第十七届全国大学生
智能汽车竞赛
完全模型组技术报告

学　　校：常熟理工学院

队伍名称：皮卡丘开皮卡

参赛队员：申越 姜浩 张潇 孙涛 吴佳桐

带队教师：徐健 华斯亮

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：申越 姜浩 张潇 孙涛 吴佳桐

带队教师签名：徐健 华斯亮

日　　期：2022 年 8 月 21 日

目录

引言

全国大学生智能汽车竞赛是一项以“立足培养、重在参与、鼓励探索、追求卓越”为指导思想，面向全国大学生开展的具有探索性的工程实践活动，能够进一步促进高等学校加强对学生创新精神、协作精神和工程实践能力的培养，提高学生解决实际问题的能力，充分利用面向大学生的群众性科技活动，为优秀人才的脱颖而出创造条件，不断提高人才培养质量。

本文以第十七届全国大学生智能汽车竞赛完全模型组为背景。完全模型组的基本比赛任务为：

选手以英飞凌单片机作为运动控制核心、用 EdgeBoard 识别检测赛道元素，用制作的车模完成从车库出发沿着赛道运行两周。车模需要分别通过道路设置的各种元素，识别道路中心的标志完成特殊路段通行。

比赛时间从车模驶出车库到重新回到车库为止。如果车模没有能够停止在车库内停车区内，比赛时间加罚 5 秒钟。对于未完成的任务会通过相应的加罚时间叠加在比赛时间上。

为了引导比赛任务的完成，在比赛赛道的任务元素和特殊元素区域的前方指定的区域贴有固定的地面标志，主要有泛行区标志、禁止通行标志、施工区标志、坡道标志、加油站标志以及加油站出口数字标志。主要的特殊路段包括泛行区、施工区、加油站、坡道。

本文技术报告主要包括机械系统、硬件系统、软件系统等，详尽地阐述了本组完全模型组的设计方案，具体表现在硬件电路的设计以及控制算法的部分想法。希望可以和其他学校同学进行交流，改进我们现有的技术，实现共同进步。

第一章 系统总体设计

1.1 系统概述

本系统采用百度 EdgeBoard 作为赛道信息和赛道元素检测核心，采用英飞凌 TriCore 架构的 AURIX 系列 TC377 单片机作为运动控制核心。通过摄像头收集道路和标志的信息，并且采用 EdgeBoard 进行数据处理，通过串口将信息传输给 TC377 后做出相应的动作，使用 PID 闭环控制算法来控制小车的运动。此外，通过按键、显示器等进行辅助调试。系统结构图如图 1.1。

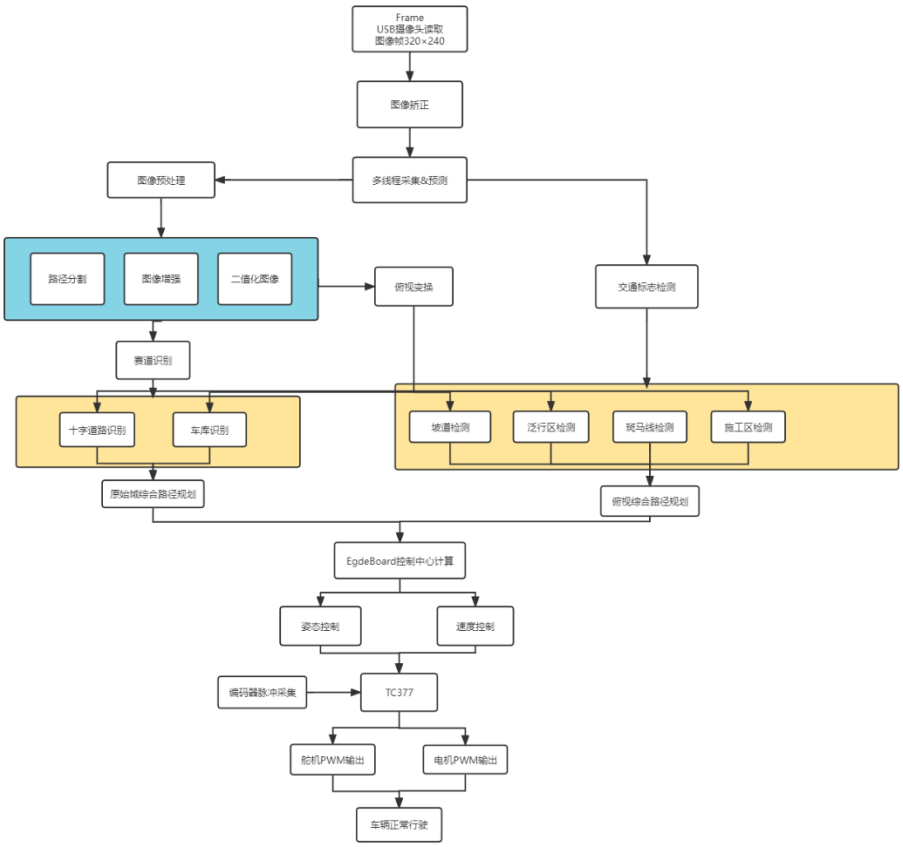

图 1.1 系统框架结构

1.2 整车布局

整车布局如图 1.2、图 1.3。此布局遵守轻量性、稳定性的设计原则，主要优点如下：

1. EdgeBoard 安装在最上层，保证散热。
2. 摄像头支撑采用空心轻质碳素杆，便于调整高度。
3. 压低电池，降低重心以提高稳定性。
4. 车壳进行简单的切割处理，便于后期调试。

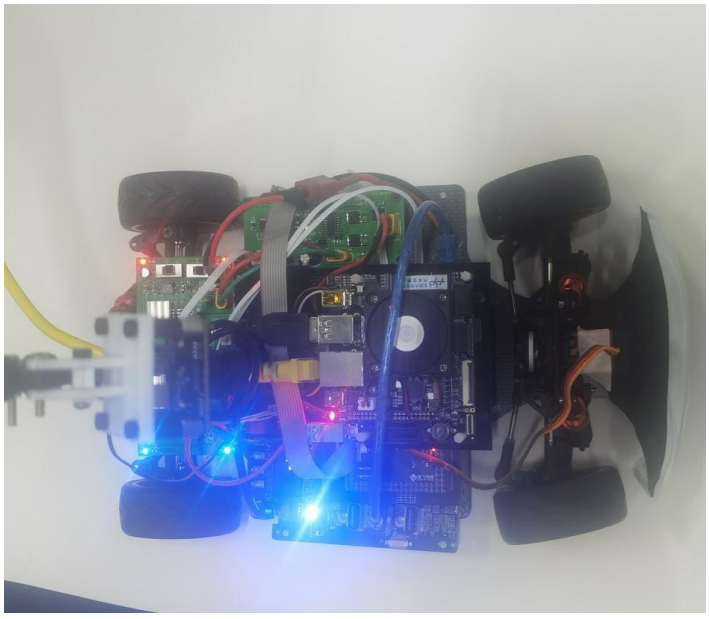

图 1.2 整车布局 1

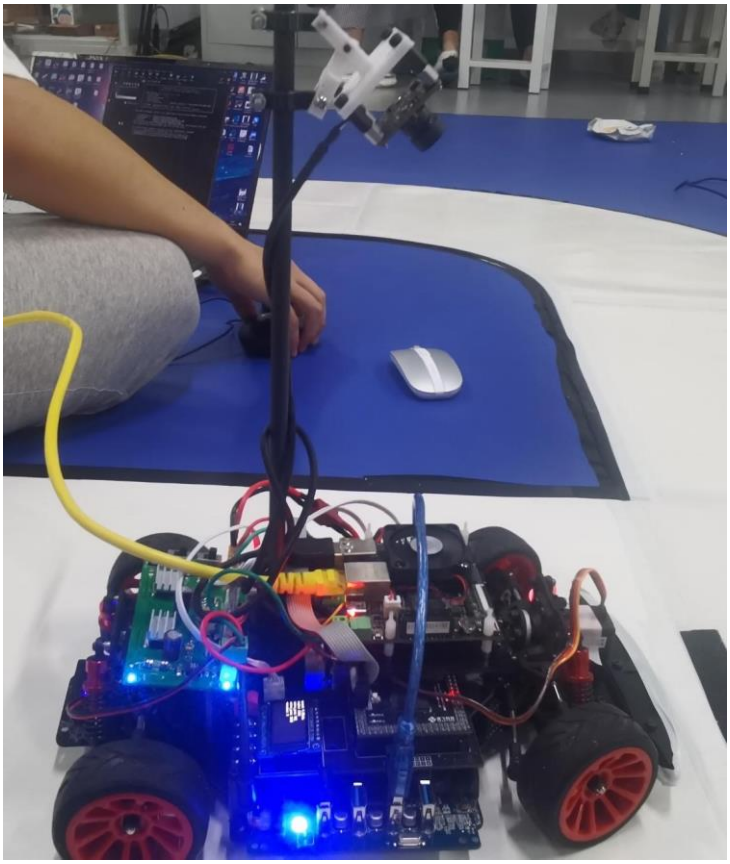

图 1.3 整车布局 2

第二章 机械系统设计及实现

机械结构会对小车的稳定运行和车模的性能都具有着重要的影响。机械结构模型车的机械机构和组装形式是整个车身的基础，结构的好坏对智能汽车的运行速度有着直接的影响。经过大量的实验测试我们得出，机械结构决定了智能汽车的上限，软件算法只有在良好的机械结构上才能够更好的提高整体性能，算法的优化可使车速不断接近最大极限。

2.1 车模的选择和设计

根据组委会的规定，本次完全模型组可以使用竞赛指定的以碳纤维为材质的 I 型车模。车壳必须完整的包裹车辆的本身，车身的底板外边缘，4 个轮子和越过车壳的传感器及其支撑件外，车辆的正视图，侧视图和俯视图看不到车辆的内部细节。车壳的限制使用光敏树脂、PLA、ABS、尼龙等塑料材质制作而成。如图 2.1、图 2.2。

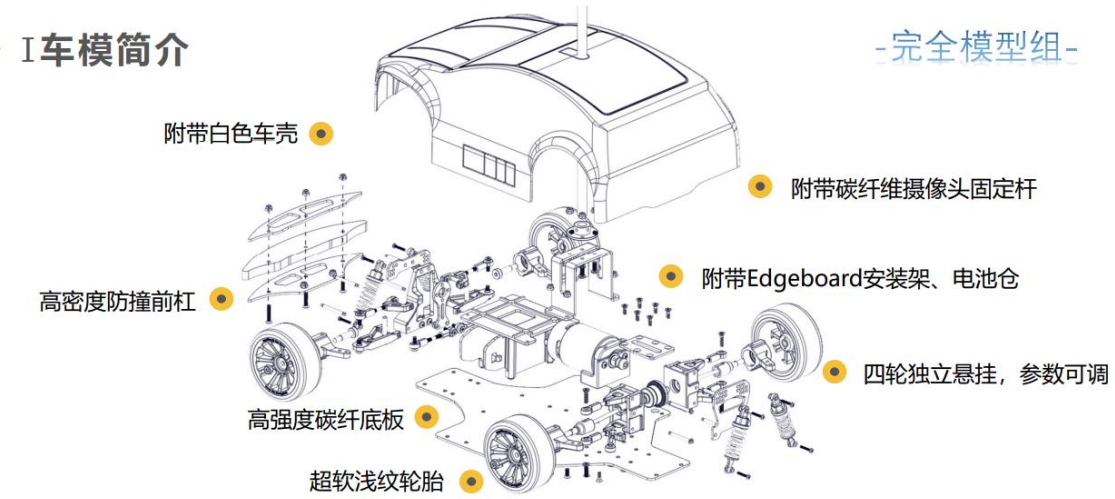

图 2.1 车模结构

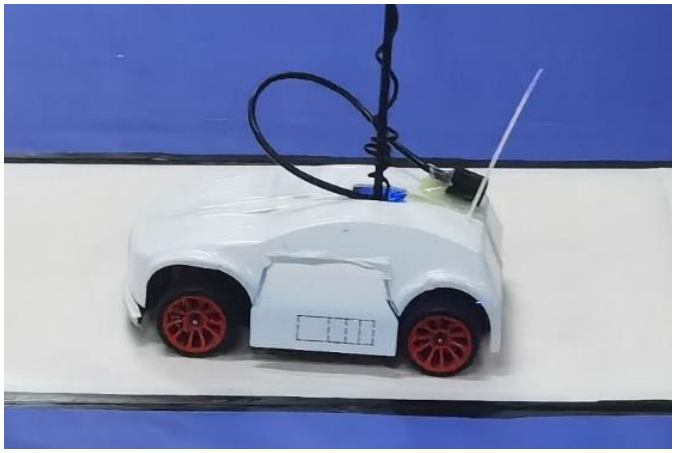

图 2.2 车壳设计与安装

2.2 其它机械结构调整与安装

2.2.1 摄像头杆的安装

为了降低整车重心，需要严格控制摄像头的安装位置和重量，我们自行设计3D 打印的零件配合摄像头进行组装，如图 2.3。这样达到了调整前瞻的目的，整套装置具有很高的定位精度和刚度，使摄像头便于拆卸和维修。

