第十七届全国大学生
智能汽车竞赛

RT-Thread 创新论文

AI 复杂任务环境下 RT-Thread 等开源工具在智能小车制作中的应用

学　　校：　　山东大学（威海）

队伍名称：　　海韵五队

参赛队员：　　王昕阳

　　崔海勤

　　刘子晖

　　张传钰

带队教师：　　王小利

关于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。

张传钰

参赛队员签名：王昕阳 崔瀚勃 刘子辉

带队教师签名：

日　　期：2021/7/27

摘要：本文主要介绍了基于 RT-Thread 等开源软件设计完成的自主循迹小车的原理及其设计，探究证实了 RT-Thread 等开源软件在小车开发中的重要作用。对小车的设计包括机械结构的调整、小车硬件设计、AI 识别部分的训练、传感器的信号处理以及控制策略的原理和实现。智能小车以 I 车模为基础，使用赛题要求的 USB 摄像头赛道数据，由 EdgeBoard 进行赛道数据的处理，英飞凌的 TC264 单片机以及 RT-Thread 作为控制核心。CSPE5-500 型光电编码器实时获取小车的速度信息，配合小车自带的显示屏、无线串口、按键进行速度参数的调整。搭载有 RT-Thread 的控制核心充分利用了其多线程、FinSH 等功能，并且借助其实现了更好的性能，以及自制的上位机以方便进行调参，使得智能小车在赛道上的稳定循迹运行，并在最近结束的山东赛区的比赛中获得了线下赛综合奖励时间后计算得出-2.173s 的好成绩，为山东赛区及全国省赛已公示成绩中线下赛第一名。

关键字：智能汽车比赛；RT-Thread；PaddlePaddle；EdgeBoard；FPGA；嵌入式系统；机器学习；开源软件

目录

第一章　引言

社会的不断进步与发展，伴随的是对自动化技术需求的不断增加，同时自动化程序也呈现出任务复杂化和性能要求高的特点。由此而产生的智能汽车比赛，可以很好的将工程实践带入到学习中，帮助大学生快速掌握实践中的技巧。

为防止克隆车老爷车的出现，同时也为了竞赛趣味性，大赛组委会不断新增和调整赛题任务，使得当前的智能车比赛中小车需要完成的任务不断变多，且内容也变得更加复杂。面对这种问题，将工程思想、方案和前沿的科技带入到小车制作中是刻不容缓的。

智能汽车竞赛完全模型组一次引入了许多往年不存在，但是具有很强创新性的元素，此外还新增了 EdgeBoard 的使用，引入了 PaddlePaddle AI 框架。复杂的任务环境，使竞速组的比赛有了创意组的趣味，也对参赛队伍带来了很大的挑战。

RT-Thread 提供的多线程支持，及其配套的内存管理，线程同步，控制台等功能，可以有效的拆分程序代码，提升代码可维护性，并且**提升重要任务的执行性能，降低开发复杂性**。

经过我们的实践经验表明，在 RT-Thread 提供的许多便捷功能的加持下，参赛队伍的成绩可以有很大的改善，其引入，很大的提升了代码的模块化水平，**使得恶性 bug 的发生率出现了明显的下降**，调车过程有了很大的提速，从而使我们团队在综合了奖励时间后，得到了车模运行时间为**负数**的好成绩。

与此同时，我们团队经过 2 年参赛过程中的 RT-Thread 的使用经验积累，沉淀了许多技术细节。其中的部分内容，基于我们对开源软件的热爱，以及对回馈社区的渴望，已经上传到 GitHub 等开源社区，供全体参赛队伍参考。

小车以全新的 I 型车模作为机械结构的基础，搭载有组别特殊要求的 EdgeBoard 开发板进行图像信息的处理。TC264 作为车模控制与调试的核心，运行有 RT-Thread 操作系统。

本文针对 RT-Thread 等其他开源工具在我们团队的智能车制作中的实际使用情况与相应的改进方式作出论述：

（1） 第二、三章对车模的机械结构与电路设计进行了说明，讨论了不同机

械结构的调整对小车运行的影响；

（2） 第四章对小车设计中 AI 部分的工作进行了说明，论述了将 AI 技术引入小车的过程；

（3） 第五章对小车独特的硬件 EdgeBoard 使用及其对图像的处理进行了说明，介绍了新硬件 EdgeBoard 对我们带来的挑战与机遇；

对搭载有 RT-Thread 的 TC264 上任务进行说明：

（4） 第六章对小车的控制原理与策略进行了说明，介绍了我们组别在小车调参过程中的方法与技巧；

（5） 第七章对我们组别处理的各种不同的多线程环境进行了论述，并将其与 RT-Thread 进行对比，体现出 RT-Thread 独特的作用；

（6） 第八、九章对 RT-Thread 的选择与移植的过程进行了论述，简要介绍了开源的移植项目，及其关联的技术博客；

（7） **第十章介绍了利用多线程、排他锁、邮箱等 RT-Thread 特性优化小车的程序；**

（8） 第十一章介绍了功能强大的 FinSH，并且介绍了将 FinSH 作为基础编写的**有自己特色的** Web 上位机。

（9） 第十二章列出了我们为了回报 RT-Thread 等社区进行开源的项目的清单，以及其对应的技术博客。

文中我们引用了《“飞思卡尔”杯智能汽车竞赛设计与实践》作为智能小车制作的方向参考，《汽车理论》作为整车设计的参考，《嵌入式实时操作系统：RT-Thread 设计与实现》作为 RT-Thread 原理的研究，《RT-Thread 内核实现与应用开发实战指南 基于 STM32》为其实际使用中的参考，《鸟哥的 Linux 私房菜》作为 EdgeBoard 上 Linux 使用指南，《数字电子技术基础》作为电路设计的参考，《采用模糊 PID 控制率的舵机系统设计》作为 PID 整定的参考。

第二章　车模整体设计

2.1　I 车模基础结构介绍

I 车模是第十七届全国大学生智能汽车竞赛完全模型组的指定车模，车体已安装电机、轮胎、轮毂、差速器、轴承、减震器、舵机拉杆、车壳、底盘等。

I 车模底盘采用高强度碳纤板，具有较高的强度，刚性足，车模尺寸为312*192*368mm。

I 车模采用单电机后驱，前轮舵机控制阿克曼转向：

（10）驱动电机为 RS-555 微型直流电机，额定电压 12V，额定转速12000rpm，具有较为强劲的动力；

（11）舵机为 CS-3120 数字舵机，内部采用金属齿轮，精度较高，不易扫齿。空载转速在 6.8V 时可以达到 0.14s /60°，额定扭矩可达17kg*cm ，扭矩大，动力强劲；

（12）该车模具有快速，精度高、响应快、动力强劲、抓地力强等优势。

2.2　传感器

在传感器的选择上，我们使用 1 个 USB 720P 120 帧工业摄像头来进行赛道信息采集，实现寻迹任务。此摄像头优势在于：

（1） 高帧率，全局快门：图像具有高流畅性，使得车模在高速状态下依旧能流畅地获取有效的赛道信息，实现高速稳定寻迹；

（2） 高分辨率无畸变：图像能够真实还原赛道信息，便于后期对赛道信息进行处理；

（3） 全彩：便于获取赛道上的色彩信息，以完成完全模型组的特定比赛任务；

（4） USB 接口免驱动：简便操作，可靠性高。

在原厂摄像头支架上，除了螺纹紧固，我们还进行了加固处理，防止摄像头在高速震动下出现松动与位移。摄像头固定座部分也进行了加固，使得在保证摄像头前瞻的同时，尽可能提高摄像头的稳定性。同时加装偏振片以减少赛道反光对图像二值化的严重影响，如下图所示：

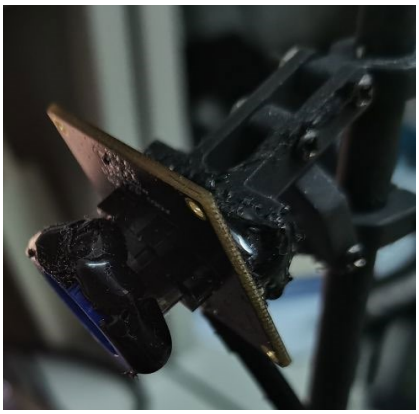


图 2.1　摄像头的紧固与偏振片的安装

配合 CSPE5-500 型光电编码器， 该编码器与电机组成一个整体，使用过程中稳定性高。通过光电编码器实现车模的闭环控制，以更好地完成比赛任务。

2.3　电池

在电池的选择上，我们选择 3S 11.1V 2400mAh 30C 锂聚合物电池 1 个。该电池的优点是：

（1） 电池容量较大，能满足较高强度的调车与板卡供电需求。
（2） 体积较为小巧，尺寸为 112mm*32mm*20mm，占用车模空间少，便于车模布局。
（3） 重量相对较轻，重量为 182g，有利于轻量化车模，提高车模速度。
4） 电压合适，能够满足电机与板卡的供电需求。
（5） 放电倍率大，可达 30C，完全满足电机的瞬时大电流需求，为车模提供强劲的爆发力。
（6） 支持充电电流较大，可达 3A，充电时间短，两块电池即可基本保证较高强度的连续调车需求。

在电池的使用上，针对锂聚合物电池的放电能力强的特点，我们在使用过程中全程连接锂电池报警器，一可以实时监控电池电压，二可以避免锂电池过度放电以造成对锂电池的不可逆性损伤。充电使用 IMAX B6AC 80W 平衡充电器，

该充电器能实现锂电池组平衡充电，有利于保护电池，维持电池容量，相对增加使用次数，而且功能强大，有快充、储存充、放电等功能。

2.4　轮胎与四轮定位

为了使汽车转向轻便，转向后能自动回正，直线行驶稳定，具有良好的直线和弯道行驶性能，同时减少轮胎和转向零件的磨损等，在转向轮、转向节和转轴之间须形成一定的相对安装位置，即四轮定位。I 车模与其他车模不同，提供了对后轮定位的可能性。

2.4.1　概念简介

2.4.1.1　主销后倾

主销后倾是指在纵向平面内主销轴线与地面垂直线之间的夹角，向后为正。转向主销(车轮转向时的旋转中心)向后倾倒，它在车辆转弯时会产生与车轮偏转方向相反的回正力矩，使车轮的方向自然朝向行驶方向。设定较大的主销后倾角可使前轮自动回正的能力增强，可提高直线行驶性能，但是主销后倾过大，会使转向沉重。I 车模通过增减垫片的数量来改变主销后倾角。由于竞赛所用的转向舵机扭矩较大，所以我们尝试了较大的主销后倾角，有效提高了直线行驶的稳定性，如下图所示

图 2.2　主销后倾

2.4.1.2 主销内倾

主销内倾是从车前后方向看轮胎时，将主销的上端向内倾斜，主销轴线与通过前轮中心的垂线之间形成一个夹角，即主销内倾角，向内为正。调整后直线行驶时轮胎内侧先着地，而当过弯的时候，车体会向弯道外侧倾斜，此时则可胎面着地，从而使车辆在弯道获得较大抓地力。同时车模本身的重力有使转向车轮回复到原来中间位置的效应，因而方向回正容易，使车轮转向轻便。在车模上的调整如下图所示。

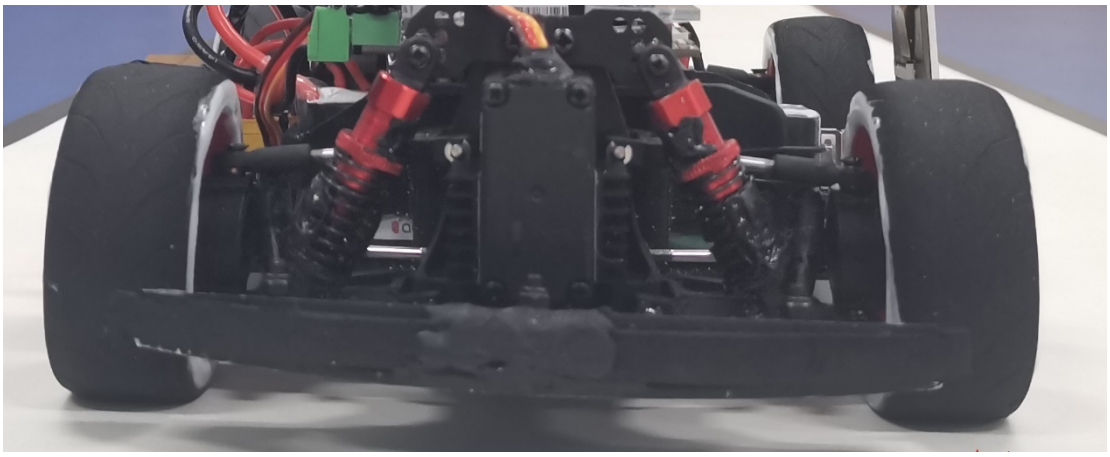


图 2.3 主销内倾

2.4.1.3 车轮前束

车轮前束指从上往下看时，前轮中心线与纵向中心线夹角。两个车轮呈“内八字”称为车轮前束，指向外的即“外八字”则称为车轮后束。前轮前束的作用是减少轮胎的磨损。由于上述主销内倾的调整，轮胎向外侧转动，如果前束适当，轮胎转动时的偏斜方向就会得到修正，左右两轮可保持直线行进，减少轮胎磨损。在 I 车模上的调整如下图所示。

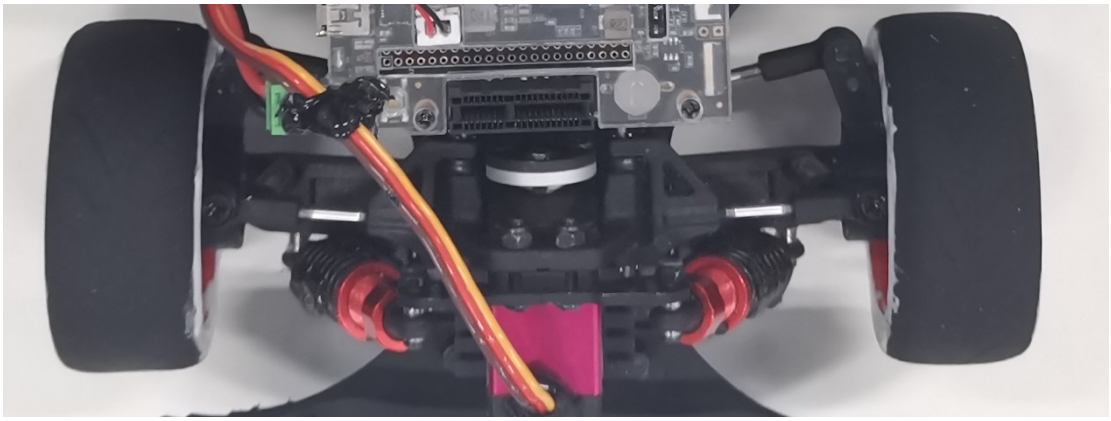


图 2.4　车轮前束

2.4.2　轮胎处理

I 车模轮胎质地较为柔软，原厂填充海绵有利于车模的平稳行驶；轮胎花纹较为稀疏，具有较强的抓地力。

轮胎尺寸为：胎宽 27mm，直径 65mm。

通过对轮胎进行适当的软化与密封处理，使轮胎与地面具有更大的接触面积和摩擦力，同时使车模具有较好的减震性，为车模在高速状态下的平稳行驶提供了保证。处理后的轮胎如下图所示。

图 2.5　处理后的轮胎

2.4.3 四轮定位中的特殊处理

与其他车模不同的是，I 车模提供了对后轮定位的可能性，可以对后轮进行主销内倾与车轮前束处理，进一步增加车模行驶的稳定性。

2.5 差速器与减速器

不同于其他单电机驱动室内赛道车模，I 车模后桥驱动中使用了一个全金属差速器来进行传动，此差速器结构类似于真实汽车中的差速器：当转弯时，外侧轮有滑拖的现象，内侧轮有滑转的现象，此时就会产生两个反向的附加力，为了保证两侧驱动车轮作纯滚动运动，让汽车曲线行驶旋转速度基本一致，且由“最小能耗原理”可得，必然两边车轮的转速不同，通过差速器的调节，行星齿轮产生自转，使内侧半轴转速减慢，外侧半轴转速加快，从而实现两侧车轮转速的差异，减少轮胎与地面的摩擦，提高了车模的转弯性能，使得车模能够在高速寻迹状态下依旧可稳定行驶。

I 车模的差速器与减速器为一体式结构。为了使电机传动更加顺畅，我们合理调节减速器与电机输出轴的啮合程度，并且使用润滑脂润滑，减少了车模高速行驶时的噪音与磨损，同时保证了传动效率，基本杜绝了打齿现象的发生。润滑脂粘度大，留存时间长，相比润滑油飞溅污染少，润滑性能也不逊色。

2.6 减震悬挂系统

不同于其他室内赛道车模，I 车模拥有一套可调的减震悬挂系统，该悬挂系统通过弹簧和阻尼器来缓冲震动，将单次的瞬时大能量冲击通过增加力的作用时间来减少冲击力。但是原厂减震系统中的弹簧劲度系数较低，直观表现为悬挂较软，回弹不及时，这对于高速运动下的车模的姿态稳定性产生了较大的影响，进而导致摄像头获取赛道信息数据失真，影响了正常寻迹。

我们通过调节弹簧至最“硬”状态，并且使用热熔胶固定一部分弹簧行程，对于阻尼器部分，我们通过涂抹油脂密封尽量提高阻尼器密封性，以减少阻尼器

的行程。通过上述方法，减震系统的硬度大大提高，高速状态下车模姿态也较为稳定。如下图所示。

图 2.6　处理后的减震悬挂系统

2.7　整体设计优化

I 车模相比传统室内组别车模，重量较大，重心较高，由于 EdgeBoard 的存在，结构布局基本相对固定。我们通过如下几个方面实现了整体设计上的优化。

（1） 降低重心与绝缘设计：电路板尽量贴合车模空余空间形状，为电路板争取尽量多的功能空间。同时采用短铜柱与底盘连接，提高连接刚性，尽量贴紧底盘以降低重心。由于底盘为碳纤板，具有导电性，设计 PCB 时特意进行了绝缘措施，并且在铜柱与电路板间以垫片以及电工胶带隔离。如下图所示。底盘下同时加装少量配重块，显著减少了车模在高速行驶中的侧翻与打滑现象。

图 2.7　降低重心与绝缘措施

（2） 紧固设计：部分不常拆卸的部件如舵机舵盘以及摄像头支架、前桥等，均进行了打胶防松处理，防止在高速振动状态下，上述部件自松。

（3） 紧凑化设计：为了使车模更加美观以及减少风阻、降低重量、增加连接可靠性，减少车壳外连线的出现，所有电源线、信号线、USB 线等均经过测量挑选，多余线头作打胶折叠处理，防止与运动机械结构产生干涉，同时保证了结构紧凑美观。

（4） 人性化设计：右手侧放置开关、按键等调参工具，左手侧紧贴车壳外侧放置屏幕便于显示与观察。车壳采用快拆设计，无需拆装摄像头即可进行车壳拆装。上述设计有效提高了调车效率。

（5） 对称化设计：机械结构上尽量使车模左右两侧重量相似，保持重心位于车模中轴线附近，提高车身高速状态下的稳定性。

（6） 轻量化设计：部件均经过挑选减重，以降低车身惯性，提高加减速性能与灵活性。

（7） 散热优化设计：车壳镂空散热、舵机与 MOS 管通过散热片散热，提高了系统整体的可靠性。

2.8　I 车模车壳设计

基于 I 车模原厂车壳，针对调试过程中暴露出的各种问题，我们对原厂车壳进行了合理改装。

（1） 调试不方便：在车模两侧切割预留调试口，顶部预留摄像头快装口，

以便于调试。

（2） 散热困难：连续高强度调车时，EdgeBoard 以及电机散热困难，积攒热量，严重影响车模性能与调车进度，遂对车模进行部分镂空处理，并且为了美观，加装黑色 PVC 防尘散热网，有效延长了车模的续航时间，提高了调车效率。

