元素识别的边界信息，即对图像的每一列由下至上的遍历一次图像的边界。处理结果如图 5.8 与图 5.9 所示.

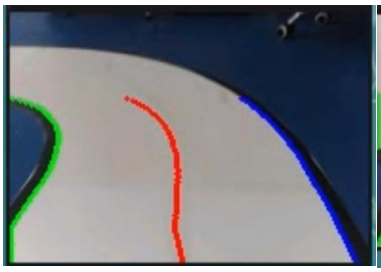


图 5.8 左右边界

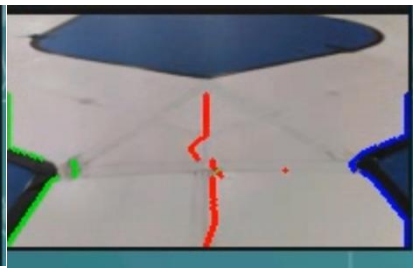


图 5.9 元素识别边界

5.2 赛道元素识别

赛道元素的辨识是保证车模按照规定路线正常运行的前提，对于一些特殊元素不仅需要正确识别，还要保证车模在高速运行时车模的转向控制稳定。因此赛道元素正确识别的情况下，对车模路径的优化也是必不可少的。总之，图像处理在整个运行过程中属于核心要素。

5.2.1 圆环处理

圆环的识别，相对其他元素特征比较明显，几乎不会出现误判的情况。摄像头处理圆环时，其前瞻与电磁传感器相比会长很多，可以提前预判到圆环的出现，当入环偏慢时，速度上可以做相应的决策。对于图像而言，圆环的特征为一侧是长直道，另一侧是一段圆弧。圆弧侧的边界特征便是会先出现一段丢线，随后才是一段圆弧，这个特征在其他元素上几乎是不会出现的，图像处理结果如图 5.10 与图 5.11 所示。

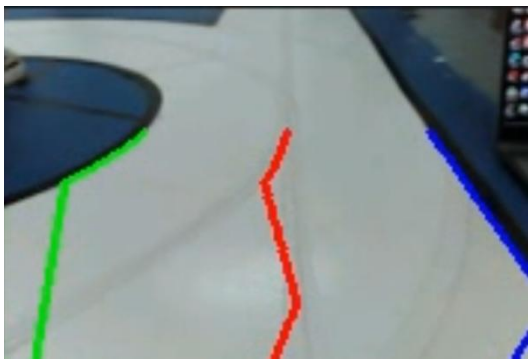


图 5.10 圆环预判断

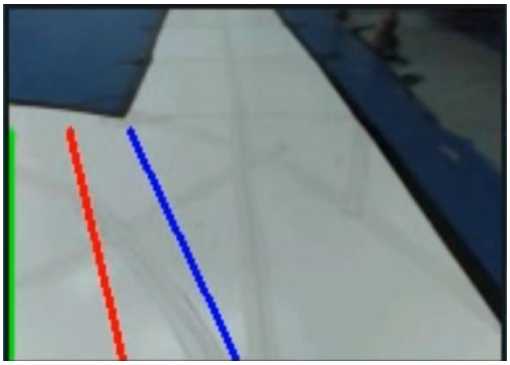


图 5.11 入环

5.2.2 三岔路口

在车模高速运行时，刹车需要较长的一段记录缓冲，因此三岔路口的预判显得至关重要。对于三岔路口的提前预判就需要使用到前面所提到的元素识别边界，如图 5.11 所示，元素识别边界会提前预判三叉从而向下位机做出提前打角。同样的在三岔路口转向时，也可以采用此特征进行补线的操作，如图 5.12 所示。

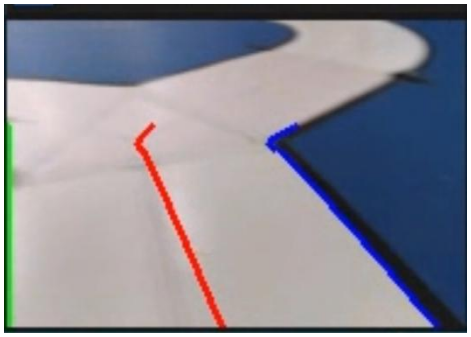


图 5.11 预判三岔路口

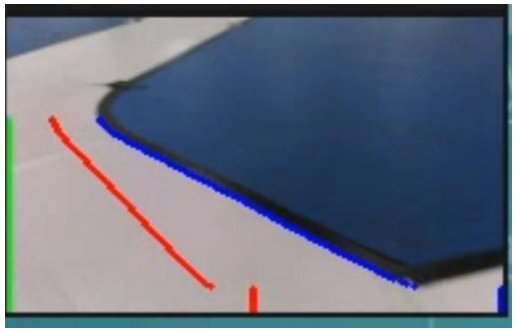


图 5.12 三岔路口补线

5.2.3 十字路口

在车模运行速度不高时，十字路口可以选择不进行处理，直接使用各行中点的加权平均即可比较稳定的驶过。但是在高速运行时，可能因为某些情况导致车模转向有点偏离轨道，使得图像中点不稳定，使车模冲出赛道，因此需要对十字路口路段进行补线的操作。正入十字路口时图像特征比较明显，左右边界都存在连续的丢线情况，找到相应的拐点进行连接即可. 如图 5.13 十字补线代码。