图 2.3 摄像头支架

2.3 车模重心

车模在搭建完之后，测量整个车模的质心分布，目的是保证整个重心降低。重心越低车模越稳定。我们分析整个方案后，在保证电路安全的同时，降低电路板和 EdgeBoard 的高度，同时也方便车壳的安装。

第三章 硬件系统设计及实现

3.1 硬件设计方案

电子系统的设计目标是高效、可靠、简洁。在整个系统设计过程中都严格地按照此规范执行[1]。

简洁是指在满足了前面的要求后，为了尽量减轻整车重量，同时提高检测的方便性和降低故障率。应使电路设计尽量简洁，尽量减少元器件和缩小电路板面积，使电路部分重量轻，易于安装和调试。

可靠性是系统的第一要求，需要对电路设计进行处理，做好部分的屏蔽、滤波等工作，将高速数字电路与模拟电路分开，使本系统的可靠性达到要求。

在对电路进行了分析后，根据需要对电路进行了简化，合理调整元件排列、电路走线，使本系统硬件电路部分轻量化指标都达到了设计要求[2]。

3.2 电路设计方案

电路系统由三部分组成：以 TC377 为核心的运动主控板，电机驱动模块，稳压模块。为了防止强电和弱电互相干扰，我们将电源稳压与主控版分离开，且驱动部分与主板的连接做了隔离，电机驱动与舵机控制连接采用排线连接[3]。

智能车主控板上集成了控制电路，包括以下部件：TC377、电源稳压电路、电机驱动模块接口、编码器接口、舵机接口、OLED 接口、指示灯、按键等。

3.3 电源稳压电路

电源分为开关电源和线性电源。开关模式电源（SMPS），PWM 开关电源是让功率晶体管工作在导通和关断的状态。与线性电源相比，PWM 开关电源更为有效的工作过程是通过“斩波”，即把输入的直流电压斩成幅值等于输入电压幅值的脉冲电压来实现的。

目前线性电源的技术很成熟，且制作成本较低，可以达到很高的稳定度，输出电压的纹波也很小，而且没有开关电源具有的干扰和噪音。全部硬件电路的供电电源采用格氏锂电池。由于电路中的不同电路模块所需要的工作电压和电流容量各不相同，因此电源模块包含多个稳压电路，将电池电压转换成各个模块所

需要的电压，防止干扰，保证稳定性[4]。

(1)5V 稳压 TPS5430DDAR

TPS5430 的输入范围为 5.5V~36V，可以持续输出 3A 的电流（峰值电流为 4A），转换效率可以达到 95%，输出电压初始精度为 1.5%。因此采用该芯片作为稳压芯片。

(2)3.3V 稳压

AMS1117-3.3 是一款输出电压为 3.3V 的正向低压差稳压器，适用于高效线性稳压器、开关电源稳压器等电路。符合单片机对电源要求。电源电路具体原理图如图 3.1。

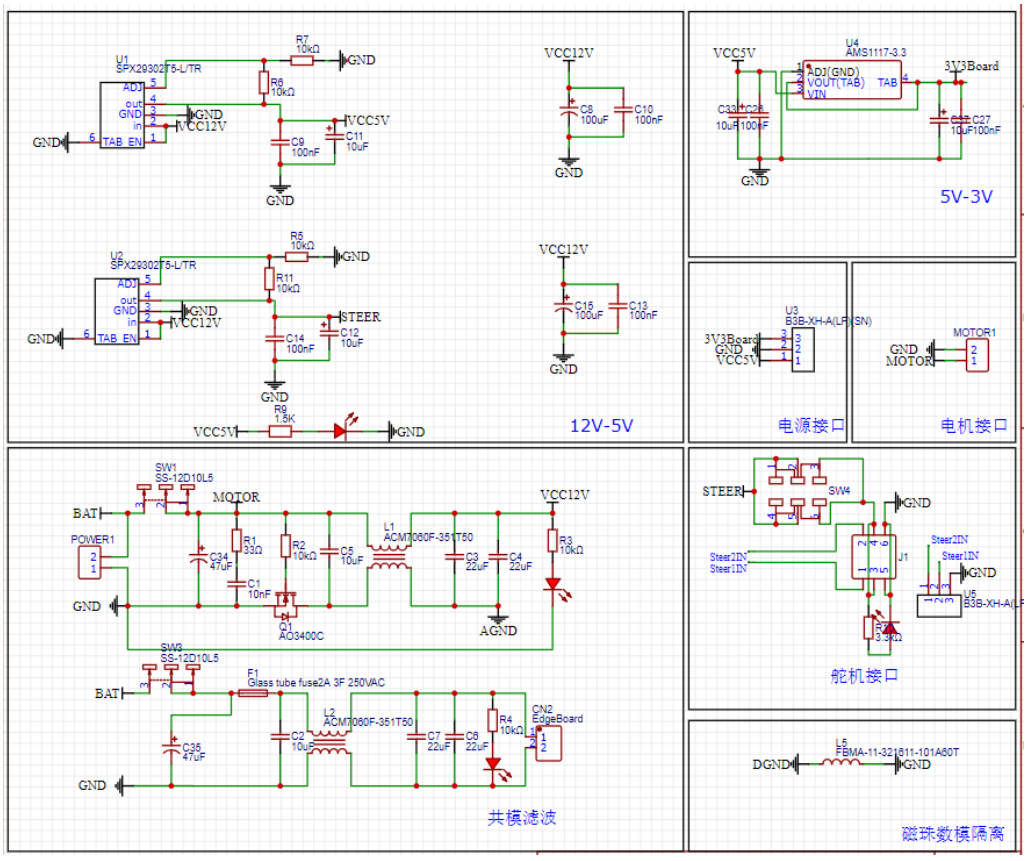

图 3.1 电源电路原理图

3.4 电机驱动模块

选择 MOSFET 时主要考虑的因素有：耐压、导通内阻和封装[2]。电源是额定电压为 11.1V 的电池，由于电机工作时可能处于再生发电状态，所以的元件耐压值最好取两倍电源电压值以上，即耐压在 22.2V 以上。并且内阻越小越好。封装越大功率越大。于是我们最终选择了 VBE1302。N 沟道漏源电压为 30V 连续漏极电流为 120A 功率为 250W。

电机驱动模块采用 HIP4082IBZT 驱动芯片电路。能通过单片机输出的 PWM 信号来快速控制电机的转速与旋转方向。如图 3.2。

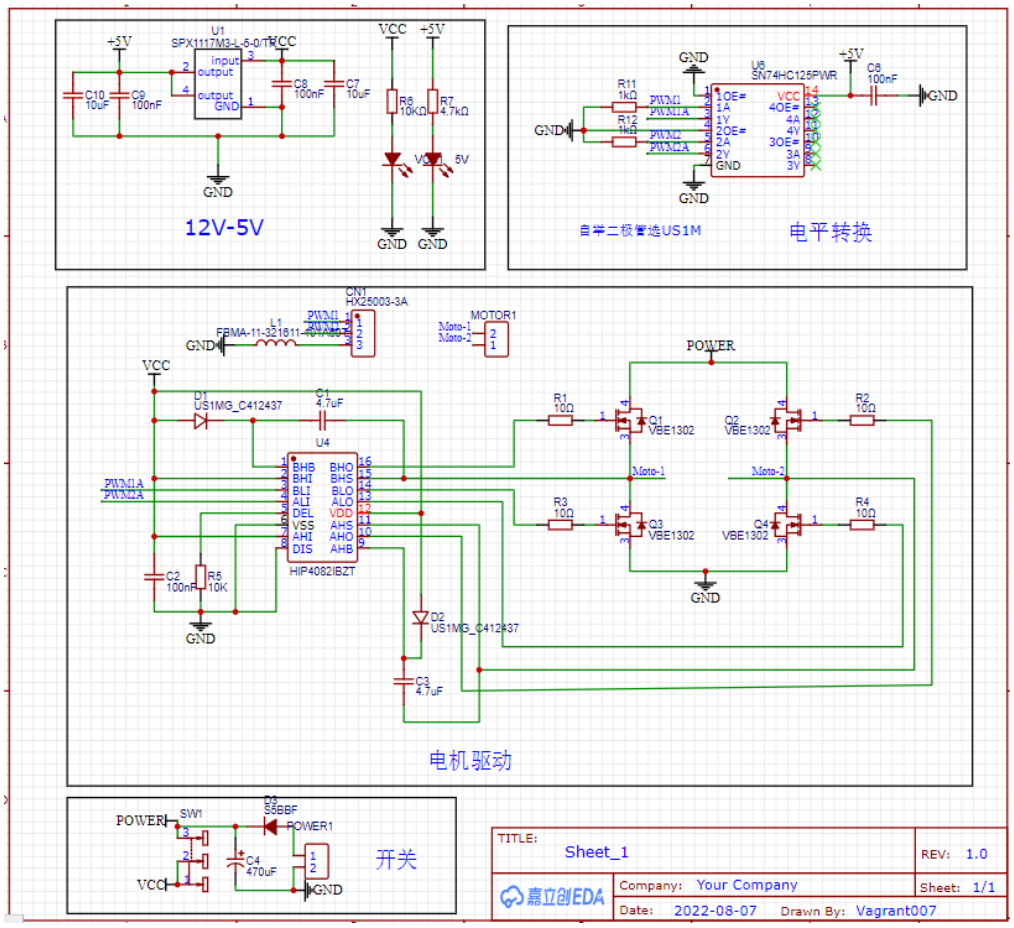

图 3.2 电机驱动电路

第四章 软件控制算法

硬件基本电路完成后，智能汽车的基础大体完成了。为了完成相应的任务拥有一套高效、省时的程序是必不可少的，其路径识别算法也是我们提高成绩的基础。我们使用 OpenCV 对赛道数据进行处理，同时加以最为经典的 PID 控制舵机，这样能够满足我们的巡线需求。对于电机控制，我们采用增量式 PID 控制算法，达到了理想的控速效果。

4.1 摄像头数据处理

根据比赛要求，在巡线之外，还需要识别标志并且执行相应的任务。由 EdgeBoard 识别通过串口和 TC377 进行通讯，便可以达到这一目的[5]。

```
if (counterRunBegin > 20) //智能车启动延时：前几场图像+AI推理不稳定
{
    //智能汽车方向控制
    motionController.pdController(controlCenterCal.controlCenter); // PD控制器姿态控制

    //智能汽车速度控制
    switch (roadType)
    {
    case RoadType::FreezoneHandle:                                          //泛行区处理速度
        if (motionController.params.FreezoneEnable)                         // AI区域的速度
            motionController.motorSpeed = motionController.params.speedFreezone; //匀速控制
        else                                                                //非AI区域的速度
            motionController.speedController(true, controlCenterCal);       //变加速控制
        break;
    case RoadType::GasstationHandle:                                    //加油站速度
        motionController.motorSpeed = motionController.params.speedGasBusy; //匀速控制
        break;
    case RoadType::BusyareaHandle:                                      //施工区速度
        motionController.motorSpeed = motionController.params.speedGasBusy; //匀速控制
        break;
    case RoadType::SlopeHandle:                                        //坡道速度
        motionController.motorSpeed = motionController.params.speedSlop; //匀速控制
        break;
    default:                                                    //基础巡线 | 十字 |环岛速度
        motionController.speedController(true, controlCenterCal); //变加速控制
        break;
    }

    if (motionController.params.debug || !motionController.params.debug) //调试模式下不控制车辆运动
    {
        driver->carControl(motionController.motorSpeed, motionController.servoPwm); //串口通信，姿态与速度控制
    }
}
```