如下图所示：

图 2.8　I 车模原厂车壳

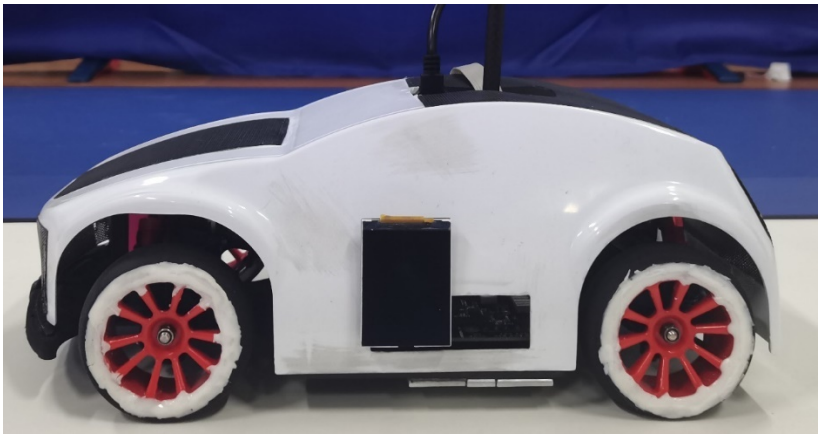


图 2.9　合理改装后的车壳

2.9 车模整体设计结构图

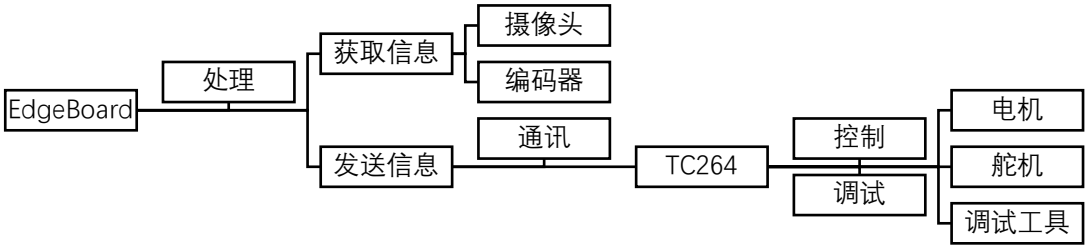


图 2.10 车模整体设计结构图

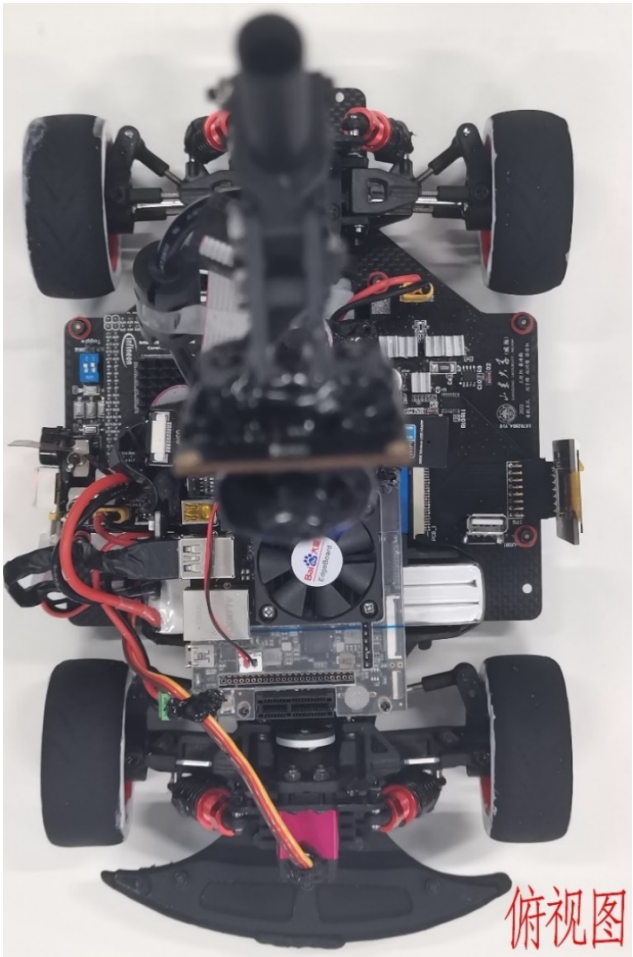


图 2.11 整车机械布局

第三章　硬件设计

3.1　运动控制芯片

车模运动控制部分使用大赛指定的英飞凌公司的 TC264 作为控制芯片，核心板原理图如下图所示。

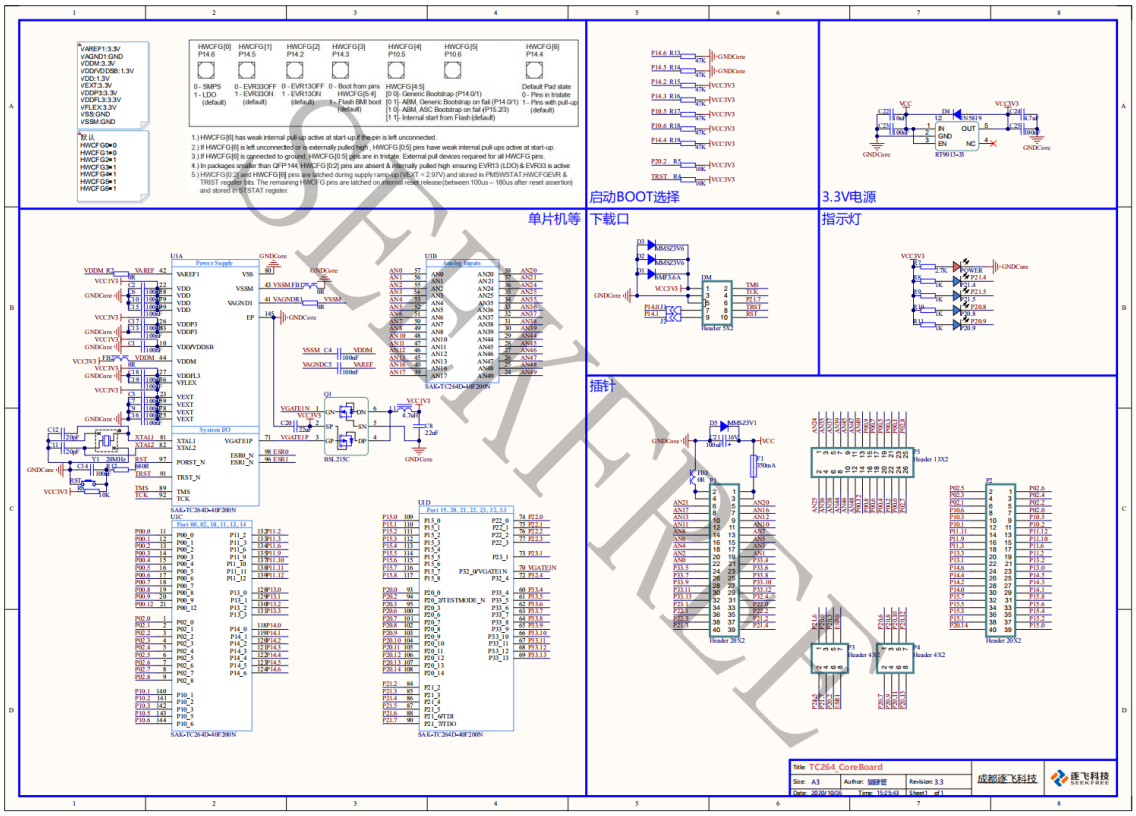


图 3.1　核心板原理图

3.2　运动控制主板设计

3.2.1　稳压电路设计

车模由于采用 11.1V3S 锂电池供电，而主板上共有三种工作电压：

5V，用于编码器与调试用无线串口供电.

6.6-7V，用于舵机供电.

3.3V，用于 TC264、CH340C 和调试工具等的供电.

所以需要一套稳定可靠的稳压电路实现主板供电。

如下图所示，12V-5V/6.6V/7V 稳压电路使用 TI 公司的 TPS5430 宽输入 3A 降压转换器搭建 BUCK 降压电路（开关型稳压电路一种）。

优点是：

（1） 带载能力强，可支持 3A 大电流（瞬时 4A）输出，能够完全满足芯片供电需求，满足舵机瞬时堵转大电流。

（2） 可支持宽电压输入（5.5-36V）。

（3） 效率高，可达 90%，发热小，可靠性高。

（4） 外部电路相对简单，可靠性强，变压输出仅需更换分压电阻即可（更好的方案是设置一个贴片滑动变阻器）。

（5） 纹波小，经测试低于 30mV，供电质量好，为芯片正常工作提供必要条件。

5V-3.3V 稳压电路使用 AMS1117 低压差线性稳压器搭建降压电路。

优点是：

（1） 压降低，符合应用场景。

（2） 外部电路简单，可靠性高。

（3） 支持 800mA 电流输出，可以满足带载需求。

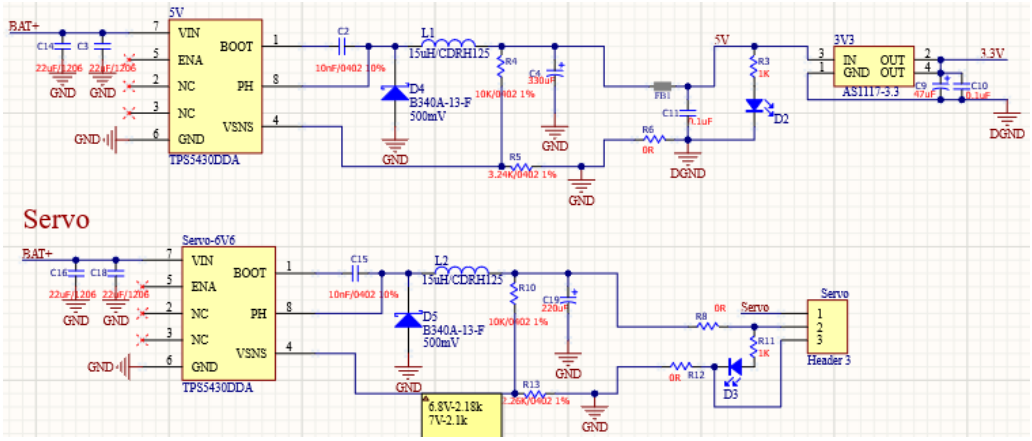


图 3.2　稳压电路设计

3.2.2　调试工具设计

为了在有限的 PCB 空间里创设更好的调试条件，遂预留了一个 2P 拨码开关、五个按键，实践证明，此方案不仅占用空间较小，而且能够满足调试需要。

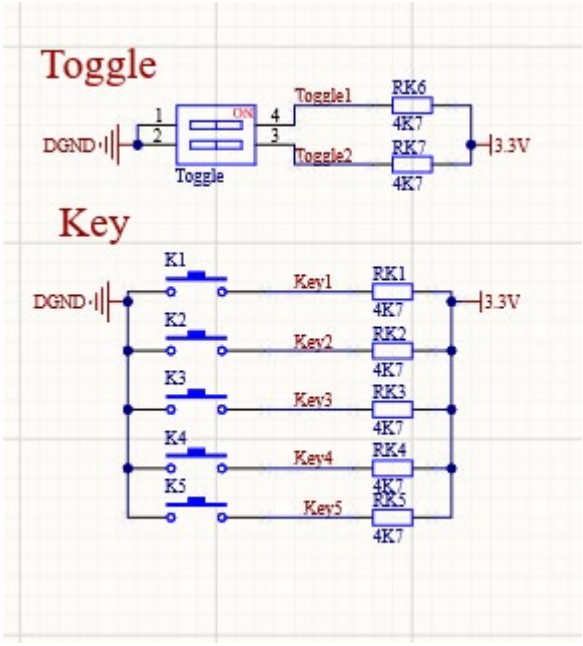


图 3.3　调试工具设计

3.2.3　通讯设计

与 EdgeBoard 通讯部分应规则要求，我们自行设计了 USB-TTL 电路，采用差分布线以提高通讯质量。芯片使用 WCH 公司 CH340C，其优点在于内置时钟，无需外部晶振，外部电路简单，占用 PCB 面积小，可靠性强。

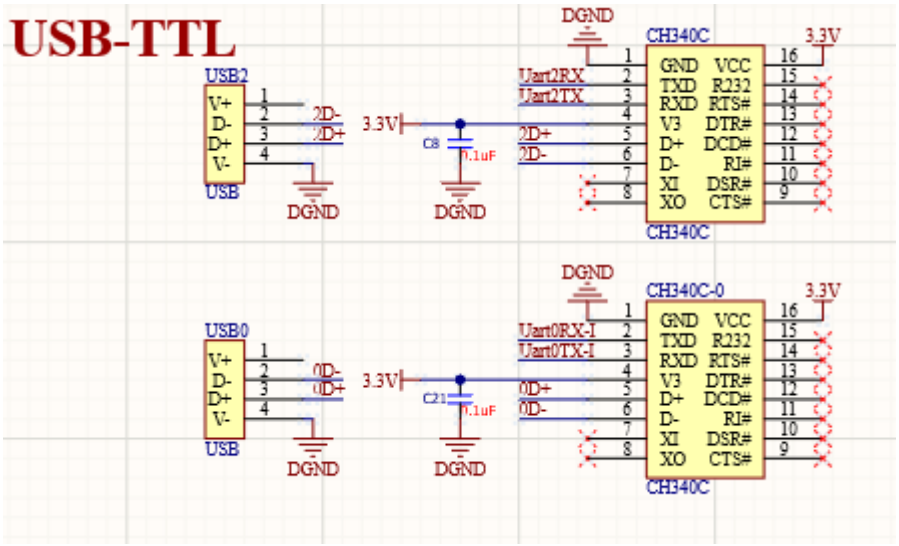


图 3.4　通讯设计

3.3　电机驱动板设计

控制信号进入电机驱动板首先经过 74HC244 隔离芯片，防止电流反冲进入主板，保护芯片。H 桥部分瞬时电流大，采用大面积铺铜以及贴敷散热片来提高电路稳定性。

电机控制部分，我们选用 TI 公司的 5.9V-45V 宽电压工作范围，且支持 1.8V、3.3V 和 5V 三种逻辑输入的 DRV8701E 全桥栅极驱动器，驱动外部 4 个 N 沟道 MOS 管。

该驱动芯片的优点在于：

（1） 内置以下保护功能：欠压锁定、电荷泵故障、过流关断、短路保护、前置驱动器故障以及过热保护。工作稳定，未曾出现故障。
（2） 加装采样电阻可测量 H 桥中电流，实现电流环控制。
（3） 封装尺寸小，外围电路占用面积小。
（4） 栅极驱动可调节。

H 桥部分的 N 沟道 MOS 管我们选用东芝公司的 TPH1R403NL。

该 MOS 管的优点在于：

（1） 漏源电压可耐受达 30V，栅源电压可耐受达 20V。

（2） 漏极电流最大可耐受 60A

可靠性强，合理使用下未曾烧毁。

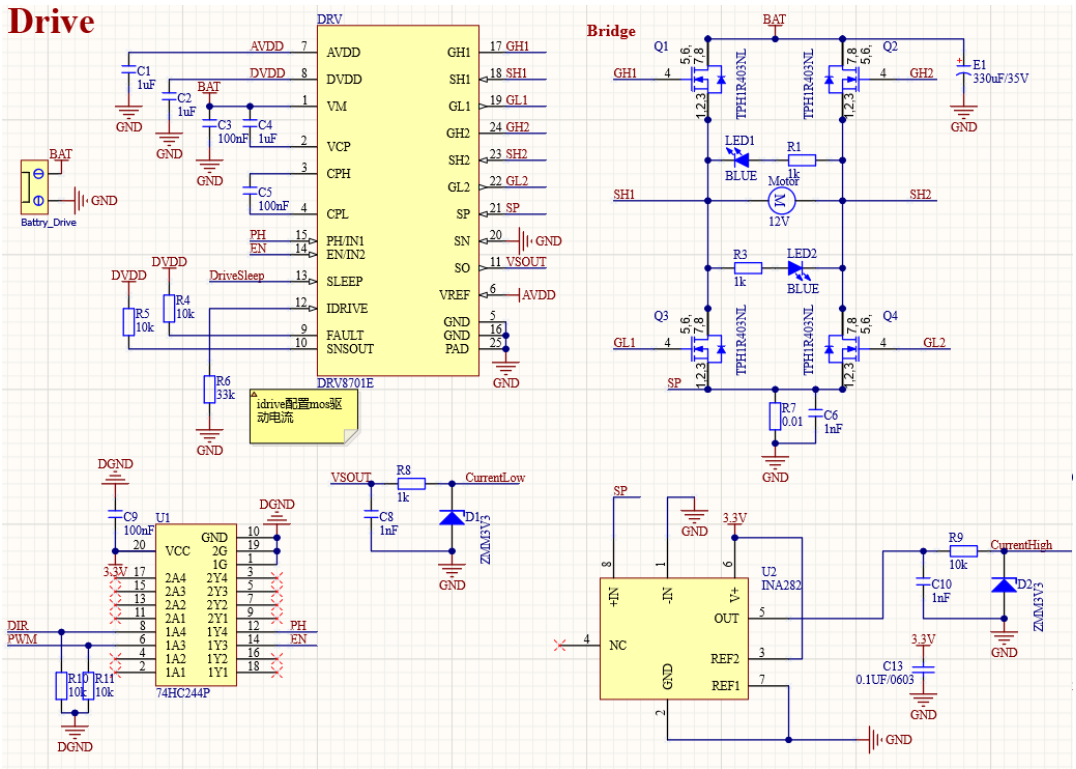


图 3.5　电机驱动板设计

第四章　AI 在智能车比赛中的调试与应用

4.1　简介

随着信息化进程的不断加深，多样化数据的处理成为了一个新的挑战。AI 等技术的发明，可以从数据中挖掘中不断发掘新的规律，处理一些传统方式无法处理的问题，如手写数字识别等，使得数据有了更加深刻的价值。故将 AI 技术引入生产生活中非常有必要。本次比赛中我们采用了百度公司开源的国产 AI 学习框架 PaddlePaddle，训练过程在百度公司的 AI Studio 上进行。

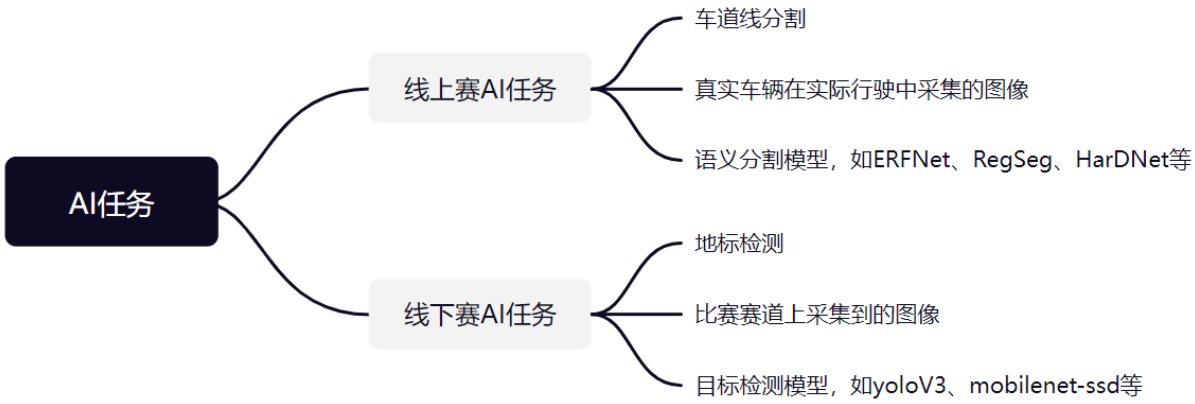


图 4.1　AI 任务简图

上图为本组别进行的 AI 任务简图，下文将会详细论述 AI 在小车中的具体应用。

4.2　任务介绍

完全模型组的比赛通过设置线上赛与线下赛两个任务来将 AI 带入比赛中：

线上赛比赛给定一组数据集，其中为百度公司在实际生产中采集的图像信息，图像中标注有车道线等信息。我们的任务为设计并且调试一个语义分割[21]模型，使模型可以读取输入的图像信息，并且将车道线信息准确的预测出来[22]。

线下赛比赛的任务开放，不限制模型的使用，能够成功的识别比赛中的赛道标识并且进行对应的处理即可。比赛过程中需要识别的赛道标识如下图所示。

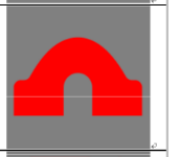
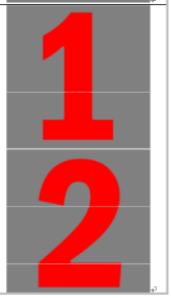
序号	名称	说明	图示
1	泛行区标志	表示前方三岔路口围成的泛行区域，内部区域包括蓝色底布均可行驶。	
2	禁止通行标志	放置在泛行区域进出口连线上，车辆需要绕过此标志进行通行。（此标志高出距离地面有 2cm 高度其余均紧贴）	
3	施工区标志	表示前方为施工区，需要绕行赛道外障碍桩围成的临时路段。	
4	坡道标志	表示道路前方有坡道。	
5	加油站标志	表示前方为加油站，车辆需要驶入加油站并按照指定的出口驶出加油站。	
6	加油站出口数字标志	加油站设置有“1”和“2”两个出口，并在出口地面贴有对应的“1”和“2”数字标志。 比赛时加油站的入口处会随机放置车辆需要驶出时的出口数字。	

图 4.2　赛道标志

4.3　数据的采集标注

神经网络作为万能函数拟合器，而一个函数的主要特征即为自变量因变量以及变量间的关系，故一个模型的训练过程主要涉及到如下图所示的内容：

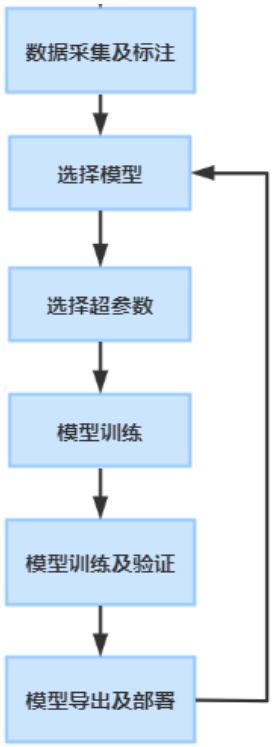


图 4.3　模型的训练过程

线上赛中，数据由百度官方提供。一份文件对应一张标注图片。

线下赛中，图像的标注以及采集需要自行进行，图为使用 labelImg 软件进行数据标注的过程。

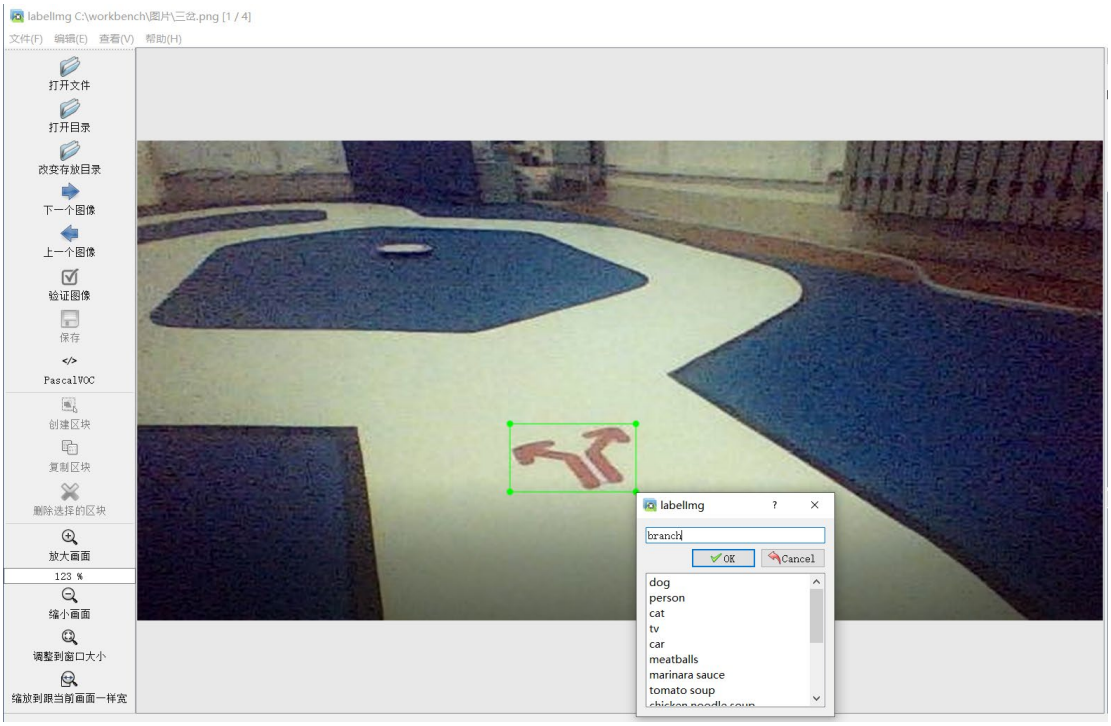


图 4.4　通过 labelImg 进行数据标注

为了增强模型的健壮性与多环境下的适应性，我们采取了：

（1）　学校中不同实验室赛道铺设环境下（光线、赛道老旧不同）进行多种数据采样；

（2）　更换不同参数的摄像头进行数据采样；

（3）　结合线下赛开源项目中提供的图片进行加强；

（4）　与其他学校的参赛队员交换进行加强。

除了在图片源头方面采用了这些方式进行加强,同时也采用了 PaddlePaddle 框架自带的图像增强函数，通过改变对比度、随机裁剪[23]等方式进行了数据增强。

数据采集上，我们也结合了 RT-Thread 提供的便捷功能，通过 Web 上位机，使用手柄遥控小车进行数据采集，该内容将在 RT-Thread 在调试中的应用一章中详细论述。