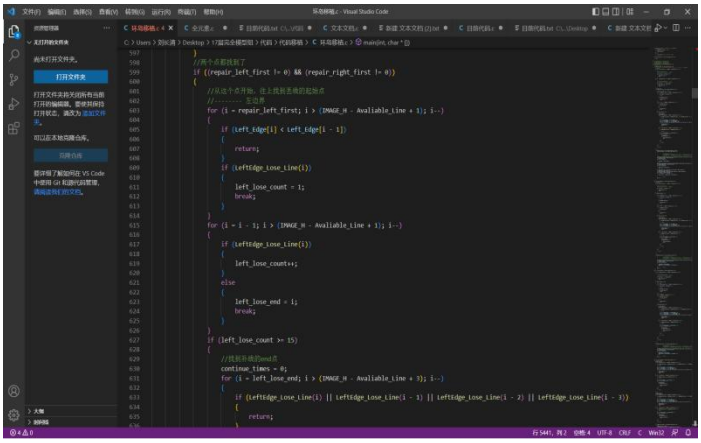


图 5.13 部分代码

5.2.4 车库

在车模运行完成两周之后需要回到车库当中，车库处含有相间分布的黑胶带以及强磁铁，采用干簧管入库的方案，在车模高速行驶时需要急刹车与倒车,处理起来相当麻烦，相对来说也比较影响成绩，而且还对寻迹有一定的影响。因此我们通过图像的特征来进行入库操作。车库的识别我们并没有采取提前预判的思想，而是通过对近处的图像进行处理，当出现车库的特征时，直接控制舵机转向入库。车库特征的识别方案是先找到车库侧的边界跳变，然后再逐行扫描图像信息，分析图像的黑白跳变次数。

5.3 控制算法

5.3.1 控制算法主要理论

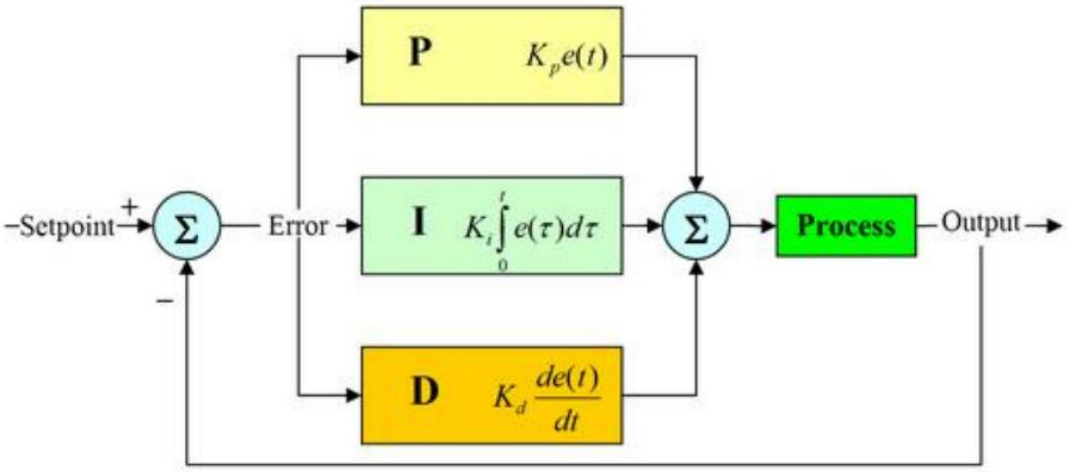


图 5.24 PID 控制系统框图

在实际工程中，应用最为广泛的调节器控制规律为比例、积分、微分控制，简称 PID 控制，又称 PID 调节。

PID 控制器问世至今以其结构简单、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。当被控对象的结构和参数不能完全掌握，或得不到精确的数学模型时，控制理论的其它技术难以采用时，系统控制器的结构和参数必须依靠经验和现场调试来确定，这时应用 PID 控制技术最为方便。即当我们不完全了解一个系统和被控对象，或不能通过有效的测量手段来获得系统参数时，

最适合用 PID 控制技术。PID 控制，实际中也有 PI 和 PD 控制。PID 控制器就是根据系统的误差，利用比例、 积分、微分计算出控制量进行控制的。其中我们智能车控制系统中舵机转向的方向环采用是 PD 算法控制，电机的速度环使用是 PI 算法。

5.3.2 舵机转向的 PD 控制算法

在舵机系统的闭环控制中，我们采用的是位置式的 PD 控制算法，如公式 5.1 所示在这里面，e(t)=目标值-当前值，自然是离散数据，也就是说 $u(t)_1=K_pe(t)$

这个输出是根据当前值和目标值的差，乘以了一个比例系数得到的输出。

在初期低速调试时，只采用一组 PD 参数可以让车模稳定行驶在赛道上，但随着速度的上升，只通过固定的 P 值运行时会出现直道振荡太多或者过弯转向不及时的问题，因此我们采用动态 P 值算法。

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

（公式 5.1）

当 P 值整定好时，随着速度的提升，在过弯之后的直道可能会出现抖动的情况，此时逐渐将 D 值调大，提高动态响应的能力，调至满意的效果。经过不断的调试，最终我们确定了一组比较理想的参数。

5.3.3 电机速度的 PI 控制算法

在速度环的控制上，闭环控制是非常有必要的，开环容易受外界的各种环境因素所影响，如摩擦力、电池电压等。在电机速度环上我们采用的是增量式 PI 算法，调试过程中先整定 I 值，当电机的响应速度达到我们想要是结果时，再逐渐整定 P 值。经过不断调试，最终确定好了一组在车模寻迹时所使用的参数。如图 5.14 电机控制代码。

```
erro_this=encoder-encoder1;     //本次差值
kp=22;
erro_diff=erro_this-erro_last;
speed_power=((erro_this-erro_last)*kd+erro_this*kp)+1950;
erro_last=erro_this;
if(speed_power>=3600)
{
    speed_power=3600;
}
if(0<=speed_power) //电机1   正转 设置占空比为 百分之 (1000/GTM_ATOM0_PWM_DUTY_MAX*100)
{
    pwm_duty(MOTOR1_PWM, speed_power);
    gpio_set(MOTOR1_DIR, 0);
}
else                //电机1   反转
{
    pwm_duty(MOTOR1_PWM, -speed_power);
    gpio_set(MOTOR1_DIR, 1);
}
```