图 4.1 赛道标志检测处理代码

4.2 车速控制

采用增量式 PID 控制速度，对赛道有良好的适应能力。增量型 PID 是对位置型 PID 取增量，这时控制器输出的是相邻两次采样时刻所计算的位置值之差，得到的结果是增量，即在上一次的控制量的基础上需要增加控制量。

```
 * @brief 姿态PD控制器
 *
 * @param controlCenter 智能车控制中心
 */
void pdController(int controlCenter)
{
    float error = controlCenter - COLSIMAGE / 2; //图像控制中心转换偏差
    static int errorLast = 0;                    //记录前一次的偏差
    if (abs(error - errorLast) > COLSIMAGE / 10)
    {
        error = error > errorLast ? errorLast + COLSIMAGE / 10 : errorLast - COLSIMAGE / 10;
    }

    turnP = abs(error) * runP2 + runP1;
    turnP = max(turnP, 0.2f);
    if (turnP < 0.2)
    {
        turnP = 0.2;
    }
    turnP = runP1 + ROWSIMAGE / 2 * runP2;

    int pwmDiff = (error * turnP) + (error - errorLast) * turnD;
    errorLast = error;

    servoPwm = (uint16_t)(PWMSERVOMID + pwmDiff); // PWM转换
}
```

```
void motor_pid(int16 s)
{
    motor_err_now = s - encoder;
    motor_output = (int16)(motor_kp * motor_err_now + motor_ki * motor_err_sum);
    motor_err_last = motor_err_now;
    motor_err_sum += motor_err_now - motor_err_last;
    if(motor_output > 1600)
        motor_output = 1600;
    if(motor_output < -1600)
        motor_output = -1600;

    motor_move(motor_output);
}
```

图 4.2 增量式 PID 代码

第五章 系统开发与调试工具

5.1 开发工具

Visual Studio Code 通过 VS Code 实现对 EdgeBoard 的程序开发与调试。

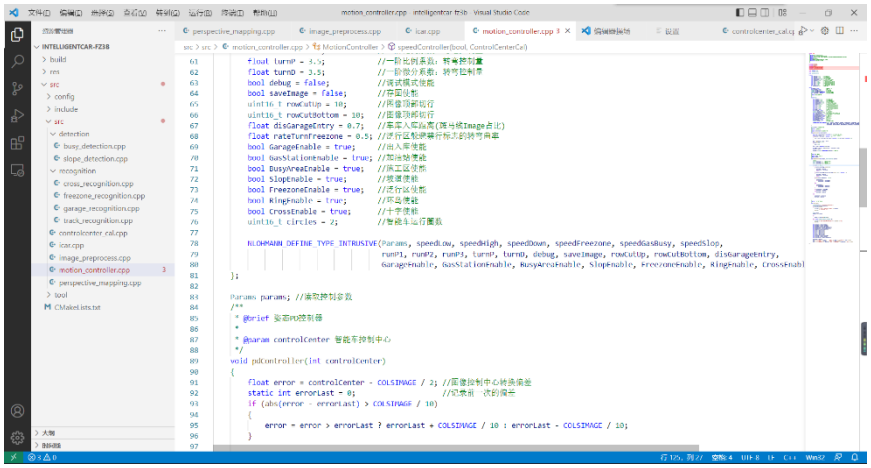

图 5.1 Visual Studio Code 开发环境

AURIX Development Studio 是一个完善可靠的芯片开发环境，支持英飞凌 AURIX 系列芯片的开发。

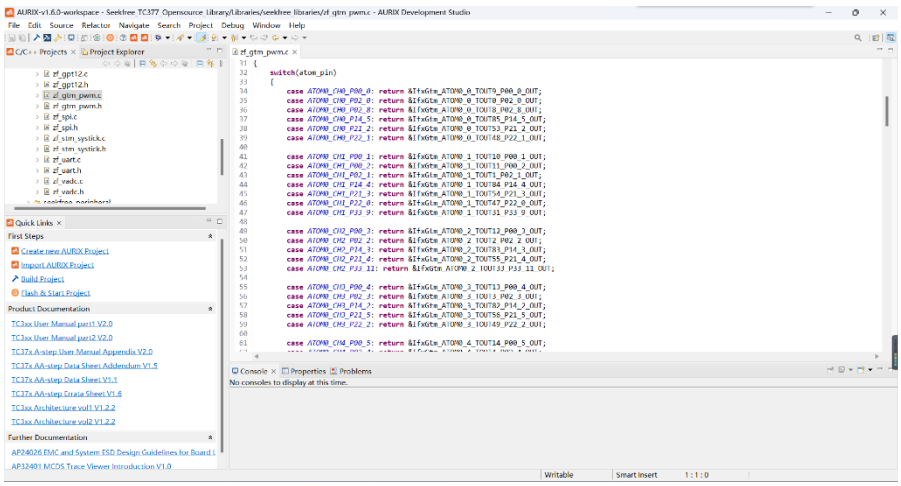

图 5.2 AURIX Development Studio 开发环境

第六章 车模的主要技术参数

车模基本参数	长	316mm
	宽	190mm
	高	398mm
	摄像头高度	332mm
微处理器型号及个数	TC377*1	
传感器种类及个数	USB 摄像头*1 500 线光电编码器*1 ICM20602*1	

结论

很荣幸有机会能参加十七届智能车竞赛。由于疫情，给我们队伍的比赛准备造成影响。不过，我们一直努力着不断克服困难，所以才有了今天的成绩。我们要感谢指导老师和实验室的同学为我们寻找场地，提供仪器，并提供了线上、线下的技术解答。

今年完全模型组属于新组别，涉及到了数字信号处理、自控原理、神经网络等内容。在一些新方面，经过学习交流与实践探索，不断的尝试，最终到达了期望。在此份技术报告中，我们主要介绍了准备比赛时的基本思路，包括机械、电路以及最重要的控制算法的思想。在机械结构方面，我们分析了整车质量分布，降低重心位置。在电路方面，我们以模块形式分类，在主板、驱动等模块分别设计，在查找资料的基础上准备了几套方案，然后我们分别实验和改进，最后确定了我们最终的电路。

当然由于是新组别，所以也有一些问题有待改进。首先车模的传动结构容易松动，有待优化；其次由于准备时间比较匆忙，车壳也没有专门准备设计，希望后面我们能够不断完善。

在大学期间，我们很庆幸能够参加这样盛大的比赛，这让我们的大学生活十分充实。

参考文献

[1]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.

[2]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.

[3]童诗白，华成英．模拟电子技术基础 [M] ．北京:高等教育出版社，2001.

[4]阎石．数字电子技术基础 [M] ．北京:高等教育出版社，2000.