经过测试，上述数据增强的方式效果良好，以此训练出的模型满足比赛使用的需要。

4.4 模型的训练部署

一个机器学习模型的好坏往往会影响这个模型吸收图片特征的能力。对于不同的比赛任务，需要采用不同的模型。

线上赛方面，尝试了 ERFNet，RegSeg，HarDNet 等模型，综合比较后采用的是 HarDNet 作为提交模型。

模型预测的过程中，我们发现部分模型无法在比赛要求下的时间内完成预测，故我们也使用了 Python 下的多线程进行了加速，该内容将在多种并行任务的挑战与 RT-Thread 的选择一章中进行论述。

线下赛方面，尝试了一些模型，其中 Yolo v3（TODO 增加 yolo 文献）模型虽然在一些特定的场景下，效果较好，但是鉴于 mobilenet ssd[24]出色的高帧率性能表现，以及小车运行中对模型速度的要求，线下赛中我们选择了 mobilenet ssd 作为网络模型。

训练过程主要在百度免费算力的 AI Studio 平台上进行，百度 AI Studio 使用上类似 Jupyter Notebook，如下图所示。

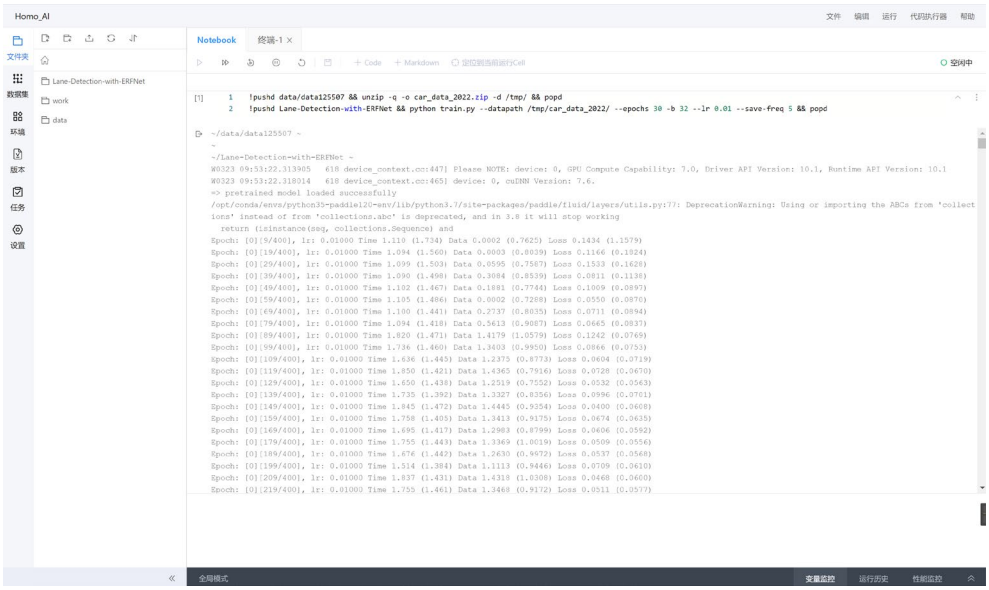


图 4.5 训练过程

相对于传统的 Notebook，百度 AI Studio 还提供有后台任务以供用户在无人值守的情况下也可以训练模型，下图为我们利用百度 AI Studio 的功能进行模型训练的过程。

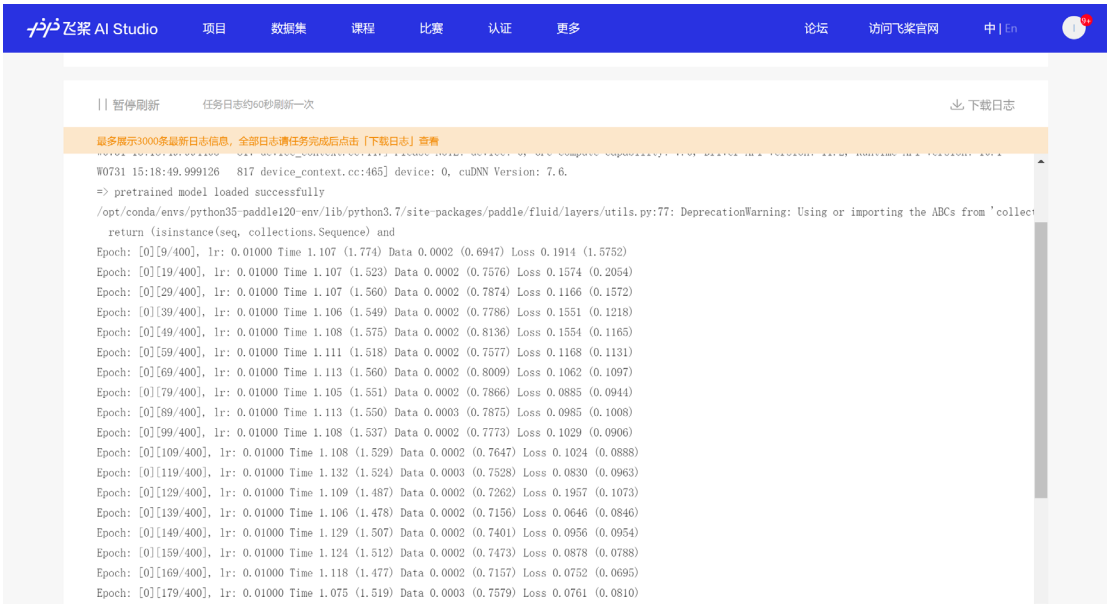


图 4.6　模型训练过程

线下赛部署上，我们使用了融合部署的方案，将 AI 与传统算法结合起来，利用 AI 的复杂能力处理能力，准确的识别并且分类地标，利用传统循迹的可预测性与稳定性，读取 AI 识别到的数据并且对其进行处理，进行锥桶连线等操作以实现中线的计算。详细过程将在 EdgeBoard 的使用和识别与通讯算法的设计一章中论述。

第五章　EdgeBoard 的使用和识别与通讯算法的设计

5.1　EdgeBoard 的配置与使用

第十七届智能车百度完全模型组要求使用 EdgeBoard 计算卡（FZ3B 赛事定制版）。基于 FPGA (Zynq ZU3)实现的 EdgeBoard FZ3B，实测算力达 1.2TOPS，接口齐全。通过嵌入集成计算卡，可快速打造端侧智能硬件。EdgeBoard 是百度面向嵌入式与边缘计算场景打造的 AI 解决方案。丰富的硬件选型，可满足多变的边缘部署需求。无缝兼容百度大脑工具平台与算法模型，开发者既可以选用已有模型，也可以自定义算法。 同时，模型训练与部署全程可视化，极大降低了开发与集成门槛。EdgeBoard 灵活的芯片架构，可适配行业内最前沿、效果最好的算法模型。对于第十七届智能车完全模型组，EdgeBoard 计算卡强大的算力以及良好的模型适配性能可以很好的满足智能车对于传统循迹任务以及赛道特殊元素 AI 识别任务的时间和模型选择要求。

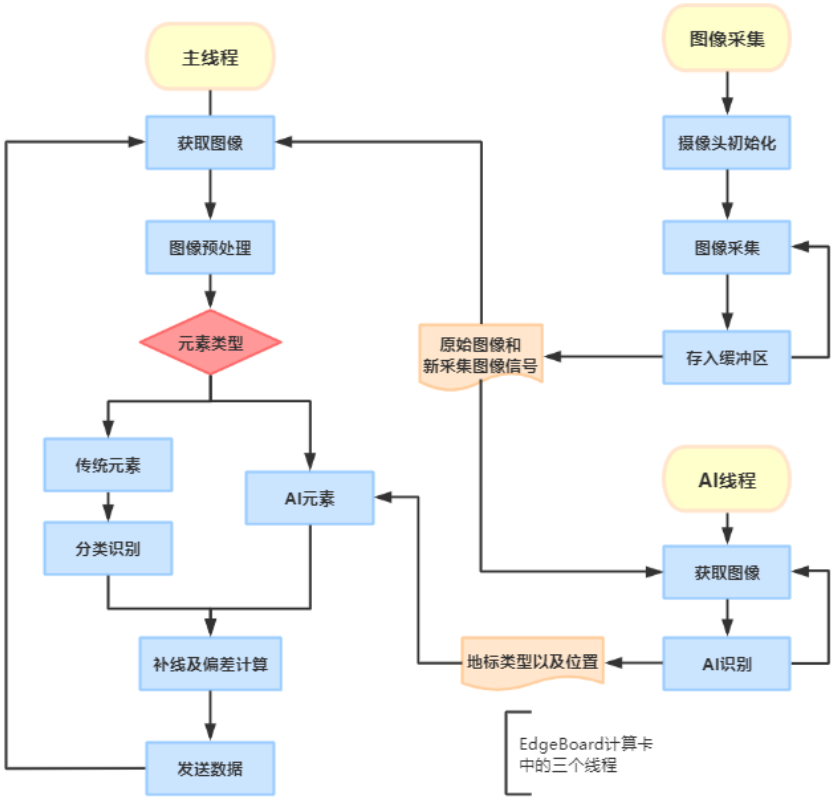


图 5.1　EdgeBoard 上程序流程框图

就十七届智能车比赛完全模型组来讲，我们使用 EdgeBoard 计算卡（FZ3B 赛事定制版）来完成上层识别以及决策。EdgeBoard 上执行的程序流程框图已在上图给出，以下将详细介绍我们组就第十七届智能车竞赛完全模型组识别任务对与 EdgeBoard 计算卡的配置与部署。

5.1.1　连接 EdgeBoard 计算卡

EdgeBoard 计算卡内部运行的系统为 Linux，为了方便部署以及调试，我们采用的连接方式为网络连接，通过网线或者将 EdgeBoard 计算卡与计算机置于同一 WiFi 环境下，用 SSH 连接 EdgeBoard。

网络配置如下：

```
nmcli c modify edgeboard_eth0 ipv4.addr '192.168.137.254/24'
nmcli c modify edgeboard_eth0 ipv4.gateway 192.168.137.1
nmcli c modify edgeboard_eth0 ipv4.dns 223.5.5.5
nmcli c up edgeboard_eth0
```

需要注意的是，需要在 Wifi 网络接口下设置共享，共享 Wifi 的网络给网线（使用 192.168.137.x 网段），让 Edgeboard 能用上电脑的网络。

5.1.2　配置 VSCode 用于远程开发

Visual Studio Code 是一个轻量级但功能强大的源代码编辑器，可在桌面上运行，适用于 Windows、macOS 和 Linux。它内置了对 JavaScript、TypeScript 和 Node.js 的支持，并为其他语言（例如 C++、C#、Java、Python、PHP、Go）和运行时（例如 .NET 和 Unity）提供了丰富的扩展生态系统。

为了方便开发调试，我们组通过使用 VSCode 的远程插件 VScode Remote-SSH[10]来连接 Edgeboard 实现了远程代码更改以及调试

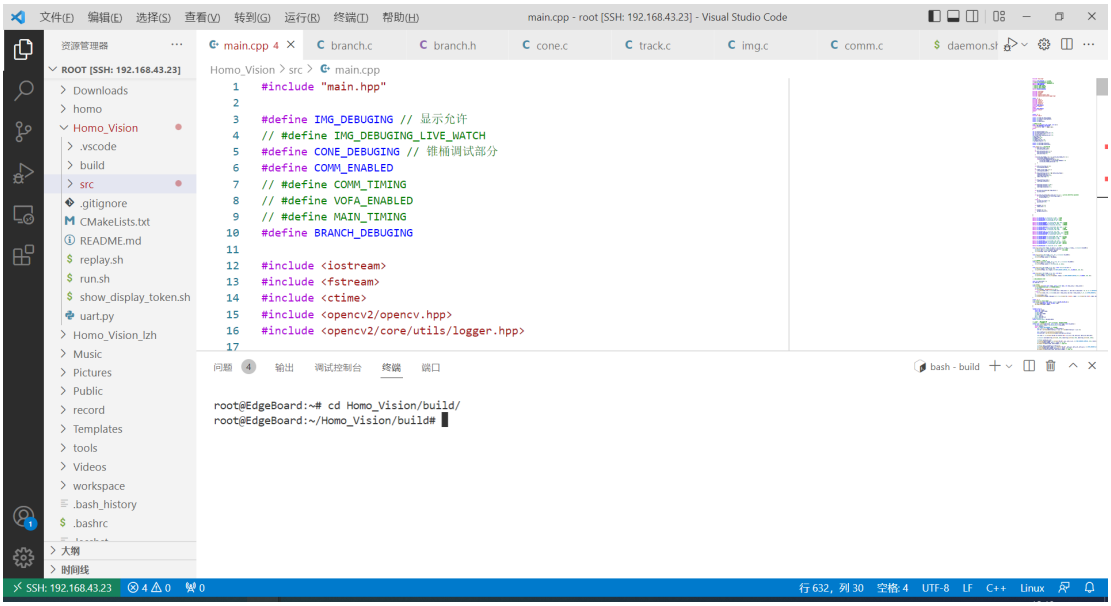


图 5.2　VSCode 远程调试界面

5.2　摄像头控制处理流程

第十七届智能车百度完全模型组赛道分为传统元素和特殊元素两部分，以下主要介绍我们组在赛道传统元素方面的识别以及补线处理策略。

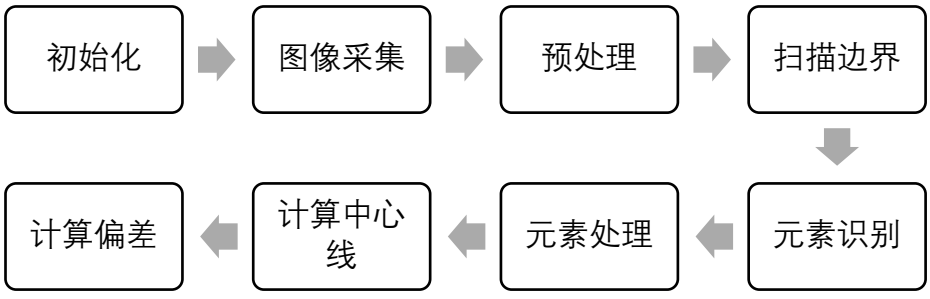


图 5.3　摄像头处理流程

5.2.1　摄像头初始化

摄像头初始化的过程 opencv 提供了现成的接口，但是由于性能问题，我们组并没有使用。我们组采用的方案是使用 v4l2 从底层实现了摄像头初始化和图

像采集逻辑，并且使用 pthread 使他以一个独立线程的方式运行。设置一个线程让摄像头采集过来的数据首先存储到缓冲区中，再从主线程中一帧一帧取走缓冲区中的数据。这样做的好处是不管 main 函数中发生了什么，采集图像过程会一直进行。

5.2.2　图像采集

首先将数据采集到内存缓存中，再通过 cv::imdecode()函数将数据解码成图像格式，图像采集帧数设置为 120 帧。另外为了方便之后的图像处理以及特征提取，我们对于摄像头的曝光度对比度等参数进行了一些更改。图像采集过程中需要注意的是，第一帧图像往往会出现问题，所以舍弃第一帧图像。

5.2.3　预处理

图像采集返回的原始图像对于赛道识别任务来说分辨率偏高，会增加一些不必要的运算量上的负担。为了提升运算速度，我们通过使用 opencv 中的 resize 和裁剪等功能从原始画面中提取出我们想要的部分，这样既达成了减少计算量的目的，又减少了原始图像中无用信息对于识别的干扰。

在对于图像进行简单的裁剪之后，我们将采集处理好之后的图像转换成灰度图像并将其 cv::Mat 类转换成 unsigned char 类型数组，集中对于灰度图像数组进行处理。对于传统元素循迹来说，直接使用灰度数组进行处理有一定难度并且会增加不必要的运算负担，所以我们组决定先对图像进行二值化处理。

图像二值化（ Image Binarization）就是将图像上的像素点的灰度值设置为 0 或 255，也就是将整个图像呈现出明显的黑白效果的过程。在数字图像处理中，二值图像占有非常重要的地位，图像的二值化使图像中数据量大为减少，从而能凸显出目标的轮廓。在对于赛道的二值化过程中，赛道部分被识别为前景，蓝布部分被识别为背景，为之后的扫线以及元素特征的提取提供了良好条件。

其中在二值化算法的选择上，opencv 库中提供了计算阈值以及二值化的接口，但是因为出于算法时间上的考量，我们还是选择了通过 c 语言实现大津法[1]以及二值化的过程。在做图像二值化计算时，需要注意的是，大津法对于光线条件的要求较高，反光，太阳直射等情况会极大影响算法的效果。为了应对光线条

件较差的情况，我们队采用了在摄像头上添加偏光片等措施，获得了较好的效果。

5.2.4 扫描边界

智能车在赛道中行驶时，获取到的图像特征相对单一，所以我们组选用八邻域扫线[9]的方式来提取赛道的基础边线。首先，从最底行往上扫直道获取有效底边，从有效底边往两边扫描赛道边界获得种子点，并从扫描到的种子点开始执行八邻域算法开始扫描边界。八邻域跟踪算法基本思想是，从种子点 x 开始，顺时针查找该点八邻域内遇到的第一个边界点 x'。令 x=x'，再进行邻域内的顺时针查找，查找的起点为刚刚 x 到 x'过程中 x'前一个边界点。

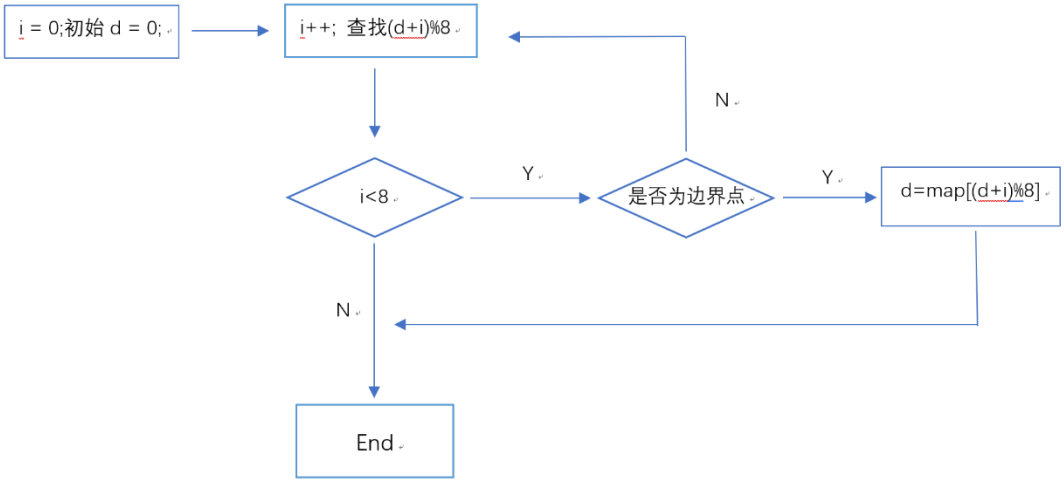


图 5.4 八邻域执行流程

以此类推，一直搜索下去直到搜索到图像顶端为止，得到的就是左右边界。

5.2.5 基于 EdgeBoard 的可视化调试方案

由于本届智能车完全模型组识别任务需要运行在 EdgeBoard 计算卡上，而 EdgeBoard 计算卡外接显示屏比较困难。将 EdgeBoard 计算卡识别后的图像传输到单片机上通过 ips 显示屏输出的方式又有占用资源多，通讯负担大等问题。所

以我们组采用了通过网络远程连接 EdgeBoard 并使用 TightVNC 来显示识别结果的方案。经测试，此方案延迟较小，且能很好的满足调试的可视化需求。

使用 VNC 远程桌面时需要注意，需要在 bash 任意目录下开启 VNC 服务并让其输出显示在当前 VNC 桌面上

```
root@EdgeBoard:~# vncserver

New 'X' desktop is EdgeBoard:1

Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/EdgeBoard:1.log

root@EdgeBoard:~# export DISPLAY=":1.0"
```