图 5.14 部分代码

5.5 图像识别部分

本届比赛图像识别任务分别为泛型区、施工区、数字、加油站并根据识别的结果进行相应的判断。

我们通过百度飞桨官方提供的 aistudio 提供的平台来训练模型，只要把采集打包好的数据集放在 aistudio 上去训练即可。如图 5.15，5.16 所示。

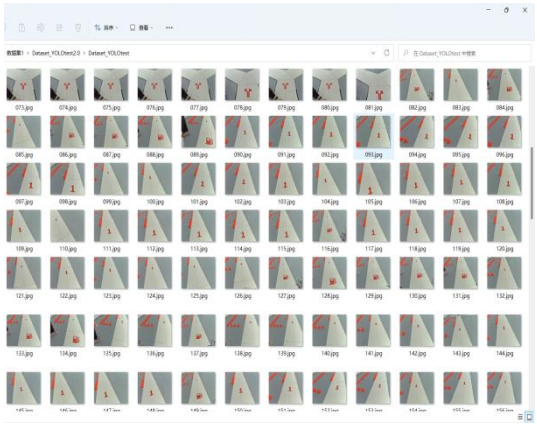


图 5.15 采集好的数据集

图 5.16 aistudio 模型训练界面

之后把训练出来的模型放到 edgeboard 中，并做出相应的识别。如图 5.27 所示。

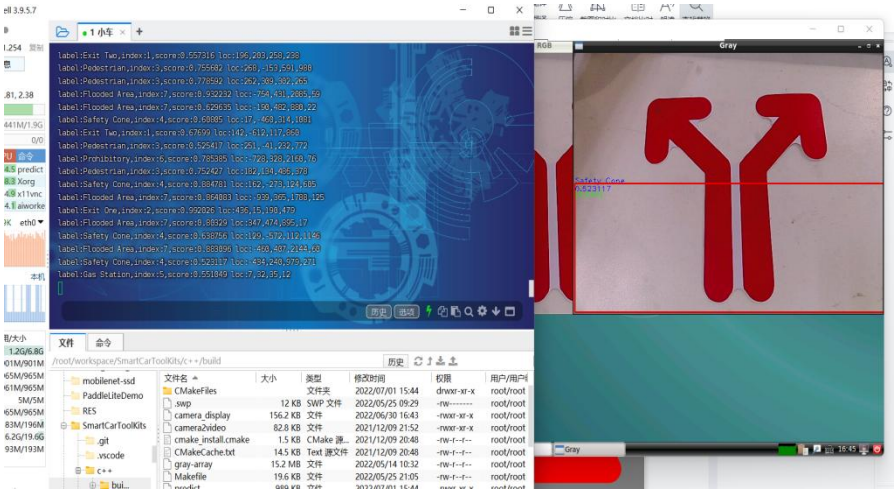


图 5.17 部署到 edgeboard 中识别

第六章

开发平台介绍

6.1 AURIX Development Studio

软件开发工具为 AURIX Development Studio 开发软件（以下简称 ADS）。ADS 是英飞凌公司于 2019 年底推出的集成开发环境，支持英飞凌 TriCore 内核 AURIX 系列 MCU。ADS 是一个完整的开发环境，包含了 Eclipse IDE、C 编译器、Multi-core 调试器、英飞凌底层驱动库（low-level driver iLLD)，同时对于编辑、编译及调试应用代码没有时间及代码大小的限制。

图 6.1 ADS 开发软件

6.2 Altium Designer

硬件电路原理图和 PCB 的绘制在 Altium Designer 19 的环境中进行，它是原 Protel 软件开发商 Altium 公司推出的一体化的电子产品开发系统，主要运行在 Windows 操作系统。这套软件通过把原理图设计、电路仿真、PCB 绘制编辑、拓扑逻辑自动布线、信号完整性分析和设计输出等技术的完美融合，为设计者提供了全新的设计解决方案，使设计者可以轻松进行设计，熟练使用这一软件使电路设计的质量和效率大大提高。其操作界面见图 6.2

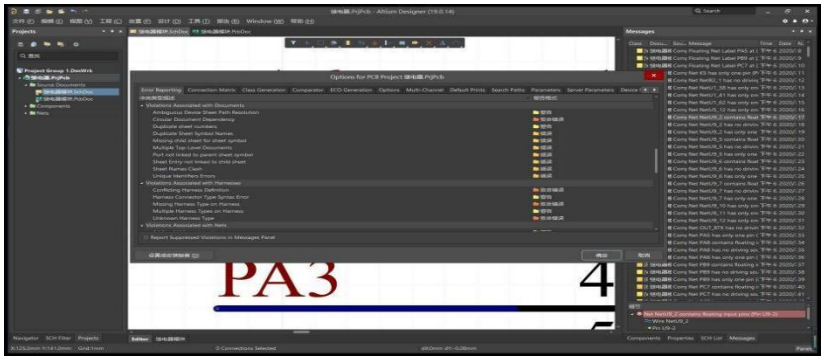


图 6.2 AD19 操作界面

6.3 FinalShell

FinalShell 官方版是一款功能强大的远程连接软件，FinalShell 最新版支持 shell 和 sftp 同屏显示，并同步切换目录，软件还带有命令自动提示。这个软件是我们用来连接 edgeboard，实现代码的改写以及 edgeboard 的状态。如图 6.3 所示。