[5]谭浩强著．C 程序设计．北京：清华大学出版社，2003．

附录 A

电路原理图：

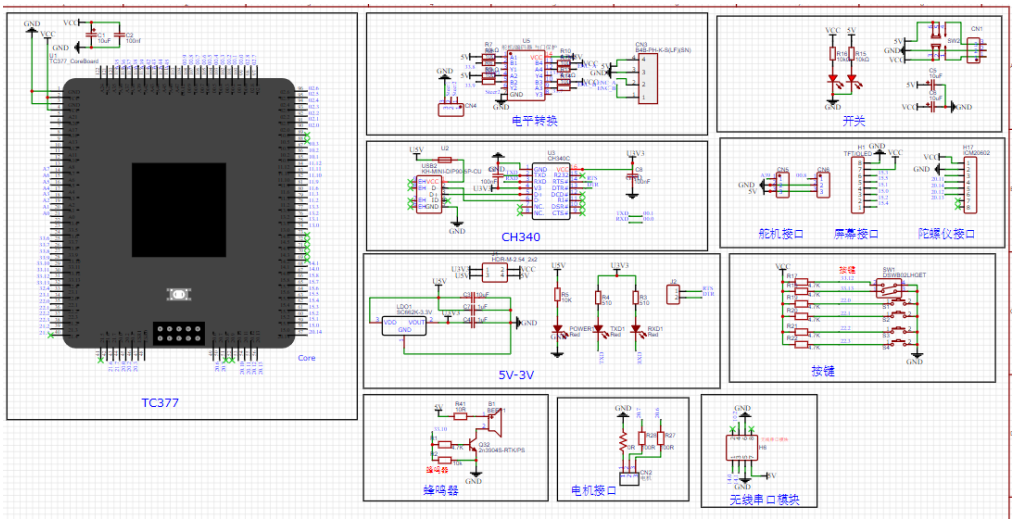

图 A-1 主控电路

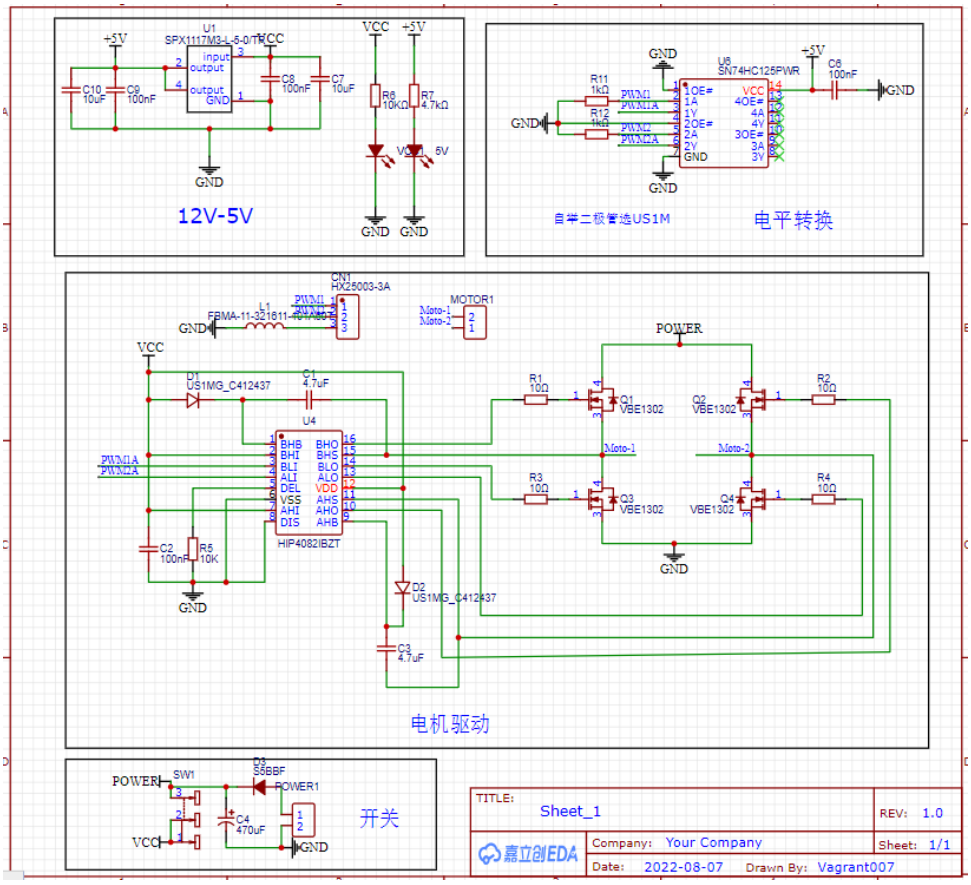

图 A-2 稳压电源电路

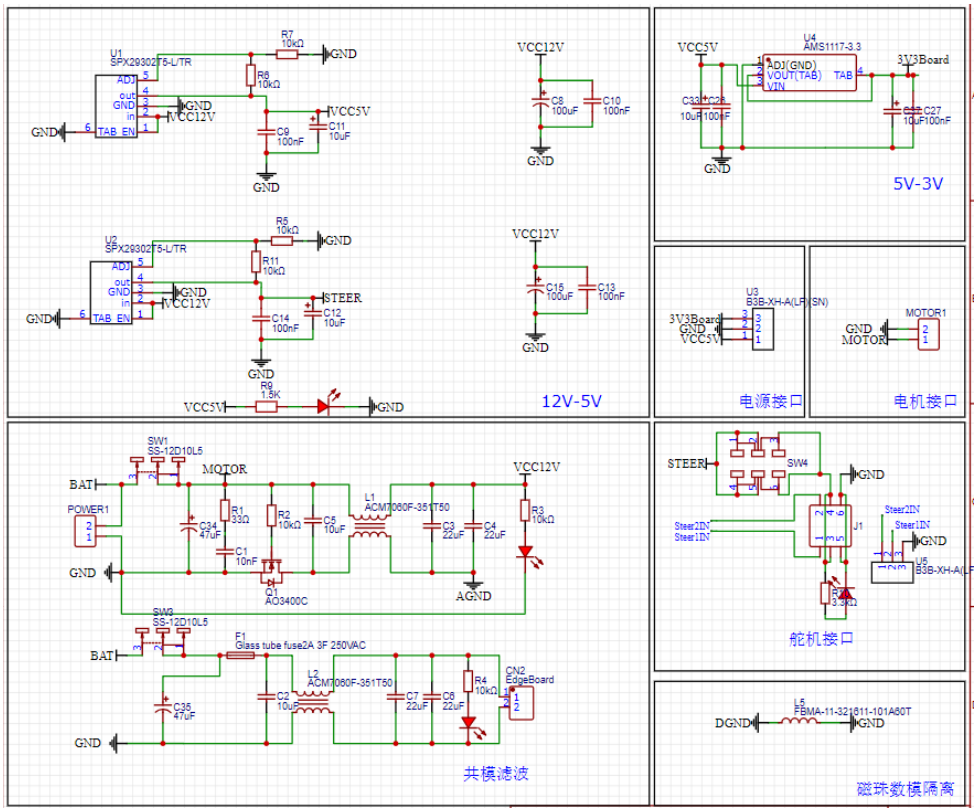

图 A-3 电机驱动电路

附录 B

程序部分源代码：

```
#include <unistd.h>
#include <iostream>
#include <signal.h>
#include <opencv2/highgui.hpp>                    //OpenCV 终端部署
#include <opencv2/opencv.hpp>                     //OpenCV 终端部署
#include "../include/uart.hpp"                    //串口通信驱动
#include "../include/detection.hpp"               //百度 Paddle 框架移动端部
署
#include "../include/common.hpp"                  //公共类方法文件
#include "image_preprocess.cpp"                   //图像预处理类
#include "recognition/track_recognition.cpp"      //赛道识别基础类
#include "controlcenter_cal.cpp"                  //控制中心计算类
#include "motion_controller.cpp"                  //智能车运动控制类
#include "recognition/cross_recognition.cpp"      //十字道路识别与路径规划类
#include "recognition/garage_recognition.cpp"     //车库及斑马线识别类
#include "recognition/freezone_recognition.cpp"   //泛行区识别类
#include "detection/busy_detection.cpp"           //施工区 AI 检测与路径规划
类
#include "detection/slope_detection.cpp"          //坡道 AI 检测与路径规划类

using namespace std;
using namespace cv;

void callbackSignal(int signum);
void displayWindowDetailInit(void);
std::shared_ptr<Driver> driver = nullptr; //初始化串口驱动

enum RoadType
{
    BaseHandle = 0,    //基础赛道处理
    RingHandle,        //环岛赛道处理
    CrossHandle,       //十字道路处理
    FreezoneHandle,    //泛行区处理
    GarageHandle,      //车库处理
    GasstationHandle, //加油站处理
    BusyareaHandle,    //施工区处理
```

```
    SlopeHandle       //坡道处理
};

int main(int argc, char const *argv[])
{
    std::shared_ptr<Detection> detection = nullptr; //初始化 AI 预测模型
    ImagePreprocess imagePreprocess;                //图像预处理类
    TrackRecognition trackRecognition;              //赛道识别
    ControlCenterCal controlCenterCal;              //车辆行驶路径拟合类
    MotionController motionController;              //运动控制
    CrossroadRecognition crossroadRecognition;      //十字道路处理
    GarageRecognition garageRecognition;            //车库识别
    FreezoneRecognition freezoneRecognition;        //泛型区识别类
    BusyareaDetection busyareaDetection;            //施工区检测
    SlopeDetection slopeDetection;                  //坡道（桥）检测类
    uint16_t counterRunBegin = 1;                   //智能车启动计数器：等
待摄像头图像帧稳定
    RoadType roadType = RoadType::BaseHandle;       //初始化赛道类型
    uint16_t counterOutTrackA = 0;                  //车辆冲出赛道计数器 A
    uint16_t counterOutTrackB = 0;                  //车辆冲出赛道计数器 B
    uint16_t circlesThis = 1;                       //智能车当前运行的圈数
    uint16_t countercircles = 0;                    //圈数计数器

    // USB 转串口的设备名为 / dev/ttyUSB0
    driver = std::make_shared<Driver>("/dev/ttyUSB0",
BaudRate::BAUD_115200);
    if (driver == nullptr)
    {
        std::cout << "Create Uart-Driver Error!" << std::endl;
        return -1;
    }
    //串口初始化，打开串口设备及配置串口数据格式
    int ret = driver->open();
    if (ret != 0)
    {
        std::cout << "Uart Open failed!" << std::endl;
        return -1;
    }
```

```
    ipm.init(Size(COLSIMAGE, ROWSIMAGE), Size(COLSIMAGEIPM,
ROWSIMAGEIPM)); // IPM 逆透视变换初始化

    signal(SIGINT, callbackSignal); //程序退出信号

    motionController.loadParams(); //读取配置文件
    trackRecognition.rowCutUp = motionController.params.rowCutUp;
    trackRecognition.rowCutBottom =
motionController.params.rowCutBottom;
    garageRecognition.disGarageEntry =
motionController.params.disGarageEntry;
    if (motionController.params.GarageEnable) //出入库使能
        roadType = RoadType::GarageHandle;    //初始赛道元素为出库

    if (motionController.params.debug) //调试模式
    {
        displayWindowDetailInit();
                                        //显示窗口初始化
        detection = Detection::DetectionInstance("/dev/video0",
"../res/model/mobilenet-ssd-v1"); // 视频输入源：本地视频 | AI 模型文件
        printAiEnable =
true;
              // 开启 AI 检测结果图像绘制：耗时
    }
    else //比赛模式
    {
        cout << "等待发车!!!" << endl;
        detection = Detection::DetectionInstance("/dev/video0",
"../res/model/mobilenet-ssd-v1"); // 视频输入源：本地视频 | AI 模型文件
        printAiEnable =
false;
// 关闭 AI 检测结果图像绘制：节省算力

        // while (!driver->receiveStartSignal()) //串口接收下位机-比赛开始
信号
        // {
        //     ;
        // }
        cout << "--------- System start!!! -------" << endl;
```

```
        for (int i = 0; i < 30; i++) // 3 秒后发车
        {
            driver->carControl(0, PWMSERVOMID); //智能车停止运动|建立下位机通信
            waitKey(100);
        }
    }

    while (1)
    {

        bool imshowRec = false; //特殊赛道图像显示标志

        // 处理帧时长监测：显示单帧时长
        if (motionController.params.debug)
        {
            static auto preTime = chrono::duration_cast<chrono::milliseconds>(chrono::system_clock::now().time_since_epoch()).count();
            auto startTime = chrono::duration_cast<chrono::milliseconds>(chrono::system_clock::now().time_since_epoch()).count();
            cout << "run frame time : " << startTime - preTime << "ms" << endl;
            preTime = startTime;
        }

        //send(0x44);

        //[01] 视频源选择
        std::shared_ptr<DetectionResult> resultAI = detection->getLastFrame(); //获取 Paddle 多线程模型预测数据
        Mat frame = resultAI->rgb_frame;                                        //获取原始摄像头图像
        if (motionController.params.debug)
        {
            // imshow("frame", resultAI->det_render_frame);
            savePicture(resultAI->det_render_frame); //保存 AI 识别图像
        }
```

```
        else
        {
            if (motionController.params.saveImage) //保存原始图像
                savePicture(frame);
        }

        //[02] 图像预处理
        // Mat imgaeCorrect =
imagePreprocess.imageCorrection(frame);          // 相机矫正
        Mat imgaeCorrect = frame;
        Mat imageBinary =
imagePreprocess.imageBinaryzation(imgaeCorrect); // Gray

        //[03] 基础赛道识别：万物基于 Track！！！！！
        trackRecognition.trackRecognition(imageBinary); //赛道线识别
        if (motionController.params.debug)
        {
            Mat imageTrack = imgaeCorrect.clone();  // RGB
            trackRecognition.drawImage(imageTrack); //图像显示赛道线识别结
果
            imshow("imageTrack", imageTrack);
            savePicture(imageTrack);
        }

        // [04] 出库和入库识别与路径规划
        if (motionController.params.GarageEnable) //赛道元素是否使能
        {
            if (roadType == RoadType::GarageHandle || roadType ==
RoadType::BaseHandle)
            {
                countercircles++; //圈数计数
                if (countercircles > 110)
                    countercircles = 110;
                if
(garageRecognition.startingCheck(resultAI->predictor_results)) //检测到
起点
                {
                    busyareaDetection.reset();   //施工区检测初始化
                    freezoneRecognition.reset(); //泛行区识别复位
```

```
                    if (countercircles > 60)
                    {
                        circlesThis++;
                        countercircles = 0;
                    }
                }

                if (circlesThis >= motionController.params.circles &&
countercircles > 100) //入库使能：跑完 N 圈
                    garageRecognition.entryEnable = true;

                if
(garageRecognition.garageRecognition(trackRecognition,
resultAI->predictor_results))
                {
                    roadType = RoadType::GarageHandle;
                    if (garageRecognition.garageStep ==
garageRecognition.GarageEntryFinish) //入库完成
                    {
                        cout << ">>>>>>>   入库结束 !!!!!" << endl;
                        callbackSignal(0);
                    }
                    if (motionController.params.debug)
                    {
                        Mat imageGarage = Mat::zeros(Size(COLSIMAGE,
ROWSIMAGE), CV_8UC3); //初始化图像
                        garageRecognition.drawImage(trackRecognition,
imageGarage);
                        imshow("imageRecognition", imageGarage);
                        imshowRec = true;
                        savePicture(imageGarage);
                    }
                }
                else
                    roadType = RoadType::BaseHandle;
            }
        }

        //[05] 施工区检测
        if (motionController.params.BusyAreaEnable) //赛道元素是否使能
```

```
        {
            if (roadType == RoadType::BusyareaHandle || roadType ==
RoadType::BaseHandle)
            {
                if
(busyareaDetection.busyareaDetection(trackRecognition,
resultAI->predictor_results))
                {
                    if (motionController.params.debug)
                    {
                        Mat imageBusyarea = Mat::zeros(Size(COLSIMAGE,
ROWSIMAGE), CV_8UC3); //初始化图像
                        busyareaDetection.drawImage(trackRecognition,
imageBusyarea);
                        imshow("imageRecognition", imageBusyarea);
                        imshowRec = true;
                        savePicture(imageBusyarea);

                        // 显示俯视域的赛道图像
                        // Mat imageIpm;
                        // ipm.homography(imageBusyarea, imageIpm); //图
像的逆透视变换
                        // imshow("imageIpm", imageIpm);
                        // savePicture(imageIpm);
                    }
                    roadType = RoadType::BusyareaHandle;
                }
                else
                    roadType = RoadType::BaseHandle;
            }
        }

        // [06] 坡道（桥）检测与路径规划
        if (motionController.params.SlopEnable) //赛道元素是否使能
        {
            if (roadType == RoadType::SlopeHandle || roadType ==
RoadType::BaseHandle)
            {
                if (slopeDetection.slopeDetection(trackRecognition,
resultAI->predictor_results))
```

```
                {
                    roadType = RoadType::SlopeHandle;
                    if (motionController.params.debug)
                    {
                        Mat imageFreezone = Mat::zeros(Size(COLSIMAGE,
ROWSIMAGE), CV_8UC3); //初始化图像
                        slopeDetection.drawImage(trackRecognition,
imageFreezone);
                        imshow("imageRecognition", imageFreezone);
                        imshowRec = true;
                        savePicture(imageFreezone);
                    }
                }
                else
                    roadType = RoadType::BaseHandle;
            }
        }

        // [07] 泛行区检测与识别：非 AI 方式
        if (roadType == RoadType::FreezoneHandle || roadType ==
RoadType::BaseHandle)
        {
            if
(freezoneRecognition.freezoneRecognition(trackRecognition,
resultAI->predictor_results))
            {
                roadType = RoadType::FreezoneHandle;
                if (motionController.params.debug)
                {
                    Mat imageFreezone = Mat::zeros(Size(COLSIMAGE,
ROWSIMAGE), CV_8UC3); //初始化图像
                    freezoneRecognition.drawImage(trackRecognition,
imageFreezone);
                    imshow("imageRecognition", imageFreezone);
                    imshowRec = true;