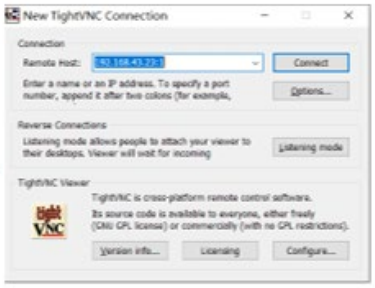


图 5.5　连接 VNC 过程

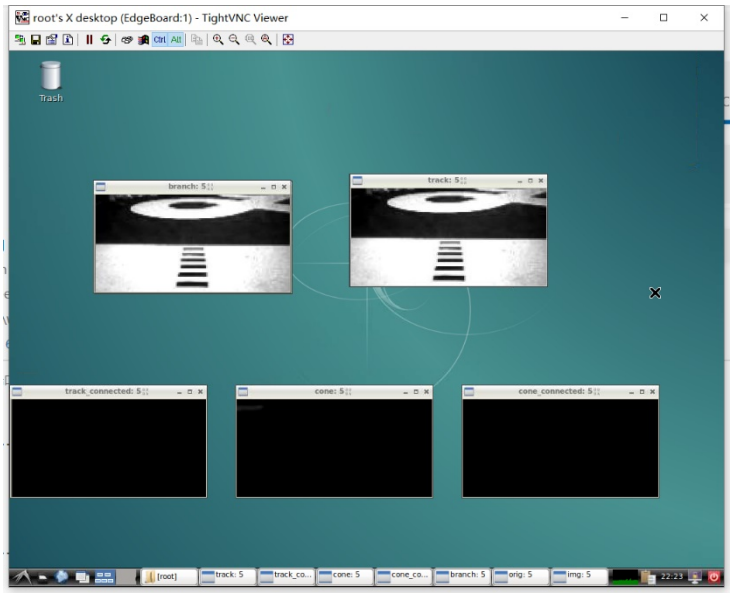


图 5.6　VNC 调试界面

对于识别结果的可视化输出，我们采用的方案是在识别处理任务完成后，生成一张新的彩色画布用来显示读取到的图像以及当前的识别结果，其中我们组通过 opencv 库中的接口实现了画线画点等函数，用于在输出画布上显示识别结果。

```
// 转换灰度化的图片为画布
cv::Mat frame_show(IMG_H, IMG_W, CV_8UC1, img);
cv::cvtColor(frame_show, frame_show, cv::COLOR_GRAY2BGR);
```

图 5.7　生成新的显示画布

```
void draw_line(cv::Mat frame, u8 start_x, u8 start_y, u8 end_x, u8 end_y, cv::Scalar CV_COLOR) {
    cv::Point start = cv::Point(start_x, start_y); //直线起点
    cv::Point end = cv::Point(end_x, end_y);       //直线终点
    cv::line(frame, start, end, CV_COLOR);
}

void draw_point(cv::Mat frame, u8 x, u8 y, cv::Scalar CV_COLOR) {
    cv::Point point = cv::Point(x, y);
    cv::circle(frame, point, 1, CV_COLOR);
}

// 以中心点坐标和半径画圆
void draw_circle(cv::Mat frame, u8 x, u8 y, u8 r, cv::Scalar CV_COLOR) {
    cv::Point point = cv::Point(x, y);
    cv::circle(frame, point, r, cv::Scalar(0, 0, 255));
}

void draw_str(cv::Mat frame, u8 x, u8 y, const std::string &str) {
    cv::Point origin = cv::Point(x, y);
    cv::putText(frame, str, origin, cv::FONT_HERSHEY_SIMPLEX, 0.4, CV_RGB(255, 230, 0));
}

void draw_num(cv::Mat frame, u8 x, u8 y, int num) {
    cv::Point origin = cv::Point(x, y);
    cv::putText(frame, std::to_string(num), origin, cv::FONT_HERSHEY_SIMPLEX, 0.4, CV_RGB(0, 255, 0));
}
```

图 5.8　基于 opencv 实现的画图函数

上述功能使得我们可以远程查看小车运行时图像，极大的方便了调试过程。

5.2.6　元素的识别与处理

5.2.6.1　直道

直道是传统循迹中最为简单的元素，直道元素左右高度对称，且左右有效边界数多，且斜率较为一致，对于直道不需要进行其他过多处理，直道元素的中线位置可直接由左右边界的位置相加除以二得出。

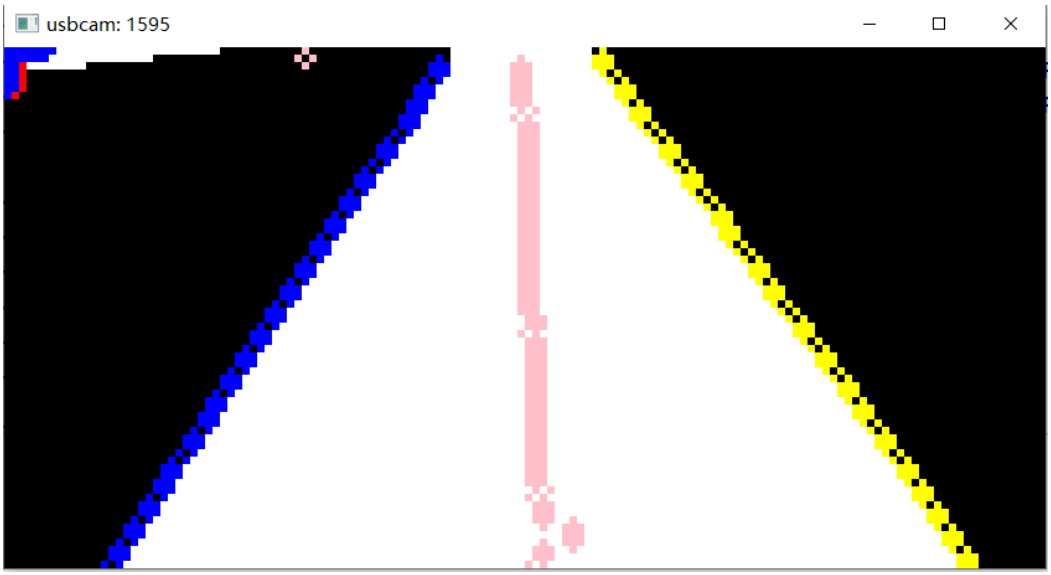


图 5.9　直道

5.2.6.2　普通弯道

普通弯道也是传统元素中的一个重要元素，其特征为一侧边线出现严重的信息丢失情况，此时不能再使用左右边界相加除以二来计算中线，应该针对丢失信息的情况进行补线处理。由于经过弯道元素时赛道与智能车当前的行进方向成一定角度，所以我们组选择通过赛道宽度和智能车与赛道的夹角来计算出当前角度画面中的赛道宽度，以此为依据进行弯道补线处理，补线效果如图所示。

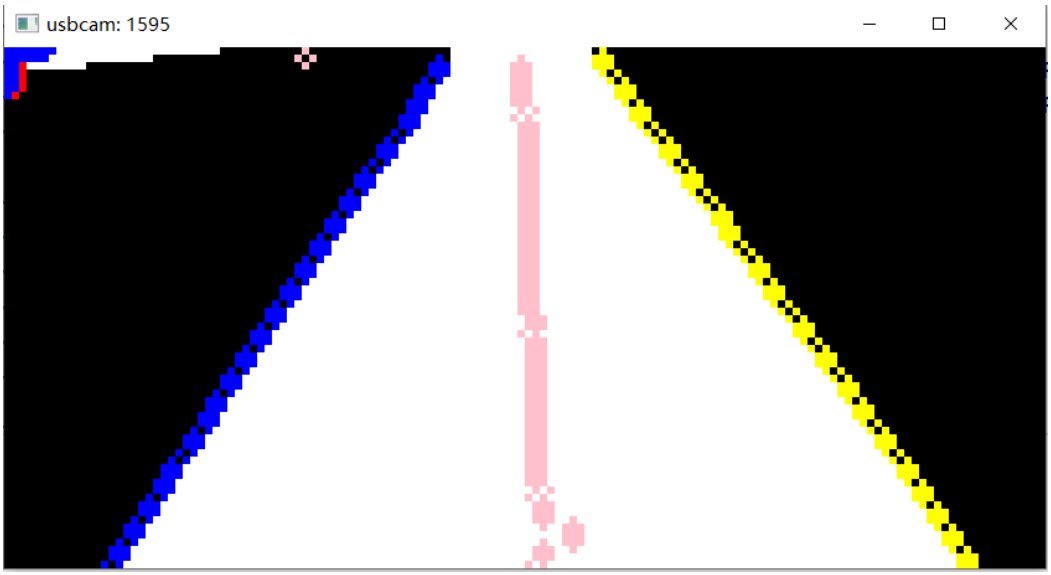


图 5.10　直道

5.2.6.3　十字

对于十字的识别，我们组采用的方案是，检测到两侧左右边线丢失信息，并且扫描到拐点，此刻判定为十字元素。十字元素的识别较为复杂，因为存在多种情况，例如在十字前可以查找到四个拐点，且左右边线丢失信息较少，在十字中只能查询到两个拐点，左右丢线数较多。同时十字还存在姿态不正时入十字补线较为困难，易识别成三岔等问题，以上问题都需要分类讨论，特殊处理。在十字前我们采用的方案是查询四个拐点并利用四个拐点的位置和拐点附近的斜率进行补线。在十字中只能查询到两个拐点时我们采用的方案是利用左右两个上拐点位置及其附近斜率进行补线。

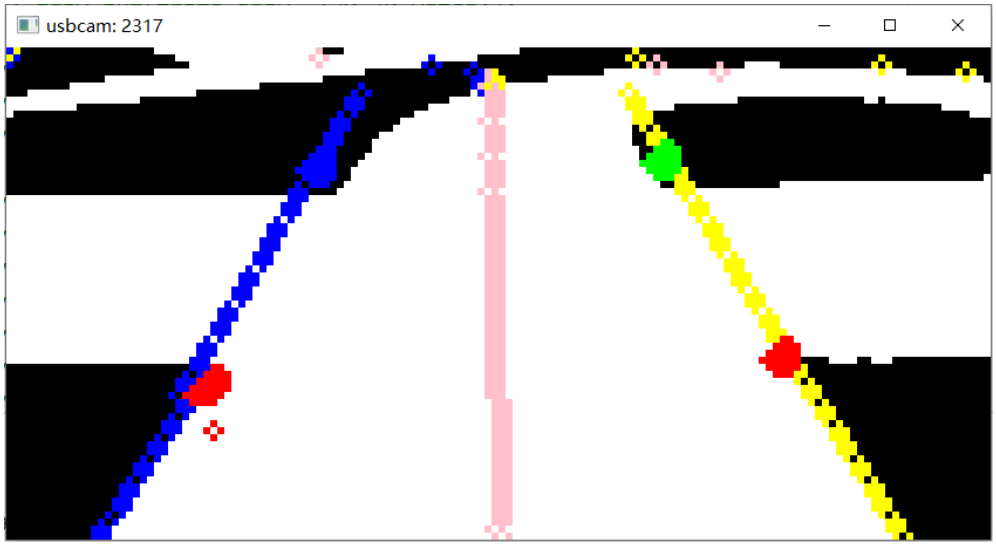


图 5.11　在十字前的补线状况

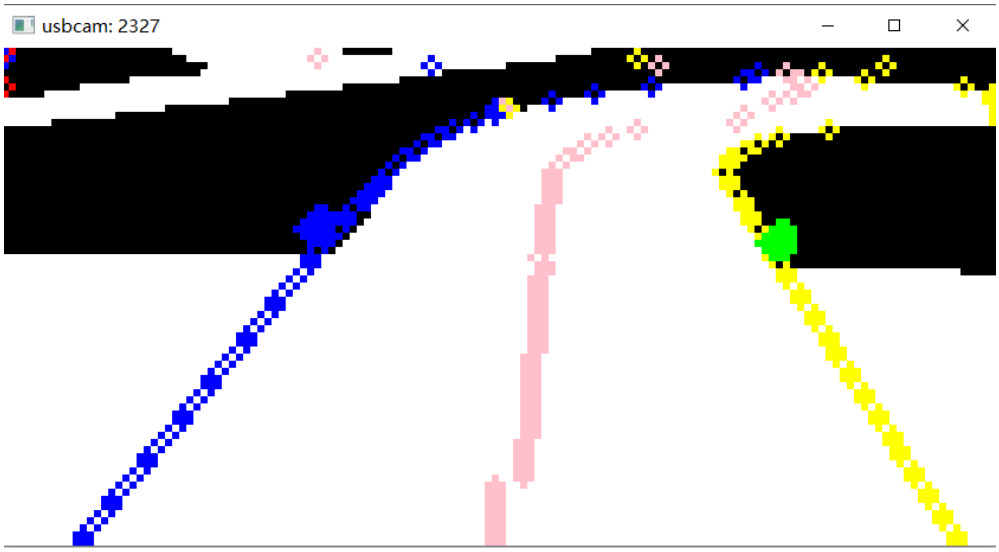


图 5.12　在十字中的补线情况

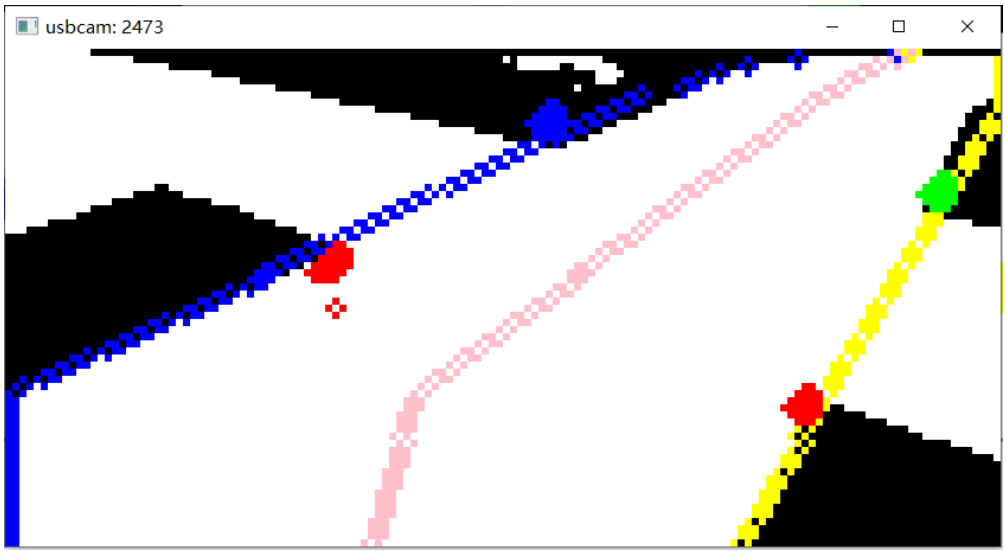


图 5.13　姿态不正时十字的处理

5.2.6.4　三岔

第十七届智能车完全模型组的三叉识别较为简单，因为有地标辅助识别。我们组采取的方案是在 AI 识别线程返回信息表示已识别到三岔标志且三岔标志位于图像一定区域内时，即判定为识别到三岔，此时开始检测几个补线点并且开启禁行标志的检测，进行泛行区内的路径规划。

5.2.6.5　环岛

环岛作为传统元素中难度较高的元素，其实也具有较为明显的特征。环岛的一侧是直道，另一侧为圆环，通过以上特征判定为环岛之后开始补线。因为入环岛时在图像上体现为直道和入环岛两条岔路，此时八邻域扫线会出现问题。所以判定为入环岛之后需要重新按照想要的路径进行八邻域扫线，同时补出入环岛的路径线。

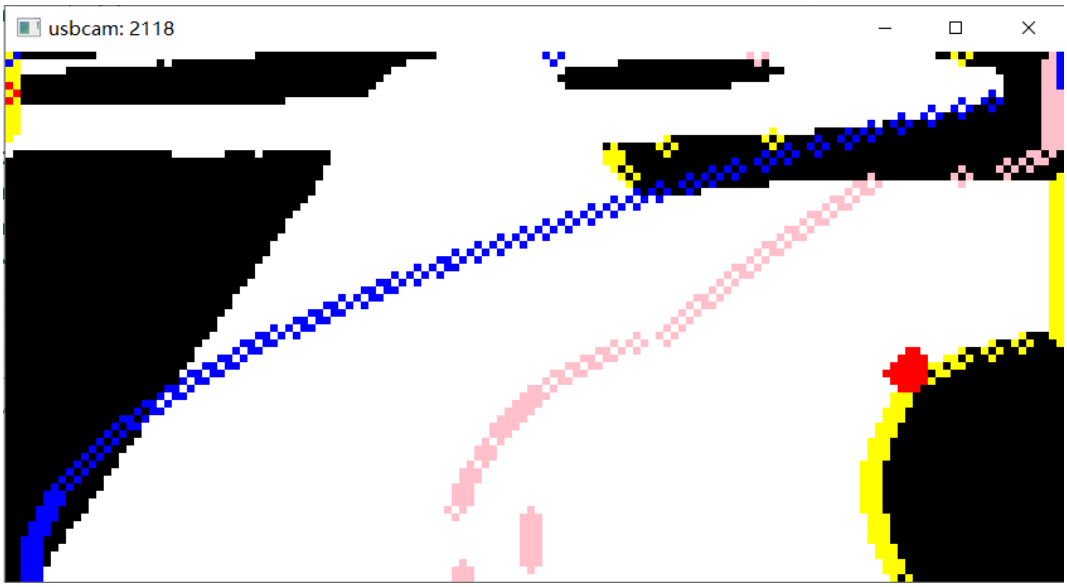


图 5.14　入环岛补线

5.2.6.6　车库

对于车库的识别与补线，我们组采取的方案是第一圈识别到车库进行屏蔽并补出正常直行中线，第二圈识别到车库后进行固定打角停车。对于斑马线的识别我们组采用的方案是指定检测行并扫描该行的黑白跳变数。如果该行的黑白跳变数大于一定值则判断为斑马线。

图 5.15　第一次经过车库时的补线

本次完全模型组的特殊之处在于赛道上会贴有施工区、三岔路等赛道标志，这些赛道标志在二值化图像中呈现黑色，在识别车库算法中较易与车库混淆。我们采用的方案是只用一行检测行来识别车库，并对赛道标志处和斑马线处的黑白跳变数进行多次采样，设置一个可靠的阈值，黑白跳变数大于这个阈值时才判定为斑马线。经过实践检验，该方案较为可靠，既解决了赛道标志误识别为斑马线的问题，又没有降低斑马线本身的识别率。

5.2.7　计算中心线与偏差

在经过元素识别与补线之后，如果得到了可靠的左右边线，中心线便可由左右边线之和除以二求得。如果遇到例如弯道和环岛内这种左右边线丢失信息较多，且难以还原的情况，我们组采取的方案是利用赛道的宽度以及丢失信息较少一侧的边线补线求得中线。

对于偏差的计算，我们尝试过使用一行或多行中线数据计算偏差来确定打角。我们发现，如果采用多行信息来计算偏差值，会因为前后文原因使舵机打角并不准确，所以我们最终采用的方案是使用特定的一行作为偏差计算行和打角行。

对于出库入库这种场景我们使用的是固定打角的方式。智能车出库时给定偏差值让舵机打角。第二次检测到斑马线时给定固定的偏差值入库。由于出入库识别准确率较高且位置较准，经过细微调整后可使得出入库路径较为准确。

5.3　与 TC264 的通讯

5.3.1　通讯协议的设计

完全模型组的任务要求是 EdgeBoard 计算卡直连摄像头，这意味着只能是 EdgeBoard 计算卡来做决策然后将信息发送给 TC264 进行控制和输出。这个结构的基础上，通讯协议的稳定性就变得至关重要，通讯协议是否可靠将直接关系到智能车的运行状态。我们组通讯协议的设计分为物理层和协议层。

物理层包括了对 libserial 库的再次封装。分为三个函数，分别为对串口的初始化函数、阻塞发送函数以及轮询接收函数。其中串口初始化函数包括打开串口，设置波特率、数据位、流控、以及校验位和停止位。如果串口打开失败则返回错误信息。阻塞发送函数调用了 libserial 库的写数据到串口的接口并且等待缓冲区耗尽然后返回。如果出现错误则返回错误信息。非阻塞式接收函数调用了 libserial 库从串口中读数据的接口，并且作为非阻塞式接收，如果没有接收到数据或者发生错误则返回错误。

我们组在协议层的大体设计思路为向 264 发送数据时采用阻塞式发送，也就是等待系统缓冲区将所有数据发送完毕后再进行其他逻辑的处理。接收数据为非阻塞式接收，也就是查询当前是否有数据，如果有数据则接收，如果没有数据则返回。具体的协议设计为，发送时将数据打包成数据包，数据包格式为包头加数据大小加数据本身加校验位。在接收数据时，查询是否接收到包头，如果接收到包头则以下一个字节作为数据包大小来读取和解析数据。其中我们还设置了校验位和超时退出，来保证通讯协议的稳定性。

5.3.2　通讯数据的选择

通讯数据的选择决定了整个系统的总体决策与执行架构。出于通讯时间和通讯协议稳定性上的考量，我们组的设计是：EdgeBoard 计算卡在收集图像信息

并作出判断和决策后，向 TC264 发送两个 unsigned char 型数据，其中一个数据为舵机打角的控制量，另一个数据为赛道当前元素类型（例如出库入库，环岛等特殊元素，由双方共同约定标志位）。TC264 则向 EdgeBoard 计算卡发送当前周期编码器获得数据以辅助决策。这种通讯结构的好处首先是双方互相交换信息数量较少，耗时较短。其次这种通讯结构基本可以完全满足执行层和决策层的所有需求，并且较为简洁，稳定性好。

```
// 不同类型的数据包大小
int comm_payload_size[] = {
    1, // COMM_TYPE_PING
    1, // COMM_TYPE_PONG
    3, // COMM_TYPE_UPDATE_TO_TC264
    4, // COMM_TYPE_UPDATE_FROM_TC264
    0, // COMM_TYPE_HELLO_FROM_TC264
};
```