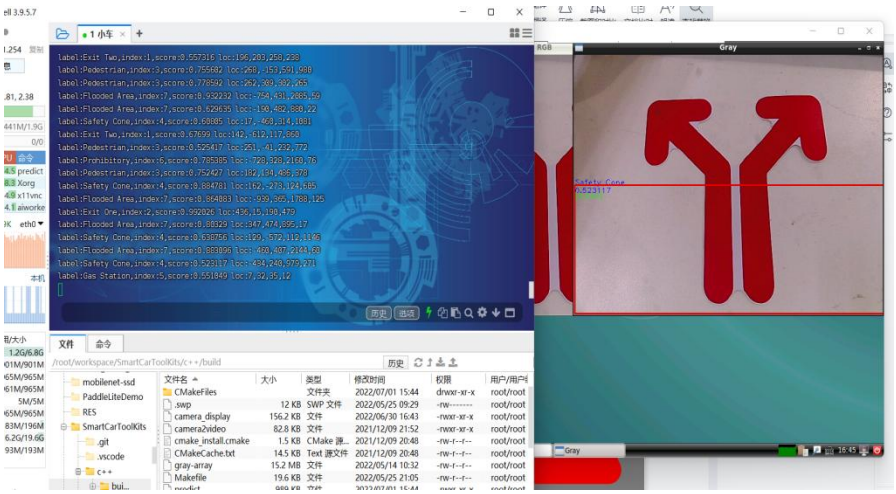


图 6.3 FinalShell 操作界面

6.4 Virtual Network Console

Virtual Network Console 是虚拟网络控制台(简称 VNC)。它是一款优秀的远程控制工具软件，由著名的 AT&T 的欧洲研究实验室开发的。VNC 是在基于 UNIX 和 Linux 操作系统的免费的开源软件，远程控制能力强大，高效实用，其性能可以和 Windows 和 MAC 中的任何远程控制软件美。它可以用来可视化 edgeboard，实时观察智能车的运动情况。如图 6.4 所示。

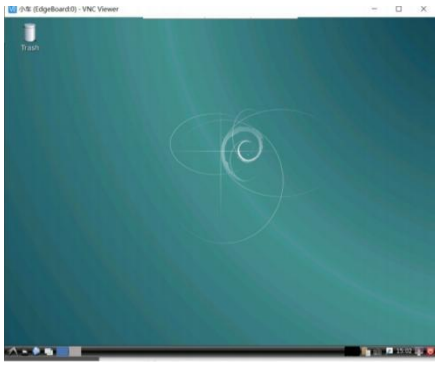


图 6.4 edgeboard 再 VNC 上的可视化

第七章

智能车基本参数

7.1 车模基本参数

总体重量：2.3Kg
长：230mm
宽：206mm
高：320mm

7.2 电路参数

电路功耗：20W
电容总容量：1600uF

7.3 传感器种类以及个数

编码器：1 个
opencv 数字摄像头 ：1 个

7.4 赛道信息检测精度、频率

赛道信息检测精度：2mm
频率：5ms

第八章

总结

在这份报告中，包含了我们组半年多时间以来的调试思路与方法，以及硬件、机械与软件等方面的设计。在本次智能车竞赛当中，自竞赛规则发布以来，我们组在此期间不断研究，深思熟虑与反复测试后，才最终确定了一套完整的方案。

一个完整的智能车是软件与硬件的结合，目标是在最短时间稳定高效的完成比赛。从车模的搭建、车模的整体机械结构设计，到硬件电路的设计、PCB 板的制作和设计，再到舵机的控制算法、双电机的差速控制算法、采用摄像头作为传感器、图像识别等。总之，从电路的设计到车模的搭建，从到程序的编写到实际的调试，我们遇到了很多困惑和难题，也增强了许多技能和能力。每一步我们都查阅了大量的资料作为参考。在这个过程中，大家学到了很多，积累了很多，能够一步一步走下来，靠的是整个团队的协作和团结。大家学到了很多，提高了很多。无论如何，这段一起做车经历终将让我们回忆铭记。

参考文献

[1] 邵贝贝. 嵌入式实时操作系统[LC／OS-Ⅱ(第 2 版)[M]. 北京．清华大学出版社．2004
[2] 邵贝贝. 单片机嵌入式应用的在线开发方法[M].北京.清华大学出版社.2004
[3] 王晓明. 电动机的单片机控制[M]．北京. 北京航空航天大学出版社．2002
[4] 臧杰，阎岩. 汽车构造[M]. 北京. 机械工业出版社．2005
[5] 安鹏，马伟．S12 单片机模块应用及程序调试[J]. 电子产品世界.2006
[6] 童诗白，华成英．模拟电子技术基础[M]．北京. 高等教育出版社．2000
[7] 谭浩强.C 程序设计[M].北京:清华大学出版社.2005
[8] 郭芳,曹桂琴.数据结构基础[M].大连：大连理工大学出版社，1994
[9] 邵贝贝.单片机嵌入式应用的在线开发方法[M].北京:清华大学出版社,2004
[10] 胡寿松.自动控制原理(第六版)[M].科学出版社,2014
[11] 张文春．汽车理论[M]．北京．机械工业出版社．2005