                    savePicture(imageFreezone);
                }
            }
            else
                roadType = RoadType::BaseHandle;
```

```
        }

        // [08] 十字道路处理
        if (motionController.params.CrossEnable) //赛道元素是否使能
        {
            if (roadType == RoadType::CrossHandle || roadType ==
RoadType::BaseHandle)
            {
                if
(crossroadRecognition.crossroadRecognition(trackRecognition))
                {
                    roadType = RoadType::CrossHandle;
                    if (motionController.params.debug)
                    {
                        Mat imageCross = Mat::zeros(Size(COLSIMAGE,
ROWSIMAGE), CV_8UC3); //初始化图像
                        crossroadRecognition.drawImage(trackRecognition,
imageCross);
                        imshow("imageRecognition", imageCross);
                        imshowRec = true;
                        savePicture(imageCross);
                    }
                }
                else
                    roadType = RoadType::BaseHandle;
            }
        }

        // [09] 控制中心计算
        if (trackRecognition.pointsEdgeLeft.size() < 30 &&
trackRecognition.pointsEdgeRight.size() < 30 && roadType !=
RoadType::FreezoneHandle && roadType != RoadType::SlopeHandle) //防止车
辆冲出赛道
        {
            counterOutTrackA++;
            counterOutTrackB = 0;
            if (counterOutTrackA > 50)
                callbackSignal(0);
        }
        else
```

```
        {
            counterOutTrackB++;
            if (counterOutTrackB > 50)
            {
                counterOutTrackA = 0;
                counterOutTrackB = 50;
            }
        }
        controlCenterCal.controlCenterCal(trackRecognition); //根据赛道
边缘信息拟合运动路径（控制中心）

        // [10] 运动控制
        if (counterRunBegin > 20) //智能车启动延时：前几场图像+AI 推理不稳
定
        {
            //智能汽车方向控制
            motionController.pdController(controlCenterCal.controlCenter
); // PD 控制器姿态控制

            //智能汽车速度控制
            switch (roadType)
            {
            case
RoadType::FreezoneHandle:                                          /
/泛行区处理速度
                if
(motionController.params.FreezoneEnable)                          //
AI 区域的速度
                    motionController.motorSpeed =
motionController.params.speedFreezone; //匀速控制
                else
              //非 AI 区域的速度
                    motionController.speedController(true,
controlCenterCal);           //变加速控制
                break;
            case
RoadType::GasstationHandle:                                       //加油
站速度
                motionController.motorSpeed =
motionController.params.speedGasBusy; //匀速控制
```

```
                break;
            case
RoadType::BusyareaHandle:                                       //施工
区速度
                motionController.motorSpeed =
motionController.params.speedGasBusy; //匀速控制
                break;
            case
RoadType::SlopeHandle:                                          //坡道速度
                motionController.motorSpeed =
motionController.params.speedSlop; //匀速控制
                break;
            default:
//基础巡线 | 十字 |环岛速度
                motionController.speedController(true,
controlCenterCal); //变加速控制
                break;
            }

            if (motionController.params.debug
|| !motionController.params.debug) //调试模式下不控制车辆运动
            {
                driver->carControl(motionController.motorSpeed,
motionController.servoPwm); //串口通信，姿态与速度控制
            }
        }
        else
            counterRunBegin++;

        // [11]调试模式下图像显示和存图
        if (motionController.params.debug)
        {
            controlCenterCal.drawImage(trackRecognition, imgaeCorrect);
            switch (roadType)
            {
            case
RoadType::BaseHandle:
                                        //基础赛道处理
```

```
                putText(imgaeCorrect, "[1] Track", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 0, 255), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::RingHandle:
                                    //环岛赛道处理
                putText(imgaeCorrect, "[1] Ring", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::CrossHandle:
                                    //十字道路处理
                putText(imgaeCorrect, "[1] Cross", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::FreezoneHandle:
                                      //泛行区处理
                putText(imgaeCorrect, "[1] Freezone", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::GarageHandle:
                                     //车库处理
                putText(imgaeCorrect, "[1] Garage", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::GasstationHandle:
                                       //加油站处理
                putText(imgaeCorrect, "[1] Gasstation", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
```

```
            case
RoadType::BusyareaHandle:
                                                //施工区处理
                putText(imgaeCorrect, "[1] Busyarea", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            case
RoadType::SlopeHandle:
                                               //坡道处理
                putText(imgaeCorrect, "[1] Slop", Point(10, 20),
cv::FONT_HERSHEY_TRIPLEX, 0.3, cv::Scalar(0, 255, 0), 1, CV_AA); //显示
赛道识别类型
                break;
            }

            putText(imgaeCorrect, "v: " +
formatDoble2String(motionController.motorSpeed, 2), Point(COLSIMAGE -
60, 80), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255), 1); //车速

            string str = to_string(circlesThis) + "/" +
to_string(motionController.params.circles);
            putText(imgaeCorrect, str, Point(COLSIMAGE - 50, ROWSIMAGE -
20), cv::FONT_HERSHEY_TRIPLEX, 0.5, cv::Scalar(0, 255, 0), 1, CV_AA);
//显示圈数
            if
(!imshowRec)
                                                    //保持调试图像存储顺序
和显示一致性
            {
                Mat imageNone = Mat::zeros(Size(COLSIMAGE, ROWSIMAGE),
CV_8UC3); //初始化图像
                imshow("imageRecognition", imageNone);
                savePicture(imageNone);
            }
            imshow("imageControl", imgaeCorrect);
            savePicture(imgaeCorrect);

            char c = waitKey(10);
        }
```

```
    }

    return 0;
}

/**
 * @brief OpenCV 图像显示窗口初始化（详细参数/Debug 模式）
 *
 */
void displayWindowDetailInit(void)
{
    //[1] 二值化图像：Gray
    string windowName = "imageTrack";
    cv::namedWindow(windowName, WINDOW_NORMAL); //图像名称
    cv::resizeWindow(windowName, 320, 240);     //分辨率
    cv::moveWindow(windowName, 10, 10);         //布局位置

    //[2] 赛道边缘图像：RGB
    windowName = "imageRecognition";
    cv::namedWindow(windowName, WINDOW_NORMAL); //图像名称
    cv::resizeWindow(windowName, 320, 240);     //分辨率
    cv::moveWindow(windowName, 10, 320);        //布局位置

    //[3] 原始图像/矫正后：RGB
    windowName = "imageControl";
    cv::namedWindow(windowName, WINDOW_NORMAL); //图像名称
    cv::resizeWindow(windowName, 640, 480);     //分辨率
    cv::moveWindow(windowName, 350, 20);        //布局位置
}

/**
 * @brief 系统信号回调函数：系统退出(按键 ctrl+c)
 *
 * @param signum 信号量
 */
void callbackSignal(int signum)
{
    driver->carControl(0, PWMSERVOMID); //智能车停止运动
    cout << "====System Exit!!!  -->  CarStopping! " << signum << endl;
    exit(signum);
```

```
    }
#include <fstream>
#include <iostream>
#include <cmath>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "../include/common.hpp"
#include "recognition/track_recognition.cpp"
using namespace cv;
using namespace std;

class ControlCenterCal
{
public:
    int controlCenter;           //智能车控制中心（0~320）
    vector<POINT> centerEdge;    //赛道中心点集
    uint16_t validRowsLeft = 0;  //边缘有效行数（左）
    uint16_t validRowsRight = 0; //边缘有效行数（右）
    double sigmaCenter = 0;      //中心点集的方差

    /**
     * @brief 控制中心计算
     *
     * @param pointsEdgeLeft 赛道左边缘点集
     * @param pointsEdgeRight 赛道右边缘点集
     */

    void controlCenterCal(TrackRecognition &track)
    {
        sigmaCenter = 0;
        controlCenter = COLSIMAGE / 2;
        centerEdge.clear();
        vector<POINT> v_center(4); //三阶贝塞尔曲线
        style = "STRIGHT";

        //边缘有效行优化：拐弯切弯重点步骤！！！！！--- 还请大家继续构建相关
方法
        // if ((track.stdevLeft < 30 && track.stdevRight > 70) ||
(track.stdevLeft > 60 && track.stdevRight < 30))
        // {
```

```
        //     validRowsCal(track.pointsEdgeLeft,
track.pointsEdgeRight); //边缘有效行计算
        //     track.pointsEdgeLeft.resize(validRowsLeft);
        //     track.pointsEdgeRight.resize(validRowsRight);
        // }

        if (track.pointsEdgeLeft.size() > 4 &&
track.pointsEdgeRight.size() > 4) //通过双边缘有效点的差来判断赛道类型
        {
            v_center[0] = {(track.pointsEdgeLeft[0].x +
track.pointsEdgeRight[0].x) / 2, (track.pointsEdgeLeft[0].y +
track.pointsEdgeRight[0].y) / 2};

            v_center[1] =
{(track.pointsEdgeLeft[track.pointsEdgeLeft.size() / 3].x +
track.pointsEdgeRight[track.pointsEdgeRight.size() / 3].x) / 2,
                            (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() / 3].y + track.pointsEdgeRight[track.pointsEdgeRight.size() /
3].y) / 2};

            v_center[2] =
{(track.pointsEdgeLeft[track.pointsEdgeLeft.size() * 2 / 3].x +
track.pointsEdgeRight[track.pointsEdgeRight.size() * 2 / 3].x) / 2,
                            (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() * 2 / 3].y + track.pointsEdgeRight[track.pointsEdgeRight.size() *
2 / 3].y) / 2};

            v_center[3] =
{(track.pointsEdgeLeft[track.pointsEdgeLeft.size() - 1].x +
track.pointsEdgeRight[track.pointsEdgeRight.size() - 1].x) / 2,
                            (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() - 1].y + track.pointsEdgeRight[track.pointsEdgeRight.size() -
1].y) / 2};

            centerEdge = Bezier(0.03, v_center);

            style = "STRIGHT";
        }
        //左单边
```

```
        else if ((track.pointsEdgeLeft.size() > 0 &&
track.pointsEdgeRight.size() <= 4) ||
                 (track.pointsEdgeLeft.size() > 0 &&
track.pointsEdgeRight.size() > 0 && track.pointsEdgeLeft[0].x -
track.pointsEdgeRight[0].x > ROWSIMAGE / 2))
        {
            style = "RIGHT";
            centerEdge = centerCompute(track.pointsEdgeLeft, 0);
        }
        //右单边
        else if ((track.pointsEdgeRight.size() > 0 &&
track.pointsEdgeLeft.size() <= 4) ||
                 (track.pointsEdgeRight.size() > 0 &&
track.pointsEdgeLeft.size() > 0 && track.pointsEdgeRight[0].x -
track.pointsEdgeLeft[0].x > ROWSIMAGE / 2))
        {
            style = "LEFT";
            centerEdge = centerCompute(track.pointsEdgeRight, 1);
        }
        else if (track.pointsEdgeLeft.size() > 4 &&
track.pointsEdgeRight.size() == 0) //左单边
        {
            v_center[0] = {track.pointsEdgeLeft[0].x,
(track.pointsEdgeLeft[0].y + COLSIMAGE - 1) / 2};

            v_center[1] =
{track.pointsEdgeLeft[track.pointsEdgeLeft.size() / 3].x,
                           (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() / 3].y + COLSIMAGE - 1) / 2};