图 5.16　通讯过程中数据包大小的设置

第六章　控制原理与策略介绍

6.1　PID 部分

6.1.1　原理部分

PID 即：Proportional（比例）、Integral（积分）、Differential（微分）的缩写。顾名思义，PID 控制算法是结合比例、积分和微分三种环节于一体的控制算法，它是连续系统中技术最为成熟、应用最为广泛的一种控制算法，该控制算法出现于 20 世纪 30 至 40 年代，适用于对被控对象模型了解不清楚的场合。实际运行的经验和理论的分析都表明，运用这种控制规律对许多工业过程进行控制时，都能得到比较满意的效果。PID 控制的实质就是根据输入的偏差值，按照比例、积分、微分的函数关系进行运算，运算结果用以控制输出。

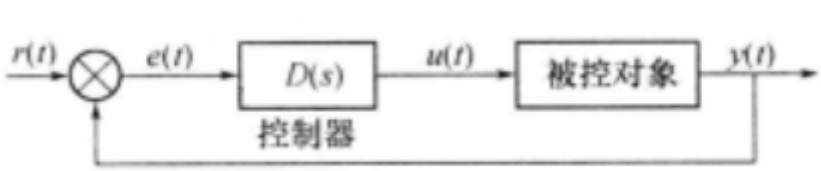


图 6.1　PID 控制图示

比例控制器的输出 u(t)与输入偏差 e(t)成正比，能迅速反映偏差，从而减小偏差，但不能消除静差，比例控制作用的大小除与偏差 e(t)有关之外，还取决于比例系数 Kp 的大小。比例系数 Kp 越小，控制作用越小，系统响应越慢；反之，比例系数 Kp 越大，控制作用也越强，则系统响应越快。但是，Kp 过大会使系统产生较大的超调和振荡，导致系统的稳定性能变差。积分环节的作用，主要用于消除静差提高系统的无差度。积分作用的强弱，取决于积分时间常数 Ti，Ti 越大积分作用越弱，反之则越强。积分控制作用的存在与偏差 e(t)的存在时间有关，只要系统存在着偏差，积分环节就会不断起作用，对输入偏差进行积分，使控制

器的输出及执行器的开度不断变化，产生控制作用以减小偏差。微分环节的作用能反映偏差信号的变化趋势（变化速率），并能在偏差信号的值变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减小调节时间。积分控制作用的引入虽然可以消除静差，但是降低了系统的响应速度，特别是对于具有较大惯性的被控对象，用 PI 控制器很难得到很好的动态调节品质，系统会产生较大的超调和振荡，这时可以引入微分作用。

6.1.2 舵机位置式 PD 控制

对于舵机的控制，我们选择了闭环控制中的位置式 PD 控制，多次采集误差后进行比例与微分的算法然后给出占空比的值使用 PWM 波控制舵机，误差由打角行的中线值与当前屏幕中间值作差得到。首先介绍位置式 PID 的原理

位置式 PID 是当前系统的实际位置，与你想要达到的预期位置的偏差，进行 PID 控制

因为有误差积分 ∑e(i)，一直累加，也就是当前的输出 u(k)与过去的所有状态都有关系，用到了误差的累加值；（误差 e 会有误差累加），输出的 u(k)对应的是执行机构的实际位置，一旦控制输出出错(控制对象的当前的状态值出现问题)，u(k)的大幅变化会引起系统的大幅变化

并且位置式 PID 在积分项达到饱和时,误差仍然会在积分作用下继续累积，一旦误差开始反向变化，系统需要一定时间从饱和区退出，所以在 u(k)达到最大和最小时，要停止积分作用，并且要有积分限幅和输出限幅

所以在使用位置式 PID 时，由于位置式 PID 积分项的滞后性一般我们直接使用 PD 控制，而位置式 PID 适用于执行机构不带积分部件的对象，如舵机。

$$u_k = \mathrm{Kp} * e_k + \mathrm{Ki} \sum_{j=0}^{k} e_j + \mathrm{K}d(e_k - e_{k-1})]$$

其中：k——采样序号，k=0,1,2，……；

u_k——第 k 次采样时刻的计算机输出值；

e_k——第 k 次采样时刻输入的偏差值；

e_{k-1}——第 k-1 次采样时刻输入的偏差值；

Ki——积分系数，Ki=Kp*T/Ti；

Kd——微分系数，Kd=Kp*Td/T；

图 6.2　位置式 PID

在实际调试中我们发现，在不同曲率的弯道或者直道中，采用不同的 PD 参数可以取得很好的控制作用，所以我们针对不同元素进行了不同的 PD 参数调节，在车的高速与低速的控制中相同的 PD 参数也会由不同的表现，所以我们根据当前速度的大小选择动态调节 PD 参数，从而达到理想的控制效果。

舵机位置式 PD 的调节，我们选择了经验调参，根据初步整定参数后的表现以及实际需求进行进一步整定，首先将微分项置为 0，调节比例项，在达到稳定可行的控制效果后再进行调节微分项，从而使车达到理想中的走线效果。

查看参数的控制效果，最简单的方法是直接通过经验观察，决定 PD 参数是否应该继续增加或者减小，也可以通过观察上位机图像的中线来进行下一步的调参。

以下为我们整定 PID 时使用的口诀：

参数整定找最佳，从小到大顺序查
先是比例后积分，最后再把微分加
曲线振荡很频繁，比例度盘要放大
曲线漂浮绕大湾，比例度盘往小扳
曲线偏离回复慢，积分时间往下降
曲线波动周期长，积分时间再加长

曲线振荡频率快，先把微分降下来
动差大来波动慢，微分时间应加长
理想曲线两个波，前高后低四比一
一看二调多分析，调节质量不会低

6.1.3 电机增量式 PI 控制

增量式 PID 控制主要是通过求出增量，将原先的积分环节的累积作用进行了替换，避免积分环节占用大量计算性能和存储空间。

增量式PID控制

根据位置式PID控制公式，写出n-1时刻的控制量:

$$u[n-1]=K_p\left\{e[n-1]+\frac{T}{T_i}\sum_{i=0}^{n-1}e[i]+\frac{T_d}{T}\{e[n-1]-e[n-2]\}\right\}$$

设

$$\Delta u[n]=u[n]-u[n-1]$$

得到

$$\Delta u[n]=K_p\{e[n]-e[n-1]\}+\frac{K_pT}{T_i}e[n]+\frac{K_pT_d}{T}\{e[n]-2e[n-1]+e[n-2]\}$$

令$K_i=K_p\frac{T}{T_i}$为积分系数；$K_d=K_p\frac{T_d}{T}$为微分系数，可以将上式简化为

$$\Delta u[n]=K_p\{e[n]-e[n-1]\}+K_ie[n]+K_d\{e[n]-2e[n-1]+e[n-2]\}$$

图 6.3 增量式 PID

增量式 PID 控制的主要优点为：

（1） 算式中不需要累加。控制增量 Δu(k)的确定仅与最近 3 次的采样值有关，容易通过加权处理获得比较好的控制效果；

（2） 计算机每次只输出控制增量，即对应执行机构位置的变化量，故机器发生故障时影响范围小、不会严重影响生产过程；

增量式 pid 调节目标速度时候参数整定：

先加大 KI，这时候会越来越接近实际速度，当 KI 过大的时候，在切换目标速度的时候，就会抖动，这时候就是 KI 大了响应速度高了，但导致超调量增加，这时候就加大增量式的 KP，来缓减抖动，减小超调量。

在调节电机 PI 时，为了方便查看响应时间与控制效果，可以使用 VOFA 进行图像观察。

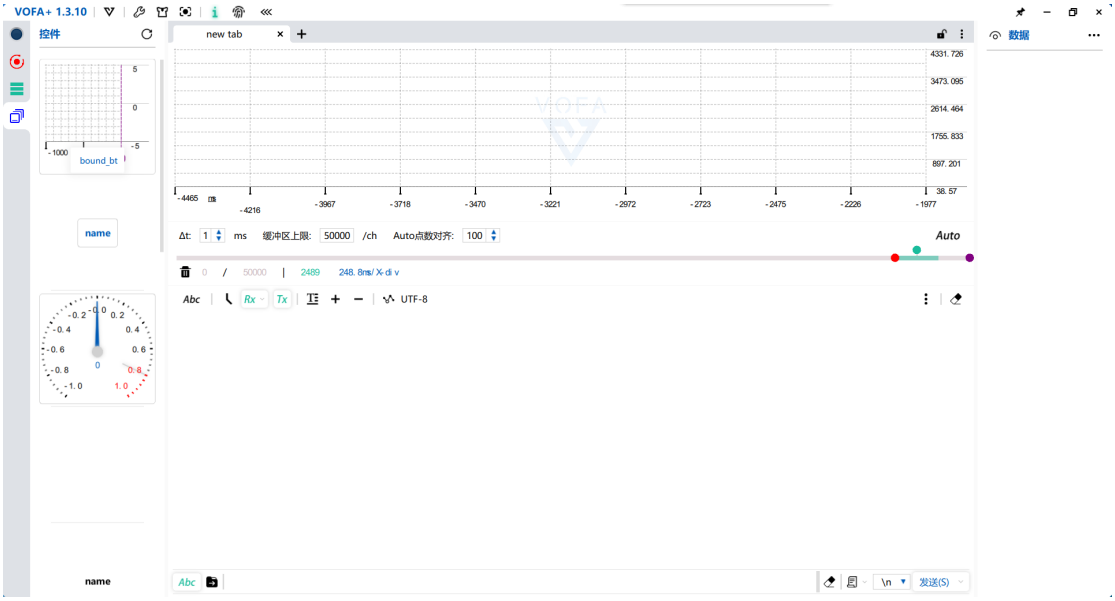


图 6.4　VOFA

将编码器的实际测量值与目标值绘在同一个图片上就能直观的观察参数是否合适。

在使用 VOFA 时，小车主板这里需要进行无线串口的配置，以逐飞库为例，无线串口部分在 SEEKFREE_WIRELESS.C 下。

调用初始化与发送函数就可以向 VOFA 发送需要显示的数据。逐飞库提供的初始化与发送函数为

```
void seekfree_wireless_init(void);
uint32 seekfree_wireless_send_buff(uint8 *buff,uint32 len);
```

同时要注意保持 VOFA 与主板通讯时波特率一致。

无线转串口使用的是逐飞配套的无线模块，如下图所示。

图 6.5　无线模块套件

为了方便调试参数，同时也可以将参数投放到屏幕上进行显示与修改。该部分的详细论述会在 RT-Thread 在调试中的应用一章中进行

6.2　控制策略

6.2.1　整体部分

对于小车对赛道的数据采集以及输出 PWM 进行控制，我们的整体方案为使用摄像头进行赛道原始数据的获取，然后在 edgeboard 上进行处理，将处理后的结果发送给 TC264，最后由 TC264 进行最终数据的输出。

对于微处理器，官方给出了下图示的单片机，我们最终选择了 TC264 作为我们车模进行处理的单片机：

（1）Infineon微控制器

- TC264，TC212
- TC377，TC364
- 此外还允许使用Infineon出品的Aurix™系列TC2XX和TC3XX其它型号的单片机

图 6.6　官方指定单片机

以下为详细的选型手册，通过手册可以看出它有 4 组 ASCLIN 接口，4 组 QSPI 接口，2 组 MSC 接口等等。

Feature overview AURIX™ family

TriCore™ microcontroller

Product type	Max. clock frequency [MHz]	Program memory [KByte]	SRAM (incl. cache) [KByte]	Co-processor [1]	Cores/lockstep	Timed I/O GPT/IO	Number of ADC channels	External bus interface	CAN/CAN FD nodes	Communication interfaces [2]	Temperature range [3]	Packages	Additional features/ remarks [4]
AURIX™ TC2xx family													
TC299TX	300	8000	2728	FPU	3/1	263	84/10 DS	Yes	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet	K	LFBGA-516	EVR, WUT, HSM
TC299TP	300	8000	728	FPU	3/1	263	84/10 DS	Yes	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet, CAN FD	K	LFBGA-516	EVR, WUT, HSM
TC298TP	300	8000	728	FPU	3/1	232	60/10 DS	Yes	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet	K	LBGA-416	EVR, WUT, HSM
TC297TA	300	8000	2728	FPU, FFT, CIF	3/1	169	60/10 DS	No	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet	K	LFBGA-292	EVR, WUT, HSM
TC297TX	300	8000	2728	FPU	3/1	263	60/10 DS	No	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet	K	LFBGA-292	EVR, WUT, HSM
TC297TP	300	8000	728	FPU	3/1	169	60/10 DS	No	6	4x ASCLIN, 6x QSPI, 3x MSC, 2x I²C, 15x SENT, HSSL, 5x PSI5, 2x FlexRay, Ethernet, CAN FD	K	LFBGA-292	EVR, WUT, HSM
TC277TP	200	4000	472	FPU	3/2	169	60/6 DS	No	4	4x ASCLIN, 4x QSPI, 2x MSC, HSSL, I²C, 10x SENT, 3x PSI5, FlexRay, Ethernet, CAN FD	K	LFBGA-292	EVR, WUT, HSM
TC275TP	200	4000	472	FPU	3/2	112	48/6 DS	No	4	4x ASCLIN, 4x QSPI, 2x MSC, HSSL, I²C, 10x SENT, 3x PSI5, FlexRay, Ethernet, CAN FD	K	LQFP-176	EVR, WUT, HSM
TC267D	200	2500	240	FPU	02/1	169	50/3 DS	No	5	4x ASCLIN, 4x QSPI, 2x MSC, I²C, 10x SENT, 3x PSI5, HSSL, FlexRay, Ethernet, CAN FD	K	LFBGA-292	EVR, WUT
TC265D	200	2500	240	FPU	2/1	112	50/3 DS	No	5	4x ASCLIN, 4x QSPI, 2x MSC, I²C, 10x SENT, HSSL, 3x PSI5, FlexRay, Ethernet, CAN FD	K	LQFP-176	EVR, WUT
TC264DA	200	2500	752	FPU, FFT, CIF	2/1	88	40/3 DS	No	5	4x ASCLIN, 4x QSPI, 2x MSC, I²C, 10x SENT, HSSL, 3x PSI5, FlexRay, Ethernet, CAN FD	K	LQFP-144	EVR, WUT
TC264D	200	2500	240	FPU	2/1	88	40/3 DS	No	5	4x ASCLIN, 4x QSPI, 2x MSC, I²C, 10x SENT, HSSL, 3x PSI5, FlexRay, Ethernet, CAN FD	K	LQFP-144	EVR, WUT
TC237LP	200	2000	192	FPU	1/1	120	24	No	6	2x ASCLIN, 4x QSPI, 4x SENT, FlexRay, CAN FD	K	LFBGA-292	EVR, WUT, HSM
TC234LA	200	2000	704	FPU, FFT	1/1	120	24	No	6	2x ASCLIN, 4x QSPI, 4x SENT, FlexRay, Ethernet	K	TQFP-144	EVR, WUT, HSM
TC234LX	200	2000	704	FPU	1/1	120	24	No	6	2x ASCLIN, 4x QSPI, 4x SENT, FlexRay, Ethernet	K	TQFP-144	EVR, WUT, HSM
TC234LP	200	2000	192	FPU	1/1	120	24	No	6	2x ASCLIN, 4x QSPI, 4x SENT, FlexRay, CAN FD	K	TQFP-144	EVR, WUT, HSM
TC233LP	200	2000	192	FPU	1/1	78	24	No	6	2x ASCLIN, 4x QSPI, 4x SENT, FlexRay, CAN FD	K	TQFP-100	EVR, WUT, HSM
TC224L	133	1000	96	FPU	1/1	120	24	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-144	EVR, WUT
TC223L	133	1000	96	FPU	1/1	78	24	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-100	EVR, WUT
TC222L	133	1000	96	FPU	1/1	59	14	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-80	EVR, WUT
TC214L	133	500	96	FPU	1/1	120	24	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-144	EVR, WUT
TC213L	133	500	96	FPU	1/1	78	24	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-100	EVR, WUT
TC212L	133	500	96	FPU	1/1	59	14	No	3	2x ASCLIN, 4x QSPI, 4x SENT	K	TQFP-80	EVR, WUT

图 6.7　选型手册

6.2.2 舵机部分

我们针对舵机部分采用的是对不同赛道元素进行不同 PD 控制，在直道部分 PD 较小，在弯道处 PD 较大，同时针对圆环、三岔（以及泛行区）、十字等进行不同的 PD 参数进行控制，当参数整定完成后在直道上不会出现抖动，在弯道上转弯连续，面对比较急的弯道也可以实现基本循中线进行行驶。

6.2.3 电机部分

我们针对电机部分采用的是针对不同赛道元素进行不同目标速度的控制，在直道部分目标速度较大，在弯道部分目标速度较小，同时为了更快的完成比赛，针对不同元素进行了不同的速度控制，使得小车在通过特殊元素时既稳定又快速的通过，同时在没有经过特殊处理的部分的速度进行了弯道减速，使其能自动调整速度来适应普通元素下的弯道。同时不论是特殊处理的元素的速度，还是弯道减速后的普通速度都不应该出现差距过大的情况，避免出现目标速度出现较为剧烈的变化，这样可以保证小车运行的稳定性和连贯性。

第七章　多种并行任务的挑战与 RT-Thread 的选择

在本次比赛中，我们很幸运的遇到了多种并行处理环境，包括：

- 线上赛中 Python 环境下的多进程
- EdgeBoard 中 Linux 环境下 C/C++混编的多线程
- TC264 上单片机环境下的多线程

对于不同环境下的并行处理，我们遇到了许多问题，在本章中，会介绍遇到的这些问题，并且论述这些问题对 RT-Thread 的采用带来的影响。

7.1　线上赛中 Python 环境下的多进程

线上赛打榜的初期，选择的模型运算速度太慢，遇到了无法在要求的 FPS 下完成比赛的问题。对于此，在本地测试的过程中发现，单线程执行的代码大多数时间都消耗在给 GPU 传递数据，以及数据的预处理与后处理中，模型本身的处理并不多，导致 GPU 使用并不饱和。即要么 CPU 忙，要么 GPU 忙。此种情况是对处理资源的浪费，为了解决这个问题，引入了并行处理。

Python 提供有 multithreading 多线程模块，但是由于 Python 拥有 GIL（全局解释器锁）[1]，致使任何时刻仅有一个线程在执行。multiprocessing 多进程模块是 Python 下基于进程的并行方案。该方案可以避免 GIL，使 Python 下并行没那么鸡肋。经过实测使用 multithreading 多线程模块，CPU 占用率仅为 70%左右，而采用 multiprocessing 多进程模块，CPU 占用率可以达到 100%。让模型实现了从无法完成比赛到可以计算成绩的跨越式飞跃。

不像线程，进程间资源独立，不容易共享数据，进程间通信需要使用 Python 提供的消息管道进行。该案例也体现了，**并非针对性能优化的语言，并行处理往往需要做特殊处理，并且常伴随有坑的出现**。

[1] GIL 并不是所有解释器都存在，这里解释器为默认环境下 CPython [5]

```python
# 创建进程
# 进程太少会有许多进程堵在 io 影响整体处理进度
# 经验谈:python线程就是shit GIL一时爽,速度火葬场 人生苦短，我用进程
# 进程太多会导致爆内存程序卡住
# paddle 很耗内存，import 一下占1g内存，初始化一下占1g内存，预测一下再占2g内存，故每个进程最少用4g
batch_size = 8
process_num = 4
input_queue = mp.Queue()
output_queue = mp.Queue()
finish_counter = mp.Value('i', 0)
processes = []
for _ in range(process_num):
    p = mp.Process(target=predict_process, args=(input_queue, output_queue, finish_counter, len(images)), daemon=True)
    p.start()
    processes.append(p)

def stopnow():
    # 给结束信号
    for p in processes:
        p.terminate()

    os._exit(1)

# 15 分钟强制结束
timer = threading.Timer(15 * 60, stopnow)
timer.start()

# 喂数据
for start_index in range(0, len(images), batch_size):
    image_paths = images[start_index:start_index+batch_size]
    input_queue.put(image_paths)

# 给结束信号
for p in processes:
    input_queue.put(None)
```

图 7.1　Python 下并行处理代码

7.2　EdgeBoard 中 Linux 环境下 C/C++混编的多线程

EdgeBoard 下，采集图像，使用传统方案处理图像，以及使用 AI 处理图像部分，为了尽可能高的利用 EdgeBoard 的多核优势，我们采取了 pthread 作为并行处理方案。pthread 是 POSIX 的线程标准，定义了创建和操作线程的一套 API。[2]

```
335         pthread_mutex_init(&new_image_meta_lock, NULL);
336
337         // 这个线程会不断取回图像（不管图像用或者没有用），
338         // 以尽量清空缓冲区，使得获取的图像最新
339         // TODO 提升该线程优先级
340         // ref: https://sites.google.com/site/myembededlife/Home/applications--d
341         int ret = pthread_create(&tid, NULL, (void *(*)(void *))thread, NULL);
342         assert(ret == 0);
```

```
47  static int new_image_id;
48  static void *new_image_p;
49  static int new_image_size;
50  static bool new_image_flag;
51
52  #ifdef WITH_GNU_COMPILER
53  static pthread_mutex_t new_image_meta_lock;
54  static pthread_cond_t new_image_sig;
55
56  static void process_image(void *p, int size) {
57      pthread_mutex_lock(&new_image_meta_lock);
58      if (new_image_flag) printf("An image ignored, processing too slow\n");
59      ++new_image_id;
60      new_image_p = p;
61      new_image_size = size;
62      new_image_flag = true;
63      pthread_cond_signal(&new_image_sig);
64      pthread_mutex_unlock(&new_image_meta_lock);
65  }
66
67  static pthread_t tid;
68
69  static void thread(void) {
70      char *dev_name = "/dev/video0";
71
```

图 7.2　Linux 上的 pthread 使用

但是线程处理单帧时，尤其在有些时候涉及到文件的读取或写入时，会出现延迟突增[2]的情况，这是 Linux 的特性所致，不可避免，可能会导致小车冲出赛道。由于比赛规则要求，摄像头必须直连到 EdgeBoard 下，但是对于实时处理，**对固定周期有要求的场景下**，摄像头的处理**理论上必须在 RT-Thread 等 RTOS 下进行，以确保时间固定可控**。对于小车这种并不是特别要求稳定性的地方，可能只会冲出赛道，而在交通、工业等领域，一次延迟激增可能会酿成安全上的大祸！