附录

源程序代码

```
EdgeBoard：VideoCapture capture(0);//读取图像 img。0 表示转
换为灰度图像读入
Mat img1;
Mat img2;
Mat img3;//原图
Mat shrink;
uchar ptr[120][188];
uint img[120][188];
int erro;
#define IMAGE_H     120
#define IMAGE_W     188
#define IMAGE_H     120
#define IMAGE_W     188
#define IMG_WHITE   255
#define IMG_BLACK   0
int yuzhi = 0;
int my_adapt_threshold(uchar* image, int col, int row); //
注意计算阈值的一定要是原图像
void get_image()
{
```

```
    //这一段程序实现：图像 img -> ptr[][] -> 图像 img2 的传递过程，即图像->二维数组->图像
    int i = 0, j = 0;
    capture.read(img3);//读取图像 img。0 表示转换为灰度图像读入
    Size dsize = Size(188, 120);
   // Mat shrink;
    resize(img3, shrink, dsize, 0, 0, INTER_AREA);//暂存图片
    int row = shrink.rows;
    int col = shrink.cols;
    int threshold;
    cvtColor(shrink, img1, COLOR_RGB2GRAY);   //转化为灰度图片
    // namedWindow("原图");
    //imshow("原图", shrink);
    img2 = Mat(row, col, CV_8UC1);//图像 img2：row*col 大小
    //1 通道创建灰度图片
    uchar* ptmp = NULL;//这是关键的指针！！
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
```

```
        {
            ptr[i][j] = img1.at<uchar>(i, j);//img 的矩阵数
据传给二维数组 ptr[][]
        }
    }
    if (yuzhi == 0)
    {
        threshold = my_adapt_threshold(ptr[0], 188, 120);
//注意计算阈值的一定要是原图像
        yuzhi = threshold;
    }
    for (i = 0; i < row; i++)
    {
        ptmp = img2.ptr<uchar>(i);//指针指向 img2 的第 i 行
        for (j = 0; j < col; j++)
        {
            if (ptr[i][j] < 130)//>yuzhi//130
            {
                ptr[i][j] = 0;
                img[i][j] = 0;
            }
            else
```

```
            {
                ptr[i][j] = 255;
                img[i][j] = 255;
            }
            ptmp[j] = ptr[i][j];//二维数组数据传给 img2 的第
i 行第 j 列
        }
    }
    // 等待 100000 ms 后窗口自动关闭
    waitKey(1);
}
int my_adapt_threshold(uchar* image, int col, int row)   //
注意计算阈值的一定要是原图像
{
#define GrayScale 256
    int width = col;
    int height = row;
    int pixelCount[GrayScale];
    float pixelPro[GrayScale];
    int i, j, pixelSum = width * height / 4;
    int threshold = 0;
    uchar* data = image;  //指向像素数据的指针
```

```
    for (i = 0; i < GrayScale; i++)
    {
        pixelCount[i] = 0;
        pixelPro[i] = 0;
    }

    int gray_sum = 0;
    //统计灰度级中每个像素在整幅图像中的个数
    for (i = 0; i < height; i += 2)
    {
        for (j = 0; j < width; j += 2)
        {
            pixelCount[(int)data[i * width + j]]++;  //将当
前的点的像素值作为计数数组的下标
            gray_sum += (int)data[i * width + j];       //
灰度值总和
        }
    }

    //计算每个像素值的点在整幅图像中的比例

    for (i = 0; i < GrayScale; i++)
```

```
    {
        pixelPro[i] = (float)pixelCount[i] / pixelSum;

    }

    //遍历灰度级[0,255]
    float w0, w1, u0tmp, u1tmp, u0, u1, u, deltaTmp, deltaMax
= 0;
    w0 = w1 = u0tmp = u1tmp = u0 = u1 = u = deltaTmp = 0;
    for (j = 0; j < GrayScale; j++)
    {

        w0 += pixelPro[j];  //背景部分每个灰度值的像素点所占
比例之和   即背景部分的比例
        u0tmp += j * pixelPro[j];  //背景部分 每个灰度值的点
的比例 *灰度值

        w1 = 1 - w0;
        u1tmp = gray_sum / pixelSum - u0tmp;

        u0 = u0tmp / w0;              //背景平均灰度
        u1 = u1tmp / w1;              //前景平均灰度
```

```
        u = u0tmp + u1tmp;              //全局平均灰度
        deltaTmp = w0 * pow((u0 - u), 2) + w1 * pow((u1 - u),
2);//平方
        if (deltaTmp > deltaMax)
        {
            deltaMax = deltaTmp;//最大类间方差法
            threshold = (int)j;
        }
        if (deltaTmp < deltaMax)
        {
            break;
        }

    }
    return threshold;
}

//******************************** 边界变量
#define NoFind_BottomUp       120//90
#define Find_BottomUp_Line(j)       (img[j+2][i] ==
IMG_WHITE && img[j+1][i] == IMG_WHITE && img[j][i] ==
IMG_BLACK)
```

```
//寻找左边界的方法       //黑 2 黑 1 白，记录下来黑 1 的纵坐标
#define Find_LeftEdge   (img[i][j] == IMG_BLACK && img[i][j
- 1] == IMG_BLACK && img[i][j + 1] == IMG_WHITE)
//寻找右边界的方法       //白 黑 1 黑 2，记录下来黑 1 的纵坐标
#define Find_RightEdge  (img[i][j] == IMG_BLACK && img[i][j
+ 1] == IMG_BLACK && img[i][j - 1] == IMG_WHITE)
//右边界是否丢线情况      i 表示行
#define RightEdge_Lose_Line(i)      (Right_Edge[i] ==
(IMAGE_W - 1))
#define RightEdge_NoLose_Line(i)        (Right_Edge[i] !=
(IMAGE_W - 1))
//左边界是否丢线情况
#define LeftEdge_Lose_Line(i)       (Left_Edge[i] == 0)
#define LeftEdge_NoLose_Line(i)     (Left_Edge[i] != 0)

uint Right_Edge[IMAGE_H];                       //右边界坐
标
uint Left_Edge[IMAGE_H];                        //左边界坐
标
int Middle_Point[IMAGE_H];                      //中点坐标
uint Left_LoseLine;                                 //左边
界丢线数
```

```
uint Right_LoseLine;                                    //右边
界丢线数
uint Avaliable_Line = IMAGE_H - 1;
//有效行数
uint Repair_Data[IMAGE_W];