            v_center[2] =
{track.pointsEdgeLeft[track.pointsEdgeLeft.size() * 2 / 3].x,
                           (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() * 2 / 3].y + COLSIMAGE - 1) / 2};

            v_center[3] =
{track.pointsEdgeLeft[track.pointsEdgeLeft.size() - 1].x,
                           (track.pointsEdgeLeft[track.pointsEdgeLeft.si
ze() - 1].y + COLSIMAGE - 1) / 2};
```

```
            centerEdge = Bezier(0.02, v_center);

            style = "RIGHT";
        }
        else if (track.pointsEdgeLeft.size() == 0 &&
track.pointsEdgeRight.size() > 4) //右单边
        {
            v_center[0] = {track.pointsEdgeRight[0].x,
track.pointsEdgeRight[0].y / 2};

            v_center[1] =
{track.pointsEdgeRight[track.pointsEdgeRight.size() / 3].x,
                           track.pointsEdgeRight[track.pointsEdgeRight.s
ize() / 3].y / 2};

            v_center[2] =
{track.pointsEdgeRight[track.pointsEdgeRight.size() * 2 / 3].x,
                           track.pointsEdgeRight[track.pointsEdgeRight.s
ize() * 2 / 3].y / 2};

            v_center[3] =
{track.pointsEdgeRight[track.pointsEdgeRight.size() - 1].x,
                           track.pointsEdgeRight[track.pointsEdgeRight.s
ize() - 1].y / 2};

            centerEdge = Bezier(0.02, v_center);

            style = "LEFT";
        }

        //加权控制中心计算
        int controlNum = 1;
        for (auto p : centerEdge)
        {
            if (p.x < ROWSIMAGE / 2)
            {
                controlNum += ROWSIMAGE / 2;
                controlCenter += p.y * ROWSIMAGE / 2;
            }
            else
```

```
            {
                controlNum += (ROWSIMAGE - p.x);
                controlCenter += p.y * (ROWSIMAGE - p.x);
            }
        }
        if (controlNum > 1)
        {
            controlCenter = controlCenter / controlNum;
        }

        if (controlCenter > COLSIMAGE)
            controlCenter = COLSIMAGE;
        else if (controlCenter < 0)
            controlCenter = 0;

        //控制率计算
        if (centerEdge.size() > 20)
        {
            vector<POINT> centerV;
            int filt = centerEdge.size() / 5;
            for (int i = filt; i < centerEdge.size() - filt; i++) //过滤
中心点集前后 1/5 的诱导性
            {
                centerV.push_back(centerEdge[i]);
            }
            sigmaCenter = sigma(centerV);
        }
        else
            sigmaCenter = 1000;
    }

    /**
     * @brief 显示赛道线识别结果
     *
     * @param centerImage 需要叠加显示的图像
     */
    void drawImage(TrackRecognition track, Mat &centerImage)
    {
        //赛道边缘绘制
        for (int i = 0; i < track.pointsEdgeLeft.size(); i++)
```

```
        {
            circle(centerImage, Point(track.pointsEdgeLeft[i].y,
track.pointsEdgeLeft[i].x), 1, Scalar(0, 255, 0), -1); //绿色点
        }
        for (int i = 0; i < track.pointsEdgeRight.size(); i++)
        {
            circle(centerImage, Point(track.pointsEdgeRight[i].y,
track.pointsEdgeRight[i].x), 1, Scalar(0, 255, 255), -1); //黄色点
        }

        //绘制中心点集
        for (int i = 0; i < centerEdge.size(); i++)
        {
            circle(centerImage, Point(centerEdge[i].y, centerEdge[i].x),
1, Scalar(0, 0, 255), -1);
        }

        //绘制加权控制中心：方向
        Rect rect(controlCenter, ROWSIMAGE - 20, 10, 20);
        rectangle(centerImage, rect, Scalar(0, 0, 255), CV_FILLED);

        //详细控制参数显示
        int dis = 20;
        string str;
        putText(centerImage, style, Point(COLSIMAGE - 60, dis),
FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255), 1); //赛道类型

        str = "Edge: " + formatDoble2String(track.stdevLeft, 1) + " | "
+ formatDoble2String(track.stdevRight, 1);
        putText(centerImage, str, Point(COLSIMAGE - 150, 2 * dis),
FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255), 1); //斜率：左|右

        str = "Center: " + formatDoble2String(sigmaCenter, 2);
        putText(centerImage, str, Point(COLSIMAGE - 120, 3 * dis),
FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255), 1); //中心点方差
    }

private:
    string style = ""; //赛道类型
    /**
```

```
 * @brief 搜索十字赛道突变行（左下）
 *
 * @param pointsEdgeLeft
 * @return uint16_t
 */
uint16_t searchBreakLeftDown(vector<POINT> pointsEdgeLeft)
{
    uint16_t counter = 0;

    for (int i = 0; i < pointsEdgeLeft.size() - 10; i++)
    {
        if (pointsEdgeLeft[i].y >= 2)
        {
            counter++;
            if (counter > 3)
            {
                return i - 2;
            }
        }
        else
            counter = 0;
    }

    return 0;
}

/**
 * @brief 搜索十字赛道突变行（右下）
 *
 * @param pointsEdgeRight
 * @return uint16_t
 */
uint16_t searchBreakRightDown(vector<POINT> pointsEdgeRight)
{
    uint16_t counter = 0;

    for (int i = 0; i < pointsEdgeRight.size() - 10; i++) //寻找左边
跳变点
    {
        if (pointsEdgeRight[i].y < COLSIMAGE - 2)
```

```
        {
            counter++;
            if (counter > 3)
            {
                return i - 2;
            }
        }
        else
            counter = 0;
    }

    return 0;
}

/**
 * @brief 赛道中心点计算：单边控制
 *
 * @param pointsEdge 赛道边缘点集
 * @param side 单边类型：左边 0/右边 1
 * @return vector<POINT>
 */
vector<POINT> centerCompute(vector<POINT> pointsEdge, int side)
{
    int step = 4;                    //间隔尺度
    int offsetWidth = COLSIMAGE / 2; //首行偏移量
    int offsetHeight = 0;            //纵向偏移量

    vector<POINT> center; //控制中心集合

    if (side == 0) //左边缘
    {
        uint16_t counter = 0, rowStart = 0;
        for (int i = 0; i < pointsEdge.size(); i++) //删除底部无效行
        {
            if (pointsEdge[i].y > 1)
            {
                counter++;
                if (counter > 2)
                {
                    rowStart = i - 2;
```

```
                    break;
                }
            }
            else
                counter = 0;
        }

        offsetHeight = pointsEdge[rowStart].x - pointsEdge[0].x;
        counter = 0;
        for (int i = rowStart; i < pointsEdge.size(); i += step)
        {
            int py = pointsEdge[i].y + offsetWidth;
            if (py > COLSIMAGE - 1)
            {
                counter++;
                if (counter > 2)
                    break;
            }
            else
            {
                counter = 0;
                center.emplace_back(pointsEdge[i].x - offsetHeight,
py);
            }
        }
    }
    else if (side == 1) //右边沿
    {
        uint16_t counter = 0, rowStart = 0;
        for (int i = 0; i < pointsEdge.size(); i++) //删除底部无效行
        {
            if (pointsEdge[i].y < COLSIMAGE - 1)
            {
                counter++;
                if (counter > 2)
                {
                    rowStart = i - 2;
                    break;
                }
            }
```

```
                else
                    counter = 0;
            }

            offsetHeight = pointsEdge[rowStart].x - pointsEdge[0].x;
            counter = 0;
            for (int i = rowStart; i < pointsEdge.size(); i += step)
            {
                int py = pointsEdge[i].y - offsetWidth;
                if (py < 1)
                {
                    counter++;
                    if (counter > 2)
                        break;
                }
                else
                {
                    counter = 0;
                    center.emplace_back(pointsEdge[i].x - offsetHeight,
py);
                }
            }
        }

        return center;
        // return Bezier(0.2,center);
    }
    };
#include <fstream>
#include <iostream>
#include <cmath>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "../include/common.hpp"

using namespace cv;
using namespace std;

/**
**[1] 读取视频
```

```
**[2] 图像二值化
*/

class ImagePreprocess
{
public:
    /**
     * @brief 图像二值化
     *
     * @param frame 输入原始帧
     * @return Mat  二值化图像
     */
    Mat imageBinaryzation(Mat &frame)
    {
        Mat imageGray, imageBinary;

        cvtColor(frame, imageGray, COLOR_BGR2GRAY); // RGB 转灰度图

        threshold(imageGray, imageBinary, 0, 255, THRESH_OTSU); // OTSU
二值化方法

        return imageBinary;
    }

private:
    bool correctionEnable = true; //图像矫正使能：初始化完成
    Mat cameraMatrix;             // 摄像机内参矩阵
    Mat distCoeffs;               // 相机的畸变矩阵
};

#include <fstream>
#include <iostream>
#include <cmath>
#include "../include/common.hpp"
#include "../include/json.hpp"
#include "controlcenter_cal.cpp"

using namespace std;
using nlohmann::json;
```

```
class MotionController
{
private:
    float speedLow = 1.0;       //智能车最低速
    float speedHigh = 1.0;      //智能车最高速
    float speedDown = 0.5;      //特殊区域降速速度
    float speedFreezone = 0.8; //泛行区行驶速度
    float speedGasBusy = 1.0;  //加油站|施工区行驶速度
    float speedSlop = 1.0;      //坡道（桥）行驶速度
    float runP1 = 0.9f;         //一阶比例系数：直线控制量
    float runP2 = 0.018f;       //二阶比例系数：弯道控制量
    float runP3 = 0.013f;       //三阶比例系数：弯道控制量
    float turnP = 3.5f;         //一阶比例系数：转弯控制量
    float turnD = 3.5f;         //一阶微分系数：转弯控制量
    int counterShift = 0;       //变速计数器

public:
    bool debug = false;              //调试模式使能
    bool saveImage = false;          //存图使能
    uint16_t rowCutUp = 10;          //图像顶部切行
    uint16_t rowCutBottom = 10;      //图像顶部切行
    uint16_t servoPwm = PWMSERVOMID; //发送给舵机的 PWM
    float motorSpeed = speedLow;     //发送给电机的速度
    /**
     * @brief 控制器核心参数
     *
     */
    struct Params
    {
        float speedLow = 1.0;         //智能车最低速
        float speedHigh = 1.0;        //智能车最高速
        float speedDown = 0.5;        //特殊区域降速速度
        float speedFreezone = 0.8;    //泛行区行驶速度
        float speedGasBusy = 1.0;     //加油站|施工区行驶速度
        float speedSlop = 1.0;        //坡道（桥）行驶速度
        float runP1 = 0.9;            //一阶比例系数：直线控制量
        float runP2 = 0.018;          //二阶比例系数：弯道控制量
        float runP3 = 0.0;            //三阶比例系数：弯道控制量
        float turnP = 3.5;            //一阶比例系数：转弯控制量
        float turnD = 3.5;            //一阶微分系数：转弯控制量
```

```
        bool debug = false;           //调试模式使能
        bool saveImage = false;       //存图使能
        uint16_t rowCutUp = 10;       //图像顶部切行
        uint16_t rowCutBottom = 10;   //图像顶部切行
        float disGarageEntry = 0.7;   //车库入库距离(斑马线 Image 占比)
        float rateTurnFreezone = 0.5; //泛行区躲避禁行标志的转弯曲率
        bool GarageEnable = true;     //出入库使能
        bool GasStationEnable = true; //加油站使能
        bool BusyAreaEnable = true;   //施工区使能
        bool SlopEnable = true;       //坡道使能
        bool FreezoneEnable = true;   //泛行区使能
        bool RingEnable = true;       //环岛使能
        bool CrossEnable = true;      //十字使能
        uint16_t circles = 2;         //智能车运行圈数

        NLOHMANN_DEFINE_TYPE_INTRUSIVE(Params, speedLow, speedHigh,
speedDown, speedFreezone, speedGasBusy, speedSlop,
                                       runP1, runP2, runP3, turnP, turnD,
debug, saveImage, rowCutUp, rowCutBottom, disGarageEntry,