[2] 即文件函数在处理同样大小的数据过程中延时不固定

7.3 TC264 上单片机环境下的多线程

综合上述情况，除开赛题本身以及规则要求外的部分，其他部分可以交给专业的 RTOS RT-Thread 进行处理，由于本赛题要求图像处理与运动控制分开进行，故中间需要对通信有着很高的要求，通信逻辑复杂繁琐且容易出现错误，且错误隐蔽不好排查，将通讯逻辑结合 RT-Thread 使用，单独抽出一个线程处理，可以很大程度的降低代码 bug 发生的概率，节省开发时的心智负担。

不同于常见的 CPU，单片机本身不存在用于实现虚拟内存的 MMU 模块，无法把各个线程的资源通过虚拟内存完全分割开（完全分隔开即为进程）。故也无法运行常规的操作系统，同时结合任务对实时性的要求，需要采用 RT-Thread。

RT-Thread 的详细移植与应用过程将在下文中进行论述。

第八章　RT-Thread 移植原理简要概述

经过 2 年的 RT-Thread 使用，我们对 RT-Thread 的移植与使用，有了一些经验，并感受到了解 RT-Thread 和单片机的底层原理有助于在 RT-Thread 未按照预期运行时解决问题，也有助于实现 RT-Thread 在新设备上的移植。在本章中，会对 RT-Thread 中核心模块的运行的原理予以简要概述。

8.1　RT-Thread 的线程管理原理概述

理论上，一个单核心的处理器，同一时间内，只能运行一个线程，多线程的实现其实是不断切换不同的线程，让不同线程轮流运行。RT-Thread 的线程切换是使用单片机提供的专用于 RTOS 的 SysTick 定时器进行的。SysTick 定时器会以固定的周期不断调用 RT-Thread 的线程调度器，构成了许多时间片，线程调度器根据当前线程运行情况，来决定下一个时间片需要运行的线程，并且在需要的时候通过切换堆栈和相应的寄存器来实现切换线程。

多线程涉及到数据共享时，需要对数据进行加锁处理，以防不同线程同时修改一份数据。为了确保加锁过程中不被打断（原子操作），RT-Thread 会在加锁等操作进行的时候，全局关闭中断，并在结束后开启中断（为了最小化影响对实时性的影响，RT-Thread 的代码已经保证这个过程时间尽可能的短）。

为了移植 RT-Thread，需要熟悉目标单片机的堆栈相关寄存器、切换堆栈用的汇编代码。

8.2　RT-Thread 的内存管理原理概述

不像电脑上的 CPU，单片机不存在虚拟内存单元 MMU，故单片机上 RTOS 内存管理与一般 CPU 会有差别，所有的线程共享一个相同的内存（这里内存指的是 RAM）空间，隔离性会相对较差（不过考虑到单片机一般只会运行厂商自己的代码，所以隔离性其实并不是一个重要的要求）。同时也鉴于此，单片机上的 RTOS 并不支持内存交换等技术，单片机上能用的内存就是真实的物理内存的大小。这要求程序编写的过程需要手动为线程制定栈的大小，便于 RTOS 为

线程分配栈内存。同时，这也要求了移植过程需要熟悉单片机的地址空间布局，如内存大小，和内存所在地址空间的位置。

8.3 RT-Thread 的启动过程概述

一个单片机启动一般由厂商维护的启动程序开始启动，这个启动程序一般会进行状态的初始化，如 bss 段（全局变量）的清零操作等。以去年的 RT1064 为例，芯片上电后，会先运行烧录到芯片中的 BootROM，BootROM 读取芯片启动引脚状态，根据其状态决定程序所在的位置（默认为芯片内部的 ROM），烧录的程序最开始为厂商维护的 Reset_Handler，执行将 bss 段置零，以及配置 flemram 相关步骤，之后会调用 entry 函数（GCC 下），如果没有使用 RT-Thread，GCC 中的默认 entry 函数会调用 main 函数进入程序，如果安装有 RT-Thread，RT-Thread 下维护的 entry 函数会覆盖默认的 entry 函数，其会调用 rtthread_startup，进入真正的 RT-Thread 启动过程。

```
156  #elif defined(__GNUC__)
157  /* Add -eentry to arm-none-eabi-gcc argument */
158  int entry(void)
159  {
160      rtthread_startup();
161      return 0;
162  }
```

图 8.1 RT-Thread 维护的 entry 函数

TC264 也遵循着相似的步骤，使用 TASKING 作为编译器的 ADS 在上电启动后，会默认调用_START 函数，该函数会直接调用_Core0_start，并在_Core0_start 内进行内核初始化的操作[79]。_Core0_start 结束后，会调用 core0_main，并在 core0_main 中调用 rtthread_startup，进入真正的 RT-Thread 启动过程。

```
7782    * Task entry addresses
7783    ======================
7784
7785    + core "mpe:vtc"
7786    +-----------------+
7787    | symbol | _START |
7788    +-----------------+
```

图 8.2　TASKING 链接器产生的 map 文件显示_START 函数地址被放在中断向量表中

rtthread_startup 函数内的内容不同平台内是类似的（体现出了 RT-Thread 方便移植的特性），分别进行板级初始化（调用用户设置的函数，根据主板不同进行时钟、内存等核心配置），线程调度器初始化，默认的两个线程 main 和 idle 初始化，并且运行 main 线程等。

寻找上电后执行的代码和代码的执行过程要求熟悉链接文件。链接文件与汇编文件的写法，一般编译的变化而变化，对于 GCC，链接文件一般以.ld 后缀命令，对于 TASKING，一般以.lsl 后缀命名。此外，链接器产生的 map 文件也是用于调试链接过程的好帮手。

链接过程需要注意的是，要为一些 RT-Thread 功能预留空间，分别为节.rti_fn（用于 RT-Thread 自动初始化），节 FSymTab，节 VSymTab（用于 FinSH）。

```
185     /* for RT-Thread auto init */
186     PROVIDE_HIDDEN (__rti_fn_start = .);
187     KEEP (*(SORT(.rti_fn.*)))
188     KEEP (*(.rti_fn*))
189     PROVIDE_HIDDEN (__rti_fn_end = .);
190     /* for RT-Thread FinSH function register */
191     PROVIDE_HIDDEN (__fsymtab_start = .);
192     KEEP (*(FSymTab*))
193     PROVIDE_HIDDEN (__fsymtab_end = .);
194     PROVIDE_HIDDEN (__vsymtab_start = .);
195     KEEP (*(VSymTab*))
196     PROVIDE_HIDDEN (__vsymtab_end = .);
197     /* added end */
```

图 8.3　GCC 下链接文件示例

其中，针对 RT1064 搭配 GCC 编译的过程已经在项目 https://github.com/hilookas/SeekFree_RT1064_RTThread_Library_GCC_Porting 中体现出，并且整理汇总出了一篇技术博客 https://18kas.com/seekfree-rt1064-rt-thread-gcc-experience

第九章　RT-Thread 移植过程问题解决

下文会根据移植 RT-Thread 过程中遇到的问题与其相应的解决方式进行讨论。

9.1　将 FinSH 与无线串口结合使用

FinSH 是一个很强大的命令行，提供了以往很难实现命令行功能，为程序的运行提供了更多的可变性，可调试性。

默认情况下，RT-Thread 中使用的是由逐飞提供的有线串口驱动程序，但是往往许多时候，有线串口由于线的牵制，导致无法在小车运行的过程中使用，故使用无线串口会是一个比较好的选择。

无线串口相对于有线串口，会多出一个流控引脚 RTS，此外无线串口驱动程序会在 RTS 流控脚返回忙碌信号的时候使用死循环等待空闲。该过程可能会导致程序卡住，但是串口作为一种重要的调试工具，保证输出结果的完整正确会更加重要一些，权衡利弊，我们选择仍然使用无线串口驱动程序。

默认情况下 RT-Thread 使用的串口为 uart0 串口，但是由于板子布局设计上的考虑，实际使用的串口为 uart3，故需要对相应的串口代码进行更换。

经过实际测试，发现无线串口工作在 460800 这种速度下，仍然能保持稳定工作，为了尽量减少传输所需要的时间，可以将串口的速度从默认的 115200 提升至 460800。

在 UART3 接收中断中，增加相应代码，这里采用的是 RT-Thread 的邮箱机制，将数据传回到 FinSH 中。类似的也需要将 UART0 接收中断中的相关代码进行删除：

```
IFX_INTERRUPT(uart3_rx_isr, 0, UART3_RX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart3_handle);

    extern rt_mailbox_t uart_mb;
    uint8 dat;
    uart_getchar(UART_3, &dat);
    rt_mb_send(uart_mb, dat); // 发送邮件
    wireless_uart_callback();

    rt_interrupt_leave(); // 退出中断
}
```

在 rt_hw_console_output 函数中修改为使用无线串口驱动：

```
void rt_hw_console_output(const char *str)
{
    // uart_putstr(DEBUG_UART, str);
    uint32 seekfree_wireless_send_buff(uint8 *buff, uint32 len);
    seekfree_wireless_send_buff(str, strlen(str));
}
```

在 fputc 函数中也修改为使用无线串口驱动，使得 printf 也可以通过无线串口发送：

```
int fputc(int ch, FILE *stream)
{
    // uart_putchar(DEBUG_UART, (char)ch);
    uint32 seekfree_wireless_send_buff(uint8 *buff, uint32 len);
    seekfree_wireless_send_buff(&ch, 1);
    return(ch);
}
```

在 get_clk 板级初始化时，初始化无线串口：

```
#if(PRINTF_ENABLE)
    // uart_init(DEBUG_UART, DEBUG_UART_BAUD, DEBUG_UART_TX_PIN, DEBUG_UART_RX_PIN);
    void seekfree_wireless_init(void);
    seekfree_wireless_init();
#endif
```

9.2　关于 STM0 模块

默认情况下 STM0（SysTick）将被 RT-Thread 进程调度器使用，故不应在代码中使用。

9.3　解决 FinSH 无响应的问题

在 main 线程满载的情况下，FinSH 会出现无响应的问题，这个原因 FinSH 所在的线程默认优先级要比 main 优先级低，在 main 线程未处理完成前，FinSH 不会开始处理：

```
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
tshell    20  running 0x00000004 0x00001000    07%   0x00000009 000
tidle0    31  ready   0x00000004 0x00000400    00%   0x0000000d 000
main      10  suspend 0x00000004 0x00000800    03%   0x00000013 000
```

图 9.1　排查问题时的线程截图

该问题发生后，看起来很像是串口损坏，实际并非如此。因此推荐 main 线程中只进行其他线程的初始化操作，主 while 里一直保持休眠即可。

在 RTOS 下合理分配线程的优先级十分重要。

9.4 解决无法增量编译的错误

在修改后尝试进行增量编译时，ADS 会报“*** target pattern contains no '%'”错误，应对这个错误，一般的解决方式是重新选择 Binary file。但是在项目文件移动后，这个错误会重新出现，根除这个错误的方式是打开 project_name.launch，将 “<stringAttribute key="org.eclipse.cdt.launch.PROGRAM_NAME" value="${build_config}\project_name.elf"/>”改为“<stringAttribute key="org.eclipse.cdt.launch.PROGRAM_NAME" value="${project_loc}\${build_config}\project_name.elf"/>”。

```
43    <stringAttribute key="org.eclipse.cdt.launch.PROGRAM_NAME" value="${project_loc}\${build_config}\Homo_Motion.elf"/>
44    <stringAttribute key="org.eclipse.cdt.launch.PROJECT_ATTR" value="Homo_Motion"/>
```

第十章　RT-Thread 在小车中的应用

上文介绍了 RT-Thread 在复杂多任务环境下的重要性，并且论述了 RT-Thread 的实时特性，本章中，会详细论述这些特性对小车代码优化过程中的作用，经过 RT-Thread 优化后的小车上程序流程框图如下所示：

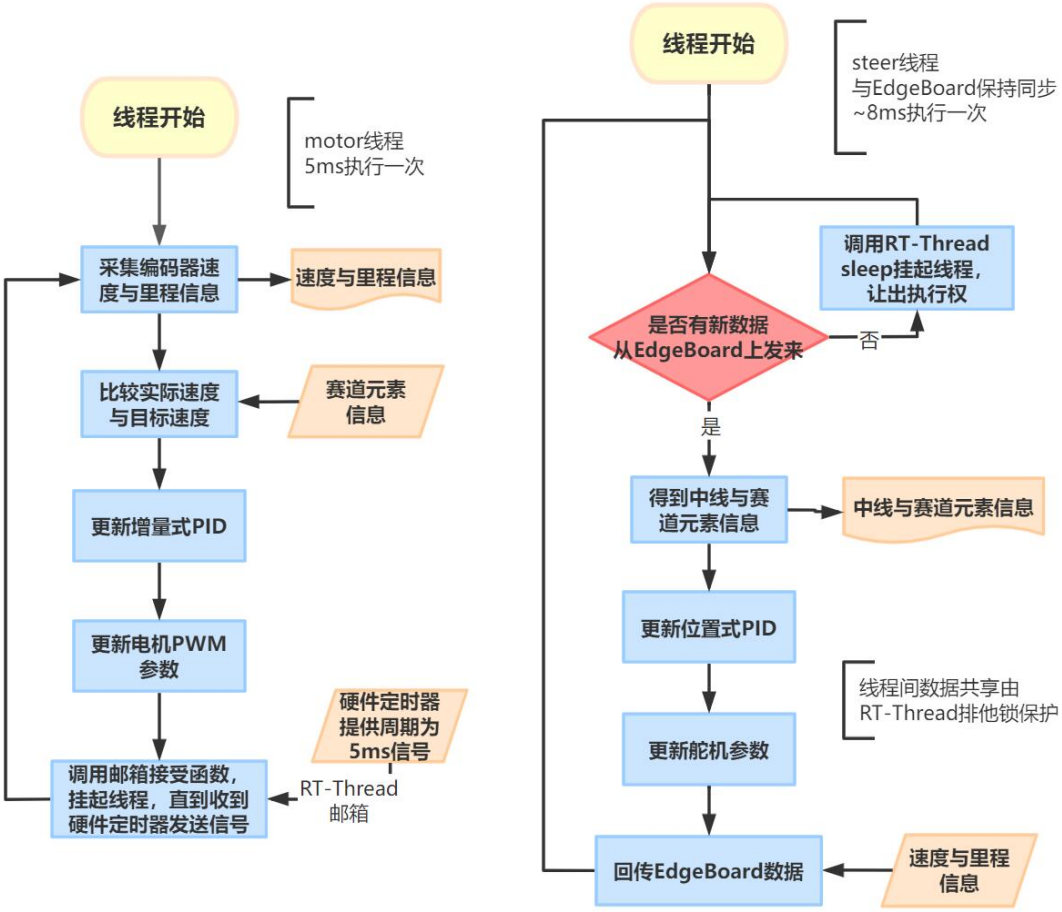


图 10.1　小车上程序流程框图

10.1　小车任务周期不同带来的代码编写挑战

小车在运行的过程中会遇到许多不同的任务，如方向控制、速度控制和一些调试上的使用，在没有 RT-Thread 多线程功能加持下，典型的代码结构如下图所示：

```c
int main() {
    while (1) {
        steer_task();
        motor_task();
    }
}
```

图 10.2　一种典型的代码结构

但是对于实际情况下，这些任务的周期往往不一定相同，于我们队伍而言，方向控制由于摄像头本身的限制，周期大致为 8ms，而速度控制为了保证响应及时、精准，周期为 5ms，以上的代码结构很难实现这种不同周期的任务。

且由于 C 语言不支持协程（coroutine）特性，在一个函数的执行过程中无法中断，故一个为了执行下一个任务，上一个任务必须完全执行完成。

举个例子，如果 steer_task 单次执行时间为 3ms，motor_task 单次执行时间为 1ms，steer_task 要求每 3ms 执行一次，motor_task 要求每 5ms 执行一次，执行过程将会如下所示：

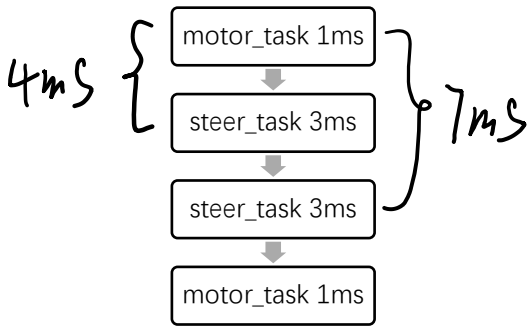


图 10.3　执行过程

会发现，motor_task 执行的时候一定会距离上次执行距离 7ms 的之间，遇到这种情况，我们可以通过在 steer_task 执行完成后固定延迟时间 1ms，以达到 motor_task 每 5ms 执行一次的要求，但是这样的话，相当于白白浪费了 1ms，浪费了处理器资源。

引入 RT-Thread 后，一个任务即使没有完全执行完，另一个优先级更高的任务达到了周期，需要开始运行，RT-Thread 线程调度器仍然可以把线程切换到需要开始执行的线程，上文的情况会变成如下：

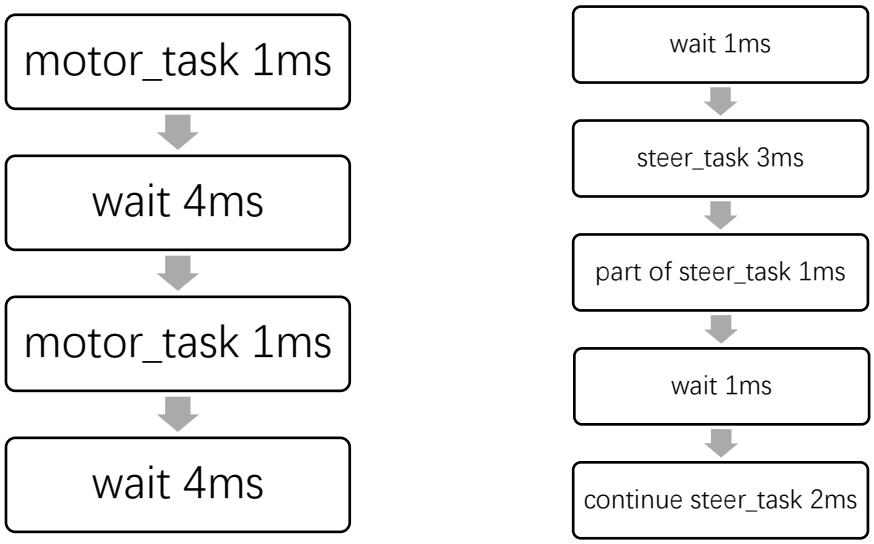


图 10.4　使用 RT-Thread 后两个线程可以充分利用处理器时间

通过使用 **RT-Thread 线程调度器**中断 steer 线程的执行，保证了 motor 线程周期的稳定性，并且在 motor 线程执行完成后，仍然可以恢复 steer 原始线程的运行。这**使任务的实时性得到了极大的提高**。

此外，传统结构中最小周期取决于运行最慢的程序，像屏幕显示这种执行一次非常消耗时间的任务，如单次处理需要 100ms，对于其他任务而言，只能等待 100ms 结束，即为了显示一次屏幕，周期为 5ms 的任务需要等待一次 100ms 函数执行，这是不可接受的。

传统方案上，屏幕显示一般用于调参过程，故会把屏幕显示过程放在主循环外面，如果想要修改参数，需要重新启动电源，对于小车可以频繁重启电源的场合，这尚能接受，但是试想如果是一个需要 24h 不断运行的大型机器，为了修改程序参数，重启电源，这是不可接受的。并且这种过程会比较损坏电源开关（开关打火），理想的状态下，调参可以在单片机始终运行的时候进行，RT-Thread 通过引入线程切换使得这种愿望得到了实现的可能。

运行时调参与 RT-Thread 在其中的运用会在下文中进行论述。

10.2 线程的拆分

实践中，随着功能不断变多，需要对代码进行合理的组织。**使用 RT-Thread 的线程功能，可以将不同功能的代码拆分开来，提升内聚度，降低代码耦合度**。

我们将代码拆分为如下几个模块，并分别对其建立线程：

- main 线程（RT-Thread 自带），初始化各个硬件模块，并且初始化线程
- finsh 线程（RT-Thread 自带），提供一个命令行功能
- steer 线程，负责方向控制，负责与执行图像算法的 EdgeBoard 通信，从图像中获取赛道当前曲率，并控制主舵机的角度。
- motor 线程，负责速度控制，主要运行 PID 算法，从编码器中获取速度数据，并且根据当前打角曲率，更新电机 PWM 参数
- menu 线程，将车辆运行的参数状态，实时显示在屏幕上，并且提供命令行与人机交互接口修改参数，以实现便捷调车。

与此同时，善用 RT-Thread 提供的锁，可以很好的同步线程数据，避免出现数据不同步导致的 bug。

图为**使用 RT-Thread 后的代码，整洁干净**，避免了原来不同代码拧在一起的混沌情况。

```
48  int main(void) {
49      // 等待所有核心初始化完毕
50      IfxCpu_emitEvent(&g_cpuSyncEvent);
51      IfxCpu_waitEvent(&g_cpuSyncEvent, 0xFFFF);
52
53      // user code start
54      // 无线串口已经在 rtt 初始化调用 get_clk 中已经初始化
55      // 屏蔽无线模块可参见 Libraries\seekfree_peripheral\SEEKFREE_WIRELESS.c:84 和
56      printf("Hello World!\n"); // Hello~
57      rt_kprintf("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789\
58
59      menu_init();
60      motor_init();
61      steer_init();
62
63      while (true) { // rtt 控制台测试用，如果main线程满占用，控制台无法响应数据
64          rt_thread_mdelay(5);
65      }
66      // user code end
67  }
```