uint Bottom_Up_Line[IMAGE_W];                  //自下而上搜线，
纵向搜线
/*****************************************************
*************************************************
* Function Name：     clear_find_line_data()
* input：                    none
* output：                 none
* state：                    复位寻线所用到的所有变量
*****************************************************
*************************************************/
void clear_find_line_data(void)
{
    int i;

    for (i = IMAGE_W - 1; i > IMAGE_H - 1; i--)
    {
        Bottom_Up_Line[i] = NoFind_BottomUp;
```

```
}

for (i = IMAGE_H - 1; i >= 0; i--)
{
    Right_Edge[i] = IMAGE_W - 1;
    Left_Edge[i] = 0;
    //Middle_Point[i] = (IMAGE_W >> 1) - 1;
    Bottom_Up_Line[i] = NoFind_BottomUp;
}
Left_LoseLine = 0;       //左边界丢线数
Right_LoseLine = 0;      //右边界丢线数
//Avaliable_Line = IMAGE_H;      //有效行数

  //清除圆环变量
/* Round_Jump = 0;
Round_JumpStart = 0;     //圆环
Round_JumpEnd = 0;

//三岔路口变量
Bifurcation.Left_Turn = 0;
Bifurcation.Right_Turn = 0;
Bifurcation.Car_Turn_Flag = 0;
```

```
    // Bifurcation.Min_Index = 0;       //三岔路口处理里面有
清零
    sancha = 0;*/
}

/*********************************************************
**********************************************
* Function Name:      get_boundary_dir()
* input:                        uint8 img[][IMAGE_W]    :    传
入的二值化图像
* input:                         uint8 dir    : 当左右都丢线时，
开始搜索的方向

dir == 1,向左

dir == 0,向右
* output:                  none
* state:                         分别向左、右搜索边界，以白黑黑
的方式确定边界，
*********************************************************
**********************************************/
void get_boundary_dir(uint img[][IMAGE_W], uint dir)
```

```
{
    int i, j;
    // int mid_line_dif = 0;   //本次中点的偏差、上一次中点的偏差
  //   int aver = 0, temp = 0; //均值、中间变量
    uint find_start_point = IMAGE_W / 2; //搜线起始列
    //uint8 Before_4_MiddleJump_Flag = 0; //前四行有中点跳变标志位
    uint Avaliable_Line_Break_Flag = 0;    //有效行强制结束标志位

    //搜索首行边界
    i = IMAGE_H - 1;
    //扫描左边界
    for (j = IMAGE_W / 2; j > 0; j--)
    {
        if (Find_LeftEdge)  //找左边界
        {
            Left_Edge[i] = j;
            break;
        }
```

```
        else if (j == 1)    //左边界没找到，可能是丢线，也可
能比较靠近右边
        {
            Left_Edge[i] = 0;
        }
    }
    //扫描右边界
    for (j = IMAGE_W / 2; j < IMAGE_W - 1; j++)
    {
        if (Find_RightEdge)
        {
            Right_Edge[i] = j;
            break;
        }
        else if (j == IMAGE_W - 2)
        {
            Right_Edge[i] = IMAGE_W - 1;
        }
    }
    //如果左边没找到边界 ，右边找到了边界
    if (LeftEdge_Lose_Line(i) && RightEdge_NoLose_Line(i))
    {
```

```
    //左边界从右边界开始搜线
    for (j = Right_Edge[i] - 1; j > 0; j--)
    {
        if (Find_LeftEdge)
        {
            Left_Edge[i] = j;
            break;
        }
        else if (j == 1)
        {
            Left_LoseLine++;
            break;
        }
    }
}
//如果右边没找到边界，左边找到了
if (RightEdge_Lose_Line(i) && LeftEdge_NoLose_Line(i))
{
    //扫描右边界
    for (j = Left_Edge[i] + 1; j < IMAGE_W - 1; j++)
    {
        if (Find_RightEdge)
```

```
            {
                Right_Edge[i] = j;
                break;
            }
            else if (j == IMAGE_W - 2)
            {
                Right_LoseLine++;
                break;
            }
        }
    }
    if (dir == 1)
    {   //如果两边都丢线，重新扫描边界
        if (RightEdge_Lose_Line(i) && LeftEdge_Lose_Line(i))
        {
            //右边界从最左边开始搜线
            for (j = 1; j < IMAGE_W - 1; j++)
            {
                if (Find_RightEdge)
                {
                    Right_Edge[i] = j;
                    break;
```

```
            }
            else if (j == IMAGE_W - 2)
            {
                Right_LoseLine++;
                break;
            }
        }
        //左边界从右边界处开始搜线
        for (j = Right_Edge[i]; j > 0; j--)
        {
            if (Find_LeftEdge)
            {
                Left_Edge[i] = j;
                break;
            }
            else if (j == 1)
            {
                Left_LoseLine++;
                break;
            }
        }
    }
```

```
}
else
{   //如果两边都丢线，重新扫描边界
    if (RightEdge_Lose_Line(i) && LeftEdge_Lose_Line(i))
    {
        //左边界从最右边开始搜线
        for (j = IMAGE_W - 2; j > 0; j--)
        {
            if (Find_LeftEdge)
            {
                Left_Edge[i] = j;
                break;
            }
            else if (j == 1)
            {
                Left_LoseLine++;
                break;
            }
        }
        //右边界从左边界处开始搜线
        for (j = Left_Edge[i]; j < IMAGE_W - 1; j++)
        {
```

```
                if (Find_RightEdge)