                                       GarageEnable, GasStationEnable,
BusyAreaEnable, SlopEnable, FreezoneEnable, RingEnable, CrossEnable,
circles); //添加构造函数
    };

    Params params; //读取控制参数
    /**
     * @brief 姿态 PD 控制器
     *
     * @param controlCenter 智能车控制中心
     */
    void pdController(int controlCenter)
    {
        float error = controlCenter - COLSIMAGE / 2; //图像控制中心转换偏
差
        static int errorLast = 0;                    //记录前一次的偏差
        if (abs(error - errorLast) > COLSIMAGE / 10)
        {
            error = error > errorLast ? errorLast + COLSIMAGE / 10 :
errorLast - COLSIMAGE / 10;
        }
```

```
        turnP = abs(error) * runP2 + runP1;
        turnP = max(turnP, 0.2f);
        if (turnP < 0.2)
        {
            turnP = 0.2;
        }
        turnP = runP1 + ROWSIMAGE / 2 * runP2;

        int pwmDiff = (error * turnP) + (error - errorLast) * turnD;
        errorLast = error;

        servoPwm = (uint16_t)(PWMSERVOMID + pwmDiff); // PWM 转换
    }

    /**
     * @brief 变加速控制
     *
     * @param enable 加速使能
     * @param control
     */
    void speedController(bool enable, ControlCenterCal control)
    {
        //控制率
        uint8_t controlLow = 0;   //速度控制下限
        uint8_t controlMid = 5;   //控制率
        uint8_t controlHigh = 10; //速度控制上限

        if (enable) //加速使能
        {
            if (control.centerEdge.size() < 10)
            {
                motorSpeed = speedLow;
                counterShift = controlLow;
                return;
            }
            if (control.centerEdge[control.centerEdge.size() - 1].x >
ROWSIMAGE / 2)
            {
                motorSpeed = speedLow;
```

```
            counterShift = controlLow;
            return;
        }
        if (abs(control.sigmaCenter) < 100.0)
        {
            counterShift++;
            if (counterShift > controlHigh)
                counterShift = controlHigh;
        }
        else
        {
            counterShift--;
            if (counterShift < controlLow)
                counterShift = controlLow;
        }

        if (counterShift > controlMid)
            motorSpeed = speedHigh;
        else
            motorSpeed = speedLow;
    }
    else
    {
        counterShift = controlLow;
        motorSpeed = speedLow;
    }
}

/**
 * @brief 加载配置参数 Json
 */
void loadParams()
{
    string jsonPath = "../src/config/motion.json";
    std::ifstream config_is(jsonPath);
    if (!config_is.good())
    {
        std::cout << "Error: Params file path:[" << jsonPath
                  << "] not find .\n";
        exit(-1);
```

```
        }

        json js_value;
        config_is >> js_value;

        try
        {
            params = js_value.get<Params>();
        }
        catch (const nlohmann::detail::exception &e)
        {
            std::cerr << "Json Params Parse failed :" << e.what() <<
'\n';
            exit(-1);
        }

        this->speedLow = params.speedLow;
        this->speedHigh = params.speedHigh;
        this->speedDown = params.speedDown;
        this->speedFreezone = params.speedFreezone;
        this->speedGasBusy = params.speedGasBusy;
        this->speedSlop = params.speedSlop;
        this->runP1 = params.runP1;
        this->runP2 = params.runP2;
        this->runP3 = params.runP3;
        this->turnP = params.turnP;
        this->turnD = params.turnD;
        this->debug = params.debug;
        this->saveImage = params.saveImage;
        this->rowCutUp = params.rowCutUp;
        this->rowCutBottom = params.rowCutBottom;

        motorSpeed = speedLow;
        cout << "--- runP1:" << runP1 << " | runP2:" << runP2 << " |
runP3:" << runP3 << endl;
        cout << "--- turnP:" << turnP << " | turnD:" << turnD << endl;
        cout << "--- speedLow:" << speedLow << "m/s  |  speedHigh:" <<
speedHigh << "m/s" << endl;
    }
};
```

```cpp
#include <iostream>
#include <stdio.h>
#include <ctime>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"

using namespace cv;
using namespace std;

class PerspectiveMapping
{
public:
    /**
     * @brief IPM 初始化
     *
     * @param origSize 输入原始图像 Size
     * @param dstSize 输出图像 Size
     */
    void init(const cv::Size &origSize, const cv::Size &dstSize)
    {
        // 原始域：分辨率 320x240
        // The 4-points at the input image
        m_origPoints.clear();
        // [第二版无带畸变镜头参数]
        m_origPoints.push_back(Point2f(0, 214));   //左下
        m_origPoints.push_back(Point2f(319, 214)); //右下
        m_origPoints.push_back(Point2f(192, 0));   //右上
        m_origPoints.push_back(Point2f(128, 0));   //左上

        // 矫正域：分辨率 320x240
        // The 4-points correspondences in the destination image
        m_dstPoints.clear();
        m_dstPoints.push_back(Point2f(100, 400)); //左下
        m_dstPoints.push_back(Point2f(220, 400)); //右下
        m_dstPoints.push_back(Point2f(220, 0));   //右上
        m_dstPoints.push_back(Point2f(100, 0));   //左上

        m_origSize = origSize;
```

```
        m_dstSize = dstSize;
        assert(m_origPoints.size() == 4 && m_dstPoints.size() == 4 &&
"Orig. points and Dst. points must vectors of 4 points");
        m_H = getPerspectiveTransform(m_origPoints, m_dstPoints); //计算
变换矩阵 [3x3]
        m_H_inv = m_H.inv();                                        //求解逆
转换矩阵

        createMaps();
    }

    /**
     * @brief 单应性反透视变换
     *
     * @param _inputImg 原始域图像
     * @param _dstImg 矫正域图像
     * @param _borderMode 矫正模式
     */
    void homographyInv(const Mat &_inputImg, Mat &_dstImg, int
_borderMode)
    {
        // Generate IPM image from src
        remap(_inputImg, _dstImg, m_mapX, m_mapY, INTER_LINEAR,
_borderMode); //, BORDER_CONSTANT, Scalar(0,0,0,0));
    }

    /**
     * @brief 单应性反透视变换
     *
     * @param _point 矫正域坐标
     * @return Point2d 原始域坐标
     */
    Point2d homographyInv(const Point2d &_point)
    {
        return homography(_point, m_H_inv);
    }

    /**
     * @brief 单应性反透视变换
     *
```

```
     * @param _point 矫正域坐标
     * @return Point3d 原始域坐标
     */
    Point3d homographyInv(const Point3d &_point)
    {
        return homography(_point, m_H_inv);
    }

    /**
     * @brief 单应性透视变换
     *
     * @param _point 原始域坐标
     * @return Point2d 矫正域坐标
     */
    Point2d homography(const Point2d &_point)
    {
        return homography(_point, m_H);
    }

    /**
     * @brief 单应性透视变换
     *
     * @param _point 原始域坐标
     * @param _H 转换矩阵
     * @return Point2d 矫正域坐标
     */
    Point2d homography(const Point2d &_point, const Mat &_H)
    {
        Point2d ret = Point2d(-1, -1);

        const double u = _H.at<double>(0, 0) * _point.x +
_H.at<double>(0, 1) * _point.y + _H.at<double>(0, 2);
        const double v = _H.at<double>(1, 0) * _point.x +
_H.at<double>(1, 1) * _point.y + _H.at<double>(1, 2);
        const double s = _H.at<double>(2, 0) * _point.x +
_H.at<double>(2, 1) * _point.y + _H.at<double>(2, 2);
        if (s != 0)
        {
            ret.x = (u / s);
            ret.y = (v / s);
```

```
        }
        return ret;
    }

    /**
     * @brief 单应性透视变换
     *
     * @param _point 原始域坐标
     * @return Point3d 矫正域坐标
     */
    Point3d homography(const Point3d &_point)
    {
        return homography(_point, m_H);
    }

    /**
     * @brief 单应性透视变换
     *
     * @param _point 原始域坐标
     * @param _H 转换矩阵
     * @return Point3d
     */
    Point3d homography(const Point3d &_point, const cv::Mat &_H)
    {
        Point3d ret = Point3d(-1, -1, 1);

        const double u = _H.at<double>(0, 0) * _point.x +
_H.at<double>(0, 1) * _point.y + _H.at<double>(0, 2) * _point.z;
        const double v = _H.at<double>(1, 0) * _point.x +
_H.at<double>(1, 1) * _point.y + _H.at<double>(1, 2) * _point.z;
        const double s = _H.at<double>(2, 0) * _point.x +
_H.at<double>(2, 1) * _point.y + _H.at<double>(2, 2) * _point.z;
        if (s != 0)
        {
            ret.x = (u / s);
            ret.y = (v / s);
        }
        else
            ret.z = 0;
        return ret;
```

```
    }

    /**
     * @brief 单应性透视变换
     *
     * @param _inputImg 原始域图像
     * @param _dstImg 矫正域图像
     */
    void homography(const Mat &_inputImg, Mat &_dstImg)
    {
        // Generate IPM image from src
        remap(_inputImg, _dstImg, m_mapX, m_mapY, CV_INTER_LINEAR); //,
BORDER_CONSTANT, Scalar(0,0,0,0));
    }

    cv::Mat getH() const { return m_H; }
    cv::Mat getHinv() const { return m_H_inv; }
    void getPoints(vector<Point2f> &_origPts, vector<Point2f> &_ipmPts)
    {
        _origPts = m_origPoints;
        _ipmPts = m_dstPoints;
    }

    /**
     * @brief 绘制掩膜外框
     *
     * @param _points
     * @param _img
     */
    void drawBorder(const std::vector<cv::Point2f> &_points, cv::Mat
&_img) const
    {
        assert(_points.size() == 4);

        line(_img, Point(static_cast<int>(_points[0].x),
static_cast<int>(_points[0].y)), Point(static_cast<int>(_points[3].x),
static_cast<int>(_points[3].y)), CV_RGB(205, 205, 0), 2);
        line(_img, Point(static_cast<int>(_points[2].x),
static_cast<int>(_points[2].y)), Point(static_cast<int>(_points[3].x),
static_cast<int>(_points[3].y)), CV_RGB(205, 205, 0), 2);
```

```
        line(_img, Point(static_cast<int>(_points[0].x),
static_cast<int>(_points[0].y)), Point(static_cast<int>(_points[1].x),
static_cast<int>(_points[1].y)), CV_RGB(205, 205, 0), 2);
        line(_img, Point(static_cast<int>(_points[2].x),
static_cast<int>(_points[2].y)), Point(static_cast<int>(_points[1].x),
static_cast<int>(_points[1].y)), CV_RGB(205, 205, 0), 2);

        for (size_t i = 0; i < _points.size(); i++)
        {
            circle(_img, Point(static_cast<int>(_points[i].x),
static_cast<int>(_points[i].y)), 2, CV_RGB(238, 238, 0), -1);
            circle(_img, Point(static_cast<int>(_points[i].x),
static_cast<int>(_points[i].y)), 5, CV_RGB(255, 255, 255), 2);
        }
    }

private:
    // Sizes
    cv::Size m_origSize;
    cv::Size m_dstSize;

    // Points
    std::vector<cv::Point2f> m_origPoints;
    std::vector<cv::Point2f> m_dstPoints;

    // Homography
    cv::Mat m_H;
    cv::Mat m_H_inv;

    // Maps
    cv::Mat m_mapX, m_mapY;
    cv::Mat m_invMapX, m_invMapY;

    void createMaps()
    {
        // Create remap images
        m_mapX.create(m_dstSize, CV_32F);
        m_mapY.create(m_dstSize, CV_32F);
        for (int j = 0; j < m_dstSize.height; ++j)
        {
```

```
            float *ptRowX = m_mapX.ptr<float>(j);
            float *ptRowY = m_mapY.ptr<float>(j);
            for (int i = 0; i < m_dstSize.width; ++i)
            {
                Point2f pt = homography(Point2f(static_cast<float>(i),
static_cast<float>(j)), m_H_inv);
                ptRowX[i] = pt.x;
                ptRowY[i] = pt.y;
            }
        }

        m_invMapX.create(m_origSize, CV_32F);
        m_invMapY.create(m_origSize, CV_32F);

        for (int j = 0; j < m_origSize.height; ++j)
        {
            float *ptRowX = m_invMapX.ptr<float>(j);
            float *ptRowY = m_invMapY.ptr<float>(j);