图为小车运行过程中截图情况，展现出了多个任务和谐相处的情况：

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Jul 28 2022
 2006 - 2021 Copyright by rt-thread team
Hello World!
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789

ps
msh >ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
steer      7  suspend 0x00000004 0x00000400    07%   0x00000005 000
motor      5  suspend 0x00000004 0x00000400    06%   0x00000005 000
menu      25  ready   0x00000004 0x00000400    08%   0x00000005 000
tshell    20  running 0x00000004 0x00001000    07%   0x00000005 000
tidle0    31  ready   0x00000004 0x00000400    00%   0x0000001a 000
main      10  suspend 0x00000004 0x00000800    02%   0x00000013 000
```

图 10.5　RT-Thread 内置的 ps 命令打印结果

10.3　motor 线程的提出及与邮箱的使用

根据控制原理，基于阿克曼转向结构的小车运行主要有方向环和速度环两个部分构成，其中 motor 线程实现速度环的部分，主要负责有：

- 编码器信息采集，计算实时速度信息与里程信息；
- 根据采集来的实际速度信息与预设的目标速度信息进行比较；
- 通过比较出的差异通过 PID 控制算法更新电机的运行参数。

其中为了保证电机响应及时、准确，提升算法控制精度，motor 线程设定为每 5ms 采集数据并且更新电机参数一次。

由于电机的控制对小车速度影响很大，并且非常影响车辆运行的稳定性，设置其优先级为 5，是全部线程中最高的优先级：

```
114     rt_thread_t tid = rt_thread_create("motor", motor_main, RT_NULL,
115         1024, 5, 5); // 创建线程 // 优先级最高，确保编码器数据准时采集
```

由于对时间的精准性要求高，软件定时器的精度不能满足要求，我们为小车单独预留了一个定时器，间隔 5ms 执行一次，但是根据中断处理最小化的原则[4]，即中断中仅进行最基本的数据采集，数据处理在线程代码中进行，定时器中只会通知线程，已经开始新的线程。通知的过程采用了 RT-Thread 的邮箱机制。

硬件定时器中断中仅会通过邮箱提供一个信号：

```
40  IFX_INTERRUPT(cc60_pit_ch1_isr, 0, CCU6_0_CH1_ISR_PRIORITY
41  {
42      rt_interrupt_enter(); // 进入中断
43
44      enableInterrupts(); // 开启中断嵌套
45      PIT_CLEAR_FLAG(CCU6_0, PIT_CH1);
46
47      // user code start
48      rt_mb_send(motor_delay_mb, 0); // 发送邮件 延时时间到
49      // user code end
50
51      rt_interrupt_leave(); // 退出中断
52  }
```

Motor 线程会一直挂起，直到收到中断发送的邮件后，进入下一个运行周期：

USER > C motor.c > motor_main(void *)

```
 99          // rt_thread_mdelay(5);
100          // 等待邮件（高精度定时）
101          rt_ubase_t dat;
102          rt_mb_recv(motor_delay_mb, &dat, RT_WAITING_FOREVER);
103      }
104  }
```

最开始需要初始化邮箱以及定时器：

```
112  motor_delay_mb = rt_mb_create("motor_delay_mb", 1, RT_IPC_FLAG_FIFO);
```

```
34      // 编码器采集定时器
35      pit_interrupt_ms(CCU6_0, PIT_CH1, 5);
36      pit_enable_interrupt(CCU6_0, PIT_CH1);
37      pit_start(CCU6_0, PIT_CH1);
```

邮箱提供的线程间通信功能，尤其是未收到邮件挂起的功能，可以**通过仅仅 2 行代码实现高精度定时，并且运行延时可控。**

相对于传统方案，把所有的逻辑堆积在中断中，导致中断运行时间过长，并且可能带来中断嵌套的问题，这种使用 RT-Thread 将中断处理**分割为前台和后台两个部分，降低了逻辑复杂性，并且帮助开发者避开了中断嵌套等问题。**

本节主要介绍了与 RT-Thread 相关的代码编写过程，PID 本身的调试过程可参见控制算法一章。

10.4　steer 线程的提出及与锁的使用

类似于 motor 线程，steer 线程实现方向环部分，负责有：

- 与 EdgeBoard 通信，获取 EdgeBoard 上处理后的图像中线与赛道元素信息；
- 共享获取的赛道元素信息，通知 motor 线程更新其目标速度（弯道降速，直道提速）；
- 根据图像中线与赛道元素信息调整舵机的打脚。

类似于 motor 线程，steer 线程也有固定的运行周期，其运行周期取决于 EdgeBoard 上图像获取的周期，由于摄像头帧率为固定 120 帧，故图像处理周期固定为 8ms（帧率不提升，只提升周期是没有意义的，两次处理同样的图片不会计算出新的数据）。

由于任务特点，优先级设置上比 main 高，但是仍然会低于 motor 线程：

```
151     rt_thread_t tid = rt_thread_create("steer", steer_main, RT_NULL,
152         1024, 7, 5); // 创建线程 // 优先级在motor和main之间
```

在每个 steer 周期前期，会不断尝试（轮询 polling）是否有 EdgeBoard 的新通信数据传递回来，如果没有新数据，线程会休眠一段时间，伪代码如下，展示了通信的逻辑：

```
int main() {
    while (1) {
        while (没有从EB上获得数据) {
            rt_thread_mdelay(1);
        }

        rt_mutex_take(midline_mutex, RT_WAITING_FOREVER);
        midline = 通信结果0;
        yuansu = 通信结果1;
        qianzhan = 通信结果2;
        rt_mutex_release(midline_mutex);

        // 舵机逻辑
    }
}
```

采用轮询的方式方便了代码的书写，没有获得数据后延迟而非不断 while(1)提供了低优先级线程运行的机会。

值得注意的是，一次通信返回的三个数字，图像中值，当前元素与前瞻图像中值三者具有关联关系，跨线程读取的过程中，要确保一次读取的三个值为关联的值，故这里**对三个值的写入加了 RT-Thread 的排他锁（mutex）**。

排他锁编写的过程中，为了避免出现优先级低的线程因为占用着锁而比高优先级线程还享有运行权，出现优先级反转的问题，**持有锁的过程要保持时间最小**，在获取数据后保留一份到线程本地后，锁需要尽快释放，以便高优先级线程运行。[3]

第十一章　RT-Thread 在调试中的应用

RT-Thread 自带了一个占用资源很低但是功能十分强大的命令行控制台 FinSH，提供了类 Unix Shell 的体验，其上操作如同操作交换机等设备。通过编写命令处理函数，并且在 FinSH 中注册处理函数，可以实现很多小车运行时调试的功能，提供了传统方式不具有的便捷调试功能。本章中，会详细论述 FinSH 以及上位机（包括一款我们自制的上位机）进行交互的过程。下图展示了基于 RT-Thread 的调试方法类型以及其特点：

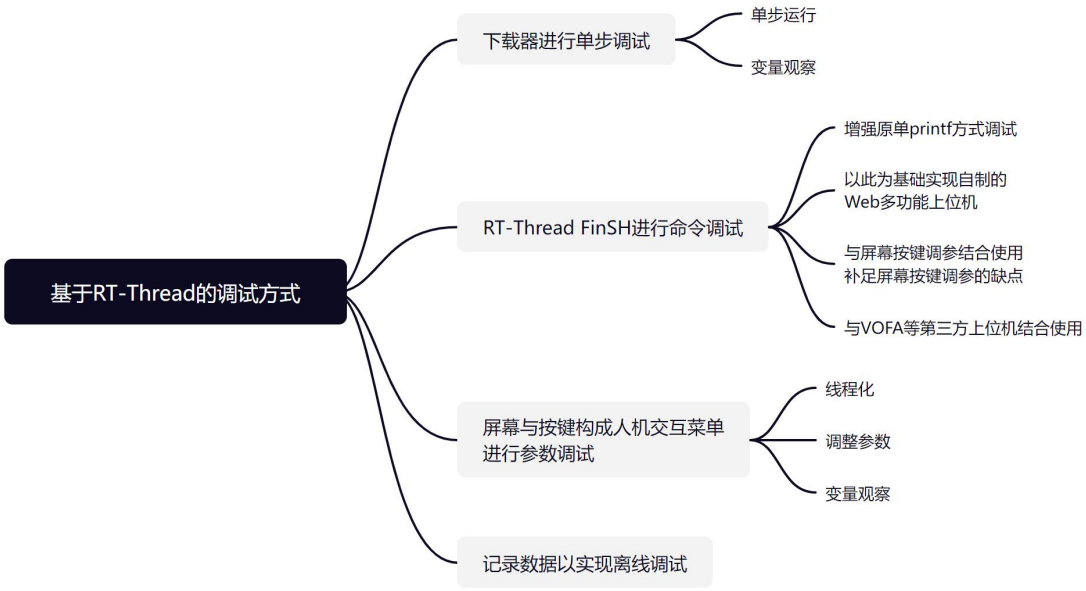


图 11.1　基于 RT-Thread 的调试方式

11.1　通过自制基于 Web 的上位机和 FinSH 进行小车的遥控与数据的采集

为了方便测试小车的控制，我们编写了上位机，让方向环也可以开环去跑。此时小车为一个遥控车，左右转与前进后端由我们远程控制。

自制上位机由两部分组成：前端和后端。

前端部分使用 Gamepad API 通过浏览器获得来自手柄的信号，将手柄的信号转换成对应的打脚值，然后组合成 FinSH 格式的命令。前端后端在最开始会通过 WebSocket 建立起一个通信渠道，FinSH 格式的命令会通过这个渠道传递给后端。

后端部分主要做中转的作用，将 WebSocket 中的命令，通过上位机系统提供的硬件 API 传递给无线串口，再由无线串口把信号传递给 FinSH 终端中。

整体流程如下图所示：

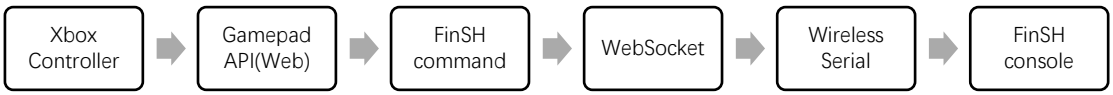


图 11.1　整体流程

小车程序需要注册两个命令，分别为“egs <speed>”和“srs <curv>”，分别为设置小车运行的速度与舵机转角的曲率。

在手柄偏角不断变化时，上位机会不断将手柄上的偏角结算为实际的曲率和速度，并且通过调用 FinSH 命令的方式通知小车变更。

前端中将手柄的信号转换为 FinSH 命令过程如下图所示：

```
150   window.changeSpeed = (newSpeed) => {
151     if (newSpeed > maxSpeed) newSpeed = maxSpeed;
152     if (newSpeed < -maxSpeed) newSpeed = -maxSpeed;
153     if (speed !== newSpeed) {
154       speed = newSpeed;
155       // console.dir(speed);
156       (async () => {
157         for (let i = 0; i < 3; ++i) { // 三次重试
158           try {
159             await runCommand('egs ' + speed, 1000);
160             return;
161           } catch (err) {}
162           await new Promise((resolve, reject) => {
163             setTimeout(resolve, 100);
164           });
165         }
166       })();
167     }
168   }
169 })();
```

上位机真实运行情况如下图所示：

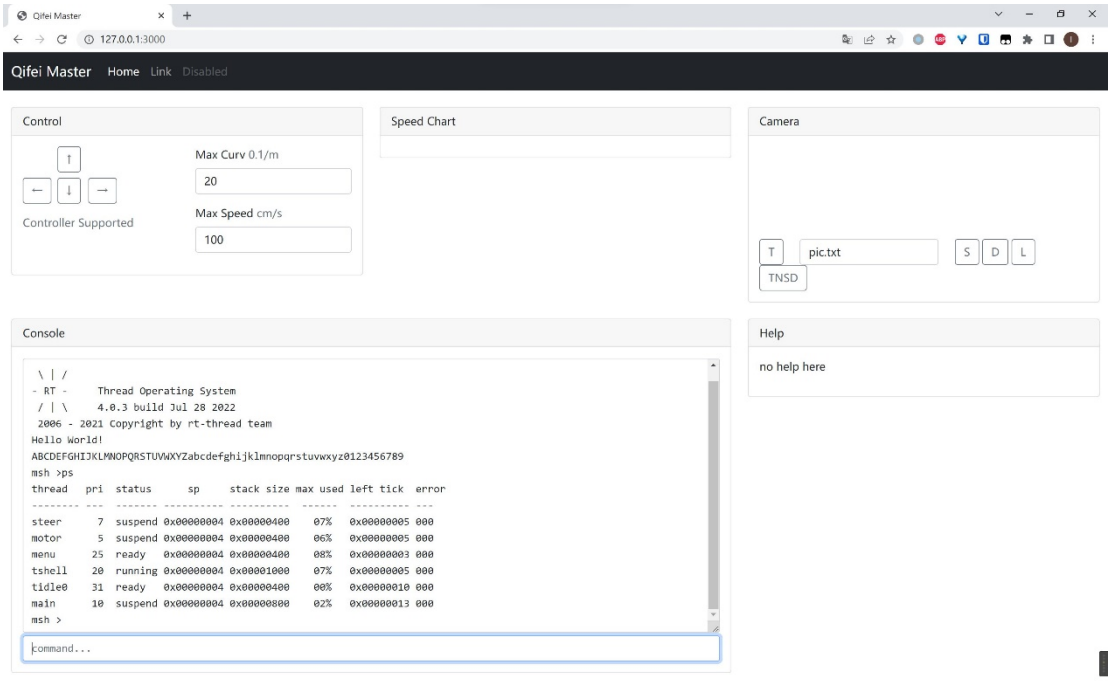


图 11.2　上位机的真实运行情况

在插入手柄后，上述功能即时启动，此上位机也提供有键盘和屏幕鼠标点击等备选方法控制小车。

上位机也提供一个简易的字符终端供小车调试过程中通过命令不断调整参数。

该项目已经在 GitHub 上开源：https://github.com/hilookas/qifei-master

11.2　解决显示屏显示影响小车性能问题

在实际调车中，会发现，如果关闭显示屏的输出，车辆的响应性能会有明显的提升。这是因为，单片机与显示屏通信需要占据大量的时间。但又由于显示屏有其重要的调试作用，并不可以简单粗暴的禁用它。故最好的解决方式是降低显示屏刷新输出的频率，即让其他任务处理与显示屏刷新的次数不一致。

传统大循环的方式，实现这种任务需要在主循环里写复杂的代码，如每两个周期调用一次显示屏刷新函数。但该方式仍然存在瞬卡的问题，即两次图像处理间隙过长，因为显示刷新的过程为一个整体，中间不可中断（不使用定时器）,该问题的已于前文通过举例的方式进行了论述。RT-Thread 提供多线程功能很好的解决了这个问题。

11.3　调参线程的优先级选择

屏幕显示，以及其对应的按键调参功能，在调车的过程中非常有用，但是单独的屏幕显示又十分耗时，故可以使用 RT-Thread 提供的线程功能，把调参线程的优先级降低，在确保其他任务已经正常执行完成后，再处理调参线程的内容，故在线程安排上，调参线程设置的优先级安排在 FinSH 线程线程的下方（不阻塞 FinSH 的运行），并且在 idle 线程的上面

```
214     rt_thread_t tid = rt_thread_create("menu", menu_main, RT_NULL,
215         1024, 25, 5); // 创建线程 // 优先级介于 main 与 shell 线程之间
```

11.4　使用 FinSH 控制台加速调试过程

调车中经常遇到需要频繁修改车辆运行参数以达到最好的车辆运行状态，面对这种情况，传统方式是在板上设置按钮同时设计图形或文字界面，调整参数时使用按钮进行。这种方式对调参过程有着很大的帮助，但是有些时候可能需要修改参数的幅度比较大，而调参微调按钮单次按下修改的参数幅度（步长）较小，调到指定大小的值，按着按钮会按很长的时间（几分钟），或者有些时候对现有参数不满意，想回退参数。这种时候，使用 FinSH 控制台，注册一个改参函数会极大的方便我们修改参数到指定的值。

图为改参函数：

```
221  // usage: ms param_name value
222  int menu_set_cli(int argc, char *argv[]) {
223      if (argc != 3) goto _err_parm;
224      int item_i = -1;
225      for (int i = 0; i < items_n; ++i) {
226          if (strcmp(argv[1], items[i].name) == 0) {
227              item_i = i;
228              break;
229          }
230      }
231      if (item_i == -1) {
232          rt_kprintf("Item not found\n");
233          return RT_ERROR;
234      }
235      float x;
236      sscanf(argv[2], "%f", &x);
237      // rt_kprintf("set: %f\n", x);
238      printf("set: %f\n", x);
239      items[item_i].value = x;
240      save();
241      assign();
242      return RT_EOK;
243  _err_parm:
244      rt_kprintf("Wrong parm!\n");
245      return RT_ERROR;
246  }
247
248  MSH_CMD_EXPORT_ALIAS(menu_set_cli, ms, menu set);
```

通过命令行传递需要修改的参数名称与对应的值，简单快速、直观高效：

```
ms steer_middle_duty 60
msh >ms steer_middle_duty 60
set: 60.000000
```

图 11.2　通过命令修改参数

因为调参程序为一个整体，通过命令行的修改也会同时反映在小车屏幕上：

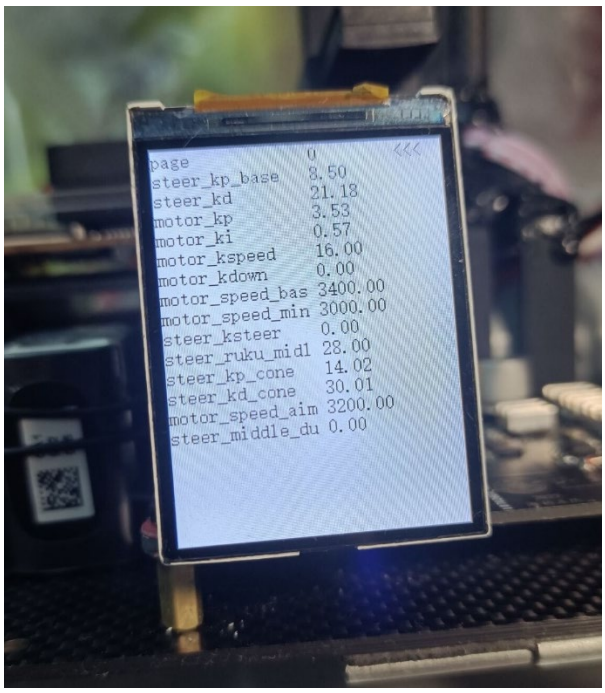


图 11.2　在命令修改后小车屏幕上数值也会同步变化（全局唯一）

也可以通过命令行列出当前参数，方便复制：

```
mg
msh >mg
steer_kp_base  8.500000
steer_kd  21.180000
motor_kp  3.530000
motor_ki  0.570000
motor_kspeed   16.000000
motor_kdown    0.000000
motor_speed_base    3400.000000
motor_speed_min     3000.000000
steer_ksteer   0.000000
steer_ruku_midline  28.000000
steer_kp_cone  14.020000
steer_kd_cone  30.010000
motor_speed_aim_cone     3200.000000
steer_middle_duty   0.000000
```

图 11.2　通过命令查询当前参数

11.5 使用 FinSH 控制台搭配 VOFA 调试电机参数

调参的过程中，尤其是电机的调试过程中，单纯靠视听的方式去判断参数的优劣并不是一个最好的方式，我们采用了 VOFA 作为电机调参时的上位机，其提供的曲线展示功能可以将单片机上回传的数字以图像的形式表示出来。

但是这个回传的过程并不需要小车一上电就运行，故可以将回传数字的过程写成一个命令函数，注册到 FinSH 中，需要的时候执行回传函数再开始发送数字。

```
// usage: mts
int motor_show_cli(int argc, char *argv[]) {
    while (true) {
        printf("speed:%d\n", motor_speed);
        rt_thread_mdelay(5);
    }
    return RT_EOK;
}