                {
                    Right_Edge[i] = j;
                    break;
                }
                else if (j == IMAGE_W - 2)
                {
                    Right_LoseLine++;
                    break;
                }
            }
        }
    }
    //计算本行中点坐标
    Middle_Point[i] = (Left_Edge[i] + Right_Edge[i]) >> 1;
    find_start_point = Middle_Point[i];
    //扫描剩余行
    for (i = IMAGE_H - 2; i > 3; i--)   //最上面四行不搜索
    {
        //左边界
        for (j = find_start_point; j > 0; j--)
        {
```

```
    if (Find_LeftEdge)
    {
        Left_Edge[i] = j;
        break;
    }
    else if (j == 1)    //左边界丢线
    {
        Left_LoseLine++;
        Left_Edge[i] = 0;
    }
}
//扫描右边界
for (j = find_start_point; j < IMAGE_W - 1; j++)
{
    if (Find_RightEdge)
    {
        Right_Edge[i] = j;
        break;
    }
    else if (j == IMAGE_W - 2)
    {
        Right_LoseLine++;
```

```
                Right_Edge[i] = IMAGE_W - 1;
            }
        }
        //计算本行中点坐标
        Middle_Point[i] = (Left_Edge[i] + Right_Edge[i]) >>
1;

        find_start_point = Middle_Point[i];
        //确定有效行
        if ((img[i - 1][Middle_Point[i]] == IMG_BLACK) &&
            (img[i - 2][Middle_Point[i]] == IMG_BLACK) &&
            (img[i - 3][Middle_Point[i]] == IMG_BLACK)) //
最后一行
        {
            if ((img[i][Middle_Point[i + 1]] == IMG_WHITE) &&
                (img[i - 1][Middle_Point[i + 1]] == IMG_WHITE)
&&
                (img[i - 2][Middle_Point[i + 1]] ==
IMG_WHITE))
            {
                find_start_point = Middle_Point[i + 1];
            }
```

```
            else
            {
                Avaliable_Line_Break_Flag = 1;
                break;
            }
        }
    }
    if (Avaliable_Line_Break_Flag == 1)
        Avaliable_Line = IMAGE_H - i;//有效行数
    else
        Avaliable_Line = IMAGE_H - i - 1;
}
/*************************************************************
*************************************************
* Function Name:      get_boundary()
* input:                    uint8 img[][IMAGE_W]    :    传
入的二值化图像
* output:                 none
* state:                      分别向左、右搜索边界，以白黑黑
的方式确定边界，版本 V3
*************************************************************
*************************************************/
```

```
void get_boundary(uint img[][IMAGE_W])
{
    if (Middle_Point[IMAGE_H - 1] >= 93)//图像宽度
    {
        get_boundary_dir(img, 0);
    }
    else
    {
        get_boundary_dir(img, 1);
    }

}
/*************************************************************
**********************************************
* File Name:      Fkaw_Boundary.c
* Author:         Fkaw
* State:              图像边缘搜索核心程序

*************************************************************
**********************************************/
```

```
#define RIGHT    1    //表示先找右边界，即从最左边往右找边界，然后再找左边界
#define LEFT     0    //表示先找左边界，即从最右边往左找边界，然后再找右边界
 //设置这个方向是为了，保证入库时，正确寻线
 //RIGHT 对应的是右转入库，    LEFT 表示左转入库
#define BOTH_LOSE_FIND_DIR  RIGHT    //两边都丢线时，重新搜线方向
/*配套搜索方案 ： */
#define Find_BottomUp_Line(j)    (img[j+2][i] == IMG_WHITE && img[j+1][i] == IMG_WHITE && img[j][i] == IMG_BLACK)
void from_bottom_to_up(uint img[][IMAGE_W], uint result[], uint start, uint end)     //记录黑边
{
    int i, j;
    for (i = start; i <= end; i++)
    {
        for (j = IMAGE_H - 1; j >= 0; j--)
        {
            if (Find_BottomUp_Line(j))
            {
                result[i] = j;
```

```
                break;
            }
        }
        if (j == -1 && result[i] == NoFind_BottomUp)
        {
            j = 0;
            if (img[j + 2][i] == IMG_WHITE && img[j + 1][i]
== IMG_WHITE)    //白白  --?
            {
                if (img[j][i] == IMG_WHITE)  //白白白   ---
                {
                    result[i] = 0;
                }
            }
        }
    }
}

/*****************************************************************
*********************************/
/*
 *
```

```
 *                                                          扫
线处理函数
 */

/*****************************************************************
*********************************/
void image_Deal(uint img[][IMAGE_W])
{
    clear_find_line_data();          //清除搜线数据
    //边界提取
    get_boundary(img);
    //from_bottom_to_up(img, 0, (IMAGE_W - 1));            //
自下而上搜线
    from_bottom_to_up(img, Bottom_Up_Line, 0, (IMAGE_W - 1));
}

/*****************************************************************
**************************************************
* Function Name:     get_middle_err()
* input:                    none
* output:                 none
```