            for (int i = 0; i < m_origSize.width; ++i)
            {
                Point2f pt = homography(Point2f(static_cast<float>(i),
static_cast<float>(j)), m_H);
                ptRowX[i] = pt.x;
                ptRowY[i] = pt.y;
            }
        }
    }
};

#include "headfile.h"
#include "stdio.h"
#include "isr.h"
#include "ui.h"
#include "servo.h"
#include "motor.h"

#pragma section all "cpu1_dsram"
//将本语句与#pragma section all restore语句之间的全局变量都放在CPU1的RAM中

uint8 str[20] = {0};
```

```c
float speed = 0;
uint16 angle = 0;
uint16 servo_pwm = 900;
int16 motor_pwm = 0;

typedef union
{
    uint8 U8_Buff[4];
    float Float;
} UNION_BIT32;

typedef union
{
    uint8 U8_Buff[2];
    uint16 U16;
} UNION_BIT16;

UNION_BIT32 UnionBit32;
UNION_BIT16 UnionBit16;

void core1_main(void)
{
    disableInterrupts();
    IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
    //用户在此处调用各种初始化函数等

    uart_init(UART_3, 115200, UART3_TX_P00_0, UART3_RX_P00_1);
    gpio_init(P20_9, GPO, 1, PUSHPULL);

    steer_init();

    ips114_init();
    ips114_clear(IPS114_BGCOLOR);

    motor_init();
    gpt12_init(GPT12_T2,GPT12_T2INB_P33_7 ,GPT12_T2EUDB_P33_6);

    //等待所有核心初始化完毕
    IfxCpu_emitEvent(&g_cpuSyncEvent);
    IfxCpu_waitEvent(&g_cpuSyncEvent, 0xFFFF);
    enableInterrupts();
```

```
    while (TRUE)
    {
        //用户在此处编写任务代码
        if (data_flag)
        {
            gpio_set(P20_9,1);
            UnionBit32.U8_Buff[0] = data[1];
            UnionBit32.U8_Buff[1] = data[2];
            UnionBit32.U8_Buff[2] = data[3];
            UnionBit32.U8_Buff[3] = data[4];
            speed = UnionBit32.Float; //电机速度

            UnionBit16.U8_Buff[0] = data[5];
            UnionBit16.U8_Buff[1] = data[6];
            angle = UnionBit16.U16; //舵机方向

            servo_pwm = (900 - 800 * ((angle - 1500) / 1000.0));

            motor_pwm=(int16)(300 * speed);

            motor_pid(motor_pwm);

            if(speed==0)
            {
                gpio_set(P20_9,0);
                gtm_pwm_init(MOTOR_PWM1, 12000, 5);
                gtm_pwm_init(MOTOR_PWM2, 12000, 0);
            }
        }
        else
        {

        }

        encoder = gpt12_get(GPT12_T2);
        gpt12_clear(GPT12_T2);
        display_menu();
        servo_pd();
    }
}
```

```c
#pragma section all restore

/*
 * servo.c
 *
 *  Created on: 2022年8月11日
 *      Author: 94380
 */

#include "servo.h"

void steer_init(void)
{

    gtm_pwm_init(S_MOTOR_PIN, 50, 900); // 750-900-1050 //500~2500
}

float p = 5;
float d = 6;
int err_now = 0;
int err_lst = 0;
uint16 output = 0;

void servo_pd(void)
{
    err_now = servo_pwm - 900;
    err_now = err_now * err_now * err_now * err_now / (150 * 150 * 150);
    output = 900 + p * err_now + err_lst * d;
    err_lst = err_now;
    if (servo_pwm < 750)
        servo_pwm = 750;
    else if (servo_pwm > 1050)
        servo_pwm = 1050;
    gtm_pwm_init(S_MOTOR_PIN, 50, servo_pwm);
}

/*
 * steer.c
 *
```

```
 *  Created on: 2022年8月4日
 *      Author: 94380
 */



#include "steer.h"
#include "key.h"

uint8 steer_enable=0;        //舵机使能
uint16 steer_duty=0;         //舵机变化量

//计算舵机位置舵机位置    （0.5ms - 2.5ms）ms/20ms * 10000（10000是PWM的满占空比时候的值）
//舵机最小值为700   最大值为1100
void steer_control(void)
{
    if(steer_enable)
    {
        steer_duty=median+10*key_flag;

        if(steer_duty<=700)
        {
            steer_duty=700;
        }
        if(steer_duty>=1100)
        {
            steer_duty=1100;
        }
        pwm_duty(S_MOTOR_PIN,steer_duty);
    }
}

/*
 * motor.c
 *
 *  Created on: 2022年8月11日
 *      Author: 94380
 */
#include "motor.h"
#include "ui.h"
```

```
int16 motor_err_now = 0;
int16 motor_err_last = 0;
int16 motor_err_sum = 0;
float motor_kp = 25;
float motor_ki = 0.3;
uint16 motor_output = 0;

void motor_init(void)
{
    gtm_pwm_init(MOTOR_PWM1, 12000, 0);
    gtm_pwm_init(MOTOR_PWM2, 12000, 0);

}

void motor_move(int16 pwm)
{
    if(pwm >= 0){
        gtm_pwm_init(MOTOR_PWM1, 12000, 0);
        gtm_pwm_init(MOTOR_PWM2, 12000, pwm);
    }
    else if(pwm < 0){
        gtm_pwm_init(MOTOR_PWM1, 12000, fabs(pwm));
        gtm_pwm_init(MOTOR_PWM2, 12000, 0);
    }
}

void motor_pid(int16 s)
{
    motor_err_now = s - encoder;
    motor_output = (int16)(motor_kp * motor_err_now + motor_ki *
motor_err_sum);
    motor_err_last = motor_err_now;
    motor_err_sum += motor_err_now - motor_err_last;
    if(motor_output > 1600)
        motor_output = 1600;
    if(motor_output < -1600)
        motor_output = -1600;

    motor_move(motor_output);
}
```

```c
/*
 * ui.c
 *
 *  Created on: 2022年8月11日
 *      Author: 94380
 */
#include "ui.h"
#include "servo.h"
#include "motor.h"

uint8 screen_info[8][50];
int16 encoder;

void display_menu(void)
{

    sprintf((char *)screen_info[0], " speed: %.2fm/s  ", speed);
    sprintf((char *)screen_info[1], " angle: %u     ", angle);
    sprintf((char *)screen_info[2], " servo_pwm: %d     ", servo_pwm);
    sprintf((char *)screen_info[3], " motor_pwm: %d     ", motor_pwm);
    sprintf((char *)screen_info[4], " encoder: %d      ", encoder);
    sprintf((char *)screen_info[5], "                    ");
    sprintf((char *)screen_info[6], "                    ");
    sprintf((char *)screen_info[7], "                    ");
    for (int i = 0; i < 8; ++i)
    {
        ips114_showstr(0, i, screen_info[i]);
    }
}


#include "isr_config.h"
#include "isr.h"

// PIT中断函数  示例
IFX_INTERRUPT(cc60_pit_ch0_isr, CCU6_0_CH0_INT_VECTAB_NUM,
CCU6_0_CH0_ISR_PRIORITY)
{
    enableInterrupts(); //开启中断嵌套
```

```
    PIT_CLEAR_FLAG(CCU6_0, PIT_CH0);
}

IFX_INTERRUPT(cc60_pit_ch1_isr, CCU6_0_CH1_INT_VECTAB_NUM,
CCU6_0_CH1_ISR_PRIORITY)
{
    enableInterrupts(); //开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_0, PIT_CH1);
}

IFX_INTERRUPT(cc61_pit_ch0_isr, CCU6_1_CH0_INT_VECTAB_NUM,
CCU6_1_CH0_ISR_PRIORITY)
{
    enableInterrupts(); //开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_1, PIT_CH0);
}

IFX_INTERRUPT(cc61_pit_ch1_isr, CCU6_1_CH1_INT_VECTAB_NUM,
CCU6_1_CH1_ISR_PRIORITY)
{
    enableInterrupts(); //开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_1, PIT_CH1);
}

IFX_INTERRUPT(eru_ch0_ch4_isr, ERU_CH0_CH4_INT_VECTAB_NUM,
ERU_CH0_CH4_INT_PRIO)
{
    enableInterrupts();                    //开启中断嵌套
    if (GET_GPIO_FLAG(ERU_CH0_REQ4_P10_7)) //通道0中断
    {
        CLEAR_GPIO_FLAG(ERU_CH0_REQ4_P10_7);
    }

    if (GET_GPIO_FLAG(ERU_CH4_REQ13_P15_5)) //通道4中断
    {
        CLEAR_GPIO_FLAG(ERU_CH4_REQ13_P15_5);
    }
}

IFX_INTERRUPT(eru_ch1_ch5_isr, ERU_CH1_CH5_INT_VECTAB_NUM,
ERU_CH1_CH5_INT_PRIO)
```

```
{
    enableInterrupts();                    //开启中断嵌套
    if (GET_GPIO_FLAG(ERU_CH1_REQ5_P10_8)) //通道1中断
    {
        CLEAR_GPIO_FLAG(ERU_CH1_REQ5_P10_8);
    }

    if (GET_GPIO_FLAG(ERU_CH5_REQ1_P15_8)) //通道5中断
    {
        CLEAR_GPIO_FLAG(ERU_CH5_REQ1_P15_8);
    }
}

//由于摄像头pclk引脚默认占用了 2通道，用于触发DMA，因此这里不再定义中断函数
// IFX_INTERRUPT(eru_ch2_ch6_isr, ERU_CH2_CH6_INT_VECTAB_NUM,
ERU_CH2_CH6_INT_PRIO)
//{
//  enableInterrupts();//开启中断嵌套
//  if(GET_GPIO_FLAG(ERU_CH2_REQ7_P00_4))//通道2中断
//  {
//      CLEAR_GPIO_FLAG(ERU_CH2_REQ7_P00_4);
//
//  }
//  if(GET_GPIO_FLAG(ERU_CH6_REQ9_P20_0))//通道6中断
//  {
//      CLEAR_GPIO_FLAG(ERU_CH6_REQ9_P20_0);
//
//  }
//}

IFX_INTERRUPT(eru_ch3_ch7_isr, ERU_CH3_CH7_INT_VECTAB_NUM,
ERU_CH3_CH7_INT_PRIO)
{
    enableInterrupts();                    //开启中断嵌套
    if (GET_GPIO_FLAG(ERU_CH3_REQ6_P02_0)) //通道3中断
    {
        CLEAR_GPIO_FLAG(ERU_CH3_REQ6_P02_0);
        if (CAMERA_GRAYSCALE == camera_type)
            mt9v03x_vsync();
        else if (CAMERA_BIN_UART == camera_type)
            ov7725_uart_vsync();
```

```
        else if (CAMERA_BIN == camera_type)
            ov7725_vsync();
    }
    if (GET_GPIO_FLAG(ERU_CH7_REQ16_P15_1)) //通道7中断
    {
        CLEAR_GPIO_FLAG(ERU_CH7_REQ16_P15_1);
    }
}

IFX_INTERRUPT(dma_ch5_isr, ERU_DMA_INT_VECTAB_NUM, ERU_DMA_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套

    if (CAMERA_GRAYSCALE == camera_type)
        mt9v03x_dma();
    else if (CAMERA_BIN_UART == camera_type)
        ov7725_uart_dma();
    else if (CAMERA_BIN == camera_type)
        ov7725_dma();
}

//串口中断函数  示例
IFX_INTERRUPT(uart0_tx_isr, UART0_INT_VECTAB_NUM, UART0_TX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart0_handle);
}
IFX_INTERRUPT(uart0_rx_isr, UART0_INT_VECTAB_NUM, UART0_RX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart0_handle);
}
IFX_INTERRUPT(uart0_er_isr, UART0_INT_VECTAB_NUM, UART0_ER_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrError(&uart0_handle);
}

//串口1默认连接到摄像头配置串口
IFX_INTERRUPT(uart1_tx_isr, UART1_INT_VECTAB_NUM, UART1_TX_INT_PRIO)
{
```

```
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart1_handle);
}
IFX_INTERRUPT(uart1_rx_isr, UART1_INT_VECTAB_NUM, UART1_RX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart1_handle);
    if (CAMERA_GRAYSCALE == camera_type)
        mt9v03x_uart_callback();
    else if (CAMERA_BIN_UART == camera_type)
        ov7725_uart_callback();
}
IFX_INTERRUPT(uart1_er_isr, UART1_INT_VECTAB_NUM, UART1_ER_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrError(&uart1_handle);
}

//串口2默认连接到无线转串口模块
IFX_INTERRUPT(uart2_tx_isr, UART2_INT_VECTAB_NUM, UART2_TX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart2_handle);
}
IFX_INTERRUPT(uart2_rx_isr, UART2_INT_VECTAB_NUM, UART2_RX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart2_handle);
    switch (wireless_type)
    {
    case WIRELESS_SI24R1:
    {
        wireless_uart_callback();
    }
    break;

    case WIRELESS_CH9141:
    {
        bluetooth_ch9141_uart_callback();
    }
    break;
```

```
    default:
        break;
    }
}
IFX_INTERRUPT(uart2_er_isr, UART2_INT_VECTAB_NUM, UART2_ER_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrError(&uart2_handle);
}

uint8 uart_buff = 0;
uint8 uart_count = 0;
uint8 data[uart_length] = {0};
uint8 data_flag = 0;
IFX_INTERRUPT(uart3_tx_isr, UART3_INT_VECTAB_NUM, UART3_TX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart3_handle);
}
IFX_INTERRUPT(uart3_rx_isr, UART3_INT_VECTAB_NUM, UART3_RX_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart3_handle);
    if (uart_query(UART_3, &uart_buff))
    {
        data[uart_count] = uart_buff;
        if (data[0] == 0x42)
        {
            uart_count++;
        }
        if (uart_count == uart_length)
        {
            if (data[uart_length - 1] == 0x10)
            {
                data_flag = 1;
                uart_count = 0;
            }
            else
            {
                for (uart_count = 0; uart_count < uart_length; uart_count++)
                    data[uart_count] = 0;
```

```
                uart_count = 0;
                data_flag = 0;
            }
        }
    }
}
IFX_INTERRUPT(uart3_er_isr, UART3_INT_VECTAB_NUM, UART3_ER_INT_PRIO)
{
    enableInterrupts(); //开启中断嵌套
    IfxAsclin_Asc_isrError(&uart3_handle);
}
```