* state：　　计算中点偏差，其后 30 行的均值作为偏差

**********/

//**********循迹宏**********/

//　　取偏差范围

```
#define CAL_USE_LINE_NUM                    65//55      //计算偏差使用的行数//73-13    71-11
float line_err, line_err2;
int  get_middle_err()
{
    uint i;
    // float use_line_num = (float)CAL_USE_LINE_NUM;
    int line_sum = 0;
    //取 40 行的中点均值
    if (Avaliable_Line < CAL_USE_LINE_NUM)
    {
        for (i = IMAGE_H - 1; i >= IMAGE_H - Avaliable_Line; i--)
        {
```

```
            line_sum += Middle_Point[i];
        }
        line_err = ((float)line_sum) / (1.0f *
Avaliable_Line);
    }
    else
    {
        for (i = IMAGE_H - 1 ; i > IMAGE_H - CAL_USE_LINE_NUM
- 1; i--)
        {
            line_sum += Middle_Point[i];
        }
        line_err = ((float)line_sum) / (CAL_USE_LINE_NUM );
    }
    return line_err;
}

void MyLines()    //绘制直线
{
    //Point 打印Point(第一个点在x轴上的位置,第一个点在y轴
上的位置)
    for (int i = IMAGE_H - 1; i > 5; i--) //画线)
```

```
    {
        Point p(Middle_Point[i], i);//初始化点坐标为(20,20)
        circle(shrink, p, 1, Scalar(0, 0, 255), -1);  // 画半径为 1 的圆(画点）//img3 原图//(0, 0, 255),红色 0, 255, 0 绿色
    }
    //图片名，第一个点，第二个点，线的颜色，线的厚度，线型
    namedWindow("二值化");
    imshow("二值化", shrink);//img2,img3 原图 img1 灰度
}

TC264:int core0_main(void)
{
   get_clk();//获取时钟频率  务必保留

   //桌大大的推文中，建议电磁组电机频率选用 13K-17K

    //在使用本例程请务必确认所使用的驱动类型
    //本例程只支持 DRV8701E 驱动
```

//在测试时可能会出现电机不转动的效果，这个时候可能是信号线接反了，将任意一头 2*3 的牛角座旋转 180 度后再插入试试。

```
//初始化 PWM 引脚
gtm_pwm_init(MOTOR1_PWM, 17000, 0);
gtm_pwm_init(MOTOR2_PWM, 17000, 0);

//初始化方向控制引脚
gpio_init(MOTOR1_DIR, GPO, 0, PUSHPULL);
gpio_init(MOTOR2_DIR, GPO, 0, PUSHPULL);

//设置默认速度  也可以通过在线调试直接修改此值  变化电机速度
//在线调试需要暂停之后才能查看或者修改变量
speed1_power = 3000;
speed2_power = 3000;

//用户在此处调用各种初始化函数等
enableInterrupts();
```

```
while (TRUE)
{
    //如何控制电机的正反转
    //每个电机驱动都有方向控制引脚，引脚为低电平正转，高电平反转
    //PWM 占空比越大电机转的越快

    if(0<=speed1_power) //电机 1   正转 设置占空比为 百分之 (1000/GTM_ATOM0_PWM_DUTY_MAX*100)
    {
        pwm_duty(MOTOR1_PWM, speed1_power);
        gpio_set(MOTOR1_DIR, 0);
    }
    else                    //电机 1   反转
    {
        pwm_duty(MOTOR1_PWM, -speed1_power);
        gpio_set(MOTOR1_DIR, 1);
    }

    if(0<=speed2_power) //电机 2   正转
    {
```

```
        pwm_duty(MOTOR2_PWM, speed2_power);
        gpio_set(MOTOR2_DIR, 0);
    }
    else                    //电机2   反转
    {
        pwm_duty(MOTOR2_PWM, -speed2_power);
        gpio_set(MOTOR2_DIR, 1);
    }

    systick_delay_ms(STM0, 1000);

    speed1_power = -3000;
    speed2_power = -3000;

    if(0<=speed1_power) //电机1   正转 设置占空比为 百分之 (1000/GTM_ATOM0_PWM_DUTY_MAX*100)
    {
        pwm_duty(MOTOR1_PWM, speed1_power);
        gpio_set(MOTOR1_DIR, 0);
    }
    else                    //电机1   反转
    {
```

```
            pwm_duty(MOTOR1_PWM, -speed1_power);
            gpio_set(MOTOR1_DIR, 1);
        }

        if(0<=speed2_power) //电机2   正转
        {
            pwm_duty(MOTOR2_PWM, speed2_power);
            gpio_set(MOTOR2_DIR, 0);
        }
        else                //电机2   反转
        {
            pwm_duty(MOTOR2_PWM, -speed2_power);
            gpio_set(MOTOR2_DIR, 1);
        }
        systick_delay_ms(STM0, 1000);
    }
}
while (TRUE)
    {
        //systick_delay_ms(STM0, 10000);
        /////**********接受元素标志位**********/////
       if(uart_query(UART_0, &uart_buff))//0-255
```

```
        {
                        //将收到的数据，再发出去
                        lcd_showstr(0,2,"OK");
                        if(uart_buff==201)
                         {
                            gpio_set(P23_1,1);//打开蜂鸣器高电平
有效
                         }
                        else if(uart_buff>=211&&uart_buff<=213)
                         {
                                    gpio_set(P23_1,1);
                         }
                        else
if(uart_buff==221||uart_buff==223||uart_buff==226)
                                {
                                    gpio_set(P23_1,1);
                                }
                        else
if(uart_buff==231||uart_buff==233||uart_buff==236)//右环岛
                                {
                                   gpio_set(P23_1,1);
                                }
```

```
                    else
                      {
                              gpio_set(P23_1,0);
                      }
                    /***************接受图像误差
**************************************/
                    if(uart_buff>60)
                    {
                        erro=-(uart_buff-60);
                    }
                    else
                    {
                        erro=uart_buff;
                    }
                    lcd_showint8(0,3,erro);
                    lcd_showint16(0,4,Steer_out);
                  //uart_putchar(UART_0, uart_buff);

      } else
        {
lcd_showstr(0,6,"NO")
} }
```