MSH_CMD_EXPORT_ALIAS(motor_show_cli, mts, motor show);
```

第十二章　RT-Thread 创新与开源项目及相应博客清单

在漫长的调车的过程中，我们积累了一些技术文章与经验，并且以此创新发展出很多实用的工具，部分工具已经在上文列出，其余有些由于篇幅原因，无法在文中详细描述，故在这里列出一个清单，希望能够给后来的参赛队伍带来帮助。

1. qifei-master 基于 FinSH 与 Web 技术的多功能上位机
   https://github.com/hilookas/qifei-master
2. 逐飞 RT1064 库 GCC (VSCode) 移植
   https://github.com/hilookas/SeekFree_RT1064_Library_GCC_Porting
3. 逐飞 RT1064 RT-Thread 库 GCC (VSCode) 移植
   https://github.com/hilookas/SeekFree_RT1064_RTThread_Library_GCC_Porting
4. 逐飞 RT1064 库 GCC (VSCode) 移植踩坑
   https://18kas.com/seekfree-rt1064-gcc-experience
5. 逐飞 RT1064 RT-Thread 库 GCC (VSCode) 移植踩坑
   https://18kas.com/seekfree-rt1064-rt-thread-gcc-experience
6. 百度 Edgeboard （Alinx） FZ3B 上 Ubuntu 18.04 安装 RT5370 无线网卡驱动
   https://18kas.com/fz3b-rt5370
7. Homo_Motion_Template 一个针对 VSCode 优化后的基础项目胚子
   https://github.com/hilookas/Homo_Motion_Template

第十三章　结论

我们小车以完全模型组 I 车模作为基础，利用百度 EdgeBoard 作为图像处理硬件，英飞凌 TC264 芯片作为小车控制核心，舵机、电机、编码器等作为反馈和执行器件，辅以长时间的软硬件联合调试（PID 等调参），实现了小车的高速运行，并且获得了综合奖励时间省赛负秒的好成绩。

车模设计实现过程中，遇到的环境复杂，有线上赛 Python 环境、EdgeBoard Linux 环境也有 TC264 上 RT-Thread 环境。我们在不断实现这比赛要求的赛题任务同时，探索着多环境开发的最佳实践，**自主开发并且在 GitHub 上开源分享了许多调试工具**，并在本文中论述了调试过程。

RT-Thread 是一个很优秀的实时操作系统，在工程实践中，可以很大程度上降低代码编写复杂度，提升代码可维护性，在开发、调试、运行过程中，都可发挥其重要的作用。尤其是在智能车比赛中，**RT-Thread 可以合理分配任务运行，降低开发负担，提升开发速度。**

通过 RT-Thread 的多线程、互斥锁、邮箱等特性，帮助代码减少了无谓的等待，**提升了代码的运行性能**。通过 RT-Thread 的 FinSH 控制台，以及其自带的一些命令，**提升了代码的调试便捷性。**

代码编写过程中，也体会到了 RT-Thread API 编写的精良，其 API 的实现尽可能与 POSIX 等行业标准靠近，极大的降低了开发者的迁移学习成本，并且通过 API 的设计尽可能抹平了不同平台的差异，使在不同单片机平台的开发体验基本保持一致。FinSH 等也提供了一个在 Linux 等命令行环境下工作的体验，帮助开发者快速熟悉环境，并且上手开发。

使用 RT-Thread 时遇到的问题已在文中论述，本着见到一个问题处理一个问题的原则，并没有遗留下具体问题等待解决。

完全模型组别的比赛任务比较复杂，不同环境下编程的复杂性在挑战着我们对代码的掌握能力。比赛过程中最容易出现的问题是错误排查起来费劲，通过引入 RT-Thread，避免在中断中执行逻辑等很大程度上解决了这个问题。此外，我们也通过将测试驱动开发（TDD）的理念引入比赛，大量使用 assert，预防问题而非出现问题后再进行处理。

智能汽车比赛是一个复杂的比赛，一个成功的小车离不开软硬件队友的相互配合，也离不开图像控制队员的沟通与理解。通过团队配合，1+1 可以产生大于 2 的效果。同时比赛也鼓励队伍间的合理交流，在此也感谢总群中的张扬学姐不断为车友们回答问题，集美大学康泽豪等同学不断交流调车经验，赛曙与百度的多场联合培训降低了完全模型组别的入门门槛。

参考文献

[1] kunkliu. AURIX TC3XX 启动文件解析. CSDN. [联机] https://blog.csdn.net/kunkliu/article/details/125221208.

[2] Linux. pthread man page.

[3] RT-Thread Team. 线程间同步. RT-Thread 文档中心. [联机] https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/programming-manual/ipc1/ipc1?id=%e4%ba%92%e6%96%a5%e9%87%8f%e5%b7%a5%e4%bd%9c%e6%9c%ba%e5%88%b6.

[4] 一. 中断管理. RT-Thread 文档中心. [联机] https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/programming-manual/interrupt/interrupt?id=%e4%b8%ad%e6%96%ad%e7%9a%84%e5%ba%95%e5%8d%8a%e5%a4%84%e7%90%86.

[5] Python Maintain Team. Global Interpreter Lock. Python Wiki. [联机] https://wiki.python.org/moin/GlobalInterpreterLock.

[6] 崔海勤. 2021. hilookas/SeekFree_RT1064_Library_GCC_Porting. GitHub. [联机] https://github.com/hilookas/SeekFree_RT1064_Library_GCC_Porting.

[7] 一. 2021. 逐飞 RT1064 RT-Thread 库 GCC (VSCode) 移植踩坑. lookas. [联机] https://18kas.com/seekfree-rt1064-rt-thread-gcc-experience/.

[8] 张超,彭金璋,刘纯鸽,何宇桦.智能车跑道图像的大津阈值分割算法[J].吉首大学学报(自然科学版),2014,35(04):27-30.

[9] 胡晋山,康建荣,张琪,刘鹏程,朱铭达.一种八邻域图像边界追踪改进算法[J].测绘通报,2018(12):21-25.DOI:10.13474/j.cnki.11-2246.2018.0451.

[10] 《Visual Studio Code 权威指南》. 电子工业出版社.

[11] 《嵌入式实时操作系统：RT-Thread 设计与实现》. 机械工业出版社.

[12] 《RT-Thread 内核实现与应用开发实战指南 基于 STM32》. 机械工业出版社.

[13] 《鸟哥的 Linux 私房菜》. 人民邮电出版社.

[14] 《数字电子技术基础》. 高等教育出版社.

[15] 侯虹.采用模糊 PID 控制律的舵机系统设计[J].航空兵器，2006,2(1):7-9.

[16] 张文春．汽车理论[M]．北京．机械工业出版社．2005

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

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

[19] 夏克俭．数据结构及算法 [M] ．北京：国防工业出版社， 2001.

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

[21] Garcia-Garcia A, Orts-Escolano S, Oprea S, et al. A review on deep learning techniques applied to semantic segmentation[J]. arXiv preprint arXiv:1704.06857, 2017.

[22] PaddlePaddle. 2022. 第十七届全国大学生智能汽车竞赛：百度完全模型组竞速赛线上资格赛. 飞桨 AI Studio. [联机] https://aistudio.baidu.com/aistudio/competition/detail/131/0/task-definition.

[23] Mikołajczyk A, Grochowski M. Data augmentation for improving deep learning in image classification problem[C]//2018 international interdisciplinary PhD workshop (IIPhDW). IEEE, 2018: 117-122.

[24] Li Y, Huang H, Xie Q, et al. Research on a surface defect detection algorithm based on MobileNet-SSD[J]. Applied Sciences, 2018, 8(9): 1678.

附录 A：主项目代码

main.c

```
#include "headfile.h"
#include "rtthread.h"
#pragma section all "cpu0_dsram"

// 函数声明

// user code start
#include "menu.h"
#include "motor.h"
#include "steer.h"
// user code end

int main(void) {
    // 等待所有核心初始化完毕
    IfxCpu_emitEvent(&g_cpuSyncEvent);
    IfxCpu_waitEvent(&g_cpuSyncEvent, 0xFFFF);

    // user code start
    // 无线串口已经在 rtt 初始化调用 get_clk 中已经初始化
    // 屏蔽无线模块可参见
Libraries\seekfree_peripheral\SEEKFREE_WIRELESS.c:84 和 152 让两个函数直
接 return 掉即可
    printf("Hello World!\n"); // Hello~
    rt_kprintf("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012
3456789\n"); // 测试控制台输出,并且利用无线模块未插入程序卡死的原理停住程序

    menu_init();
    motor_init();
    steer_init();

    while (true) { // rtt 控制台测试用，如果 main 线程满占用，控制台无法响应
数据
        rt_thread_mdelay(5);
    }
    // user code end
}
```

```
#pragma section all restore
```

isr.c

```
#include "rtthread.h"
#include "isr_config.h"
#include "isr.h"

// user code start
#include "motor.h"
// user code end

// PIT 中断函数  示例
IFX_INTERRUPT(cc60_pit_ch0_isr, 0, CCU6_0_CH0_ISR_PRIORITY)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_0, PIT_CH0);

    rt_interrupt_leave(); // 退出中断
}


IFX_INTERRUPT(cc60_pit_ch1_isr, 0, CCU6_0_CH1_ISR_PRIORITY)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_0, PIT_CH1);

    // user code start
    rt_mb_send(motor_delay_mb, 0); // 发送邮件 延时时间到
    // user code end

    rt_interrupt_leave(); // 退出中断
}

IFX_INTERRUPT(cc61_pit_ch0_isr, 0, CCU6_1_CH0_ISR_PRIORITY)
{
```

```
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_1, PIT_CH0);

    rt_interrupt_leave(); // 退出中断
}

IFX_INTERRUPT(cc61_pit_ch1_isr, 0, CCU6_1_CH1_ISR_PRIORITY)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    PIT_CLEAR_FLAG(CCU6_1, PIT_CH1);

    rt_interrupt_leave(); // 退出中断
}



IFX_INTERRUPT(eru_ch0_ch4_isr, 0, ERU_CH0_CH4_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    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);
    }

    rt_interrupt_leave(); // 退出中断
}

IFX_INTERRUPT(eru_ch1_ch5_isr, 0, ERU_CH1_CH5_INT_PRIO)
{
```

```
    rt_interrupt_enter(); // 进入中断

    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);
    }

    rt_interrupt_leave(); // 退出中断
}

// 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, 0, 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, 0, ERU_CH3_CH7_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    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);

    }

    rt_interrupt_leave(); // 退出中断
}



IFX_INTERRUPT(dma_ch5_isr, 0, ERU_DMA_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    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();

    rt_interrupt_leave(); // 退出中断
}


// 串口中断函数  示例
IFX_INTERRUPT(uart0_tx_isr, 0, UART0_TX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart0_handle);

    rt_interrupt_leave(); // 退出中断
}
```

```
IFX_INTERRUPT(uart0_rx_isr, 0, UART0_RX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart0_handle);

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart0_er_isr, 0, UART0_ER_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrError(&uart0_handle);

    rt_interrupt_leave(); // 退出中断
}

// 串口 1 默认连接到摄像头配置串口
IFX_INTERRUPT(uart1_tx_isr, 0, UART1_TX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart1_handle);

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart1_rx_isr, 0, UART1_RX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    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();

    rt_interrupt_leave(); // 退出中断
```

```
}
IFX_INTERRUPT(uart1_er_isr, 0, UART1_ER_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrError(&uart1_handle);

    rt_interrupt_leave(); // 退出中断
}


// 串口 2 默认连接到无线转串口模块
IFX_INTERRUPT(uart2_tx_isr, 0, UART2_TX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart2_handle);

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart2_rx_isr, 0, UART2_RX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart2_handle);

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart2_er_isr, 0, UART2_ER_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrError(&uart2_handle);

    rt_interrupt_leave(); // 退出中断
}
```

```
IFX_INTERRUPT(uart3_tx_isr, 0, UART3_TX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrTransmit(&uart3_handle);

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart3_rx_isr, 0, UART3_RX_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrReceive(&uart3_handle);

    extern rt_mailbox_t uart_mb;
    uint8 dat;
    uart_getchar(UART_3, &dat);
    rt_mb_send(uart_mb, dat); // 发送邮件
    wireless_uart_callback();

    rt_interrupt_leave(); // 退出中断
}
IFX_INTERRUPT(uart3_er_isr, 0, UART3_ER_INT_PRIO)
{
    rt_interrupt_enter(); // 进入中断

    enableInterrupts(); // 开启中断嵌套
    IfxAsclin_Asc_isrError(&uart3_handle);

    rt_interrupt_leave(); // 退出中断
}
```

menu.c

```
#include "menu.h"
#include "headfile.h"
```

```c
#include "zf_assert.h"
#define my_assert ZF_ASSERT

#include "motor.h"
#include "steer.h"

// 按键
#define KEY_DOWN_PIN P22_0
#define KEY_SUB_PIN P22_1
#define KEY_UP_PIN P22_2
#define KEY_ADD_PIN P22_3
#define KEY_MID_PIN P23_1
// 拨码开关
#define SW1_PIN P33_12
#define SW2_PIN P33_13

static struct item_s {
    char *name;
    float value;
    float step_add, step_sub;
};

typedef struct item_s item_t;

static item_t items[] = {
    { "steer_kp_base", 0, 0.1, 0.05 },
    { "steer_kd", 0, 0.1, 0.05 },
    { "motor_kp", 0.9, 0.05, 0.01 },
    { "motor_ki", 0.25, 0.05, 0.01 },
    { "motor_kspeed", 0, 0.25, 0.25 },
    { "motor_kdown", 1, 0.1, 0.05 },
    { "motor_speed_base", 1700, 1500, 100 },
    { "motor_speed_min", 1000, 100, 50 },
    { "steer_ksteer", 0, 0.01, 0.05 },
    { "steer_ruku_midline", 0, 1, 1 },
    { "steer_kp_cone", 0, 0.1, 0.05 },
    { "steer_kd_cone", 0, 0.1, 0.05 },
    { "motor_speed_aim_cone", 0, 100, 50 },
    { "steer_middle_duty", 0, 100, 10 },
};
```

```
static int items_n = sizeof items / sizeof (item_t);

static void save(void) {
    int k = 0;
    for (int i = 1; k < items_n && i <= 2; ++i) { // 0 不使用
        eeprom_erase_sector(i); // 清除扇区
        for (int j = 0; k < items_n && j < 16; ++j, ++k) {
            eeprom_page_program(i, 4 * j, (uint32 *)&(items[k].value));
// 写入参数
        }
    }
}

static void load(void) {
    int k = 0;
    for (int i = 1; k < items_n && i <= 2; ++i) {
        for (int j = 0; k < items_n && j < 16; ++j, ++k) {
            items[k].value = flash_read(i, 4 * j, float);
        }
    }
}

static void assign(void) {
    steer_kp_base = items[0].value;
    steer_kd = items[1].value;
    motor_kp = items[2].value;
    motor_ki = items[3].value;
    motor_kspeed = items[4].value;
    motor_kdown = items[5].value;
    motor_speed_base = items[6].value;
    motor_speed_min = items[7].value;
    steer_ksteer = items[8].value;
    steer_ruku_midline = items[9].value;
    steer_kp_cone = items[10].value;
    steer_kd_cone = items[11].value;
    motor_speed_aim_cone = items[12].value;
    steer_middle_duty = items[13].value;
}

#define MAX_PAGE 2 // [0,MAX_PAGE]
```

```
#define MAX_LINE 19 // [0,MAX_LINE]

static int page, line;

static void show(void) {
    ips200_showstr(0, 0, "page"); ips200_showint32(120, 0, page, 4);
    ips200_showstr(200, (uint16)line, "<<<");

    if (page == 0) {
        for (int i = 1; i <= MAX_LINE; ++i) {
            int k = i - 1;
            if (k < items_n) {
                ips200_showstr(0, (uint16)i, items[k].name);
                ips200_showfloat(120, (uint16)i, items[k].value, 4, 2);
            }
        }
    } else if (page == 1) {
        for (int i = 1; i <= MAX_LINE; ++i) {
            int k = i - 1 + MAX_LINE;
            if (k < items_n) {
                ips200_showstr(0, (uint16)i, items[k].name);
                ips200_showfloat(120, (uint16)i, items[k].value, 4, 2);
            }
        }
    } else if (page == 2) { // 专门用于显示运行时参数
        ips200_showstr(0, 1, "fache_flag"); ips200_showint32(120, 1,
fache_flag, 4);
        // ips200_showstr(0, 2,
"motor_speed:");     ips200_showint32(120, 2, motor_speed, 4);
        ips200_showstr(0, 3, "midline:");   ips200_showint32(120, 3,
midline, 4);
    }
}

static void action(void) {
    if (gpio_get(KEY_UP_PIN) == 0) {
        --line;
        if (line < 0) line = MAX_LINE;
        ips200_clear(IPS200_BGCOLOR);
    } else if (gpio_get(KEY_DOWN_PIN) == 0) {
```

```
        ++line;
        if (line > MAX_LINE) line = 0;
        ips200_clear(IPS200_BGCOLOR);
    } else if (gpio_get(KEY_ADD_PIN) == 0) {
        if (line == 0) {
            ++page;
            if (page > MAX_PAGE) page = 0;
            ips200_clear(IPS200_BGCOLOR);
            save(); // 更换页面时保存参数
        } else {
            if (page == 0) {
                int k = line - 1;
                if (k < items_n) {
                    items[k].value += items[k].step_add;
                }
            } else if (page == 1) {
                int k = line - 1 + MAX_LINE;
                if (k < items_n) {
                    items[k].value += items[k].step_add;
                }
            } else if (page == 2) {
                // switch (line) {
                // case 1: ++a; break;
                // }
            }
        }
    } else if (gpio_get(KEY_SUB_PIN) == 0) {
        if (line == 0) {
            --page;
            if (page < 0) page = MAX_PAGE;
            ips200_clear(IPS200_BGCOLOR);
            save(); // 切换页面时保存参数
        } else {
            if (page == 0) {
                int k = line - 1;
                if (k < items_n) {
                    items[k].value -= items[k].step_sub;
                }
            } else if (page == 1) {
                int k = line - 1 + MAX_LINE;
```

```
                if (k < items_n) {
                    items[k].value -= items[k].step_sub;
                }
            } else if (page == 2) {
                // switch (line) {
                // case 1: --a; break;
                // }
            }
        }
    } else if (gpio_get(KEY_MID_PIN) == 0) {
        line = 0;
        ips200_clear(IPS200_BGCOLOR);
        save(); // 中键按下保存参数
    }

    assign();

    if (page == MAX_PAGE) {
        fache_flag = true; // 切换到最后一页即发车
    }
}

static void menu_main(void *param) {
    // screen
    ips200_init();

    // key_init
    gpio_init(KEY_UP_PIN, GPI, 1, PULLUP);
    gpio_init(KEY_ADD_PIN, GPI, 1, PULLUP);
    gpio_init(KEY_DOWN_PIN, GPI, 1, PULLUP);
    gpio_init(KEY_SUB_PIN, GPI, 1, PULLUP);
    gpio_init(KEY_MID_PIN, GPI, 1, PULLUP);

    // sw_init
    gpio_init(SW1_PIN, GPI, 1, PULLUP);
    gpio_init(SW2_PIN, GPI, 1, PULLUP);

    page = 0, line = 0;

    while (true) {
```

```
        action();
        show();
        rt_thread_mdelay(50); // 20 帧（肯定达不到）
    }
}

static bool menu_inited;

bool menu_init(void) {
    my_assert(!menu_inited);
    menu_inited = true;

    load();
    assign();

    rt_thread_t tid = rt_thread_create("menu", menu_main, RT_NULL,
        1024, 25, 5); // 创建线程 // 优先级介于 idle 与 shell 线程之间
    my_assert(tid != RT_NULL);
    rt_thread_startup(tid); // 启动线程
    return false;
}

// usage: ms param_name value
int menu_set_cli(int argc, char *argv[]) {
    if (argc != 3) goto _err_parm;
    int item_i = -1;
    for (int i = 0; i < items_n; ++i) {
        if (strcmp(argv[1], items[i].name) == 0) {
            item_i = i;
            break;
        }
    }
    if (item_i == -1) {
        rt_kprintf("Item not found\n");
        return RT_ERROR;
    }
    float x;
    sscanf(argv[2], "%f", &x);
    // rt_kprintf("set: %f\n", x);
    printf("set: %f\n", x);
```

```
    items[item_i].value = x;
    save();
    assign();
    return RT_EOK;
_err_parm:
    rt_kprintf("Wrong parm!\n");
    return RT_ERROR;
}

MSH_CMD_EXPORT_ALIAS(menu_set_cli, ms, menu set);

// usage: mg
int menu_get_cli(int argc, char *argv[]) {
    for (int i = 0; i < items_n; ++i) {
        printf("%s\t%f\n", items[i].name, items[i].value);
    }
    return RT_EOK;
}

MSH_CMD_EXPORT_ALIAS(menu_get_cli, mg, menu get);
```

motor.c

```
#include "motor.h"
#include "headfile.h"
#include "zf_assert.h"
#define my_assert ZF_ASSERT
#include "steer.h"

#define MOTOR_PIN ATOM0_CH0_P21_2
#define MOTOR_PH_PIN P21_3

int motor_speed, motor_mileage;

// 速度 PI 控制（先调 I 再调 P）
float motor_kp, motor_ki;
// 速度策略
float motor_kspeed;
float motor_kdown;
float motor_speed_base, motor_speed_min;
```

```
// 目标速度控制
float motor_speed_aim;
float motor_speed_aim_cone = 0;

rt_mailbox_t motor_delay_mb;

static void motor_main(void *param) {
    // 电机
    gtm_pwm_init(MOTOR_PIN, 30000, 0); // 电机 EN 30000Hz
    gpio_init(MOTOR_PH_PIN, GPO, 1, PUSHPULL); // 电机方向引脚初始化

    // 编码器初始化
    gpt12_init(GPT12_T2, GPT12_T2INB_P33_7, GPT12_T2EUDB_P33_6);

    while (!fache_flag) rt_thread_mdelay(100);

    // 编码器采集定时器
    pit_interrupt_ms(CCU6_0, PIT_CH1, 5);
    pit_enable_interrupt(CCU6_0, PIT_CH1);
    pit_start(CCU6_0, PIT_CH1);

    while (true) {
        // 编码器采集
        int encoder_diff = -gpt12_get(GPT12_T2); // 单位 mm
        gpt12_clear(GPT12_T2);
        static uint32_t encoder_absolute; // 类似全局变量
        encoder_absolute += encoder_diff; // 自上电后编码器值

        motor_speed = (double)encoder_diff * 1000 / 6835/* 6835 1m 的编码
器值 */ * 1000 * 1000 / 5000; // 单位 1mm/s
        motor_mileage = (double)encoder_absolute / 6835 * 1000; // 单位
mm

        rt_mutex_take(midline_mutex, RT_WAITING_FOREVER);
        // int my_midline = midline;
        int my_yuansu = yuansu;
        // int my_qianzhan = qianzhan;
        rt_mutex_release(midline_mutex);

        // printf("speed:%d\n", motor_speed); // VOFA 回传
```

```
        // 速度策略
        // 弯道减速
        // motor_speed_aim = motor_speed_base - (steer_error *
steer_error * (motor_speed_base - motor_speed_min)) * motor_kspeed /
700.0 / 70;
        // if (motor_speed_aim <= motor_speed_min) motor_speed_aim =
motor_speed_min; // 确保最低速度

        // 前瞻减速
        // motor_speed_aim -= abs(my_qianzhan - 70) * 200 / 70 * speed /
3300 * motor_kdown;

        // 出界停车
        ++baohu_count;
        if (baohu_count >= 6) tingche_flag = true;

        if (my_yuansu == 3) {
            motor_speed_aim = motor_speed_aim_cone;
        } else if (my_yuansu == 4) {
            motor_speed_aim = 3800;
        } else if (my_yuansu == 2) {
            motor_speed_aim = 1000;
        }

        if (tingche_flag) motor_speed_aim = 0; // 停车

        // 更新 PID
        // 增量式 pi
        static float error = 0, result = 0;
        float error_pre = error;
        error = motor_speed_aim - motor_speed;
        result += motor_kp * (error - error_pre) + motor_ki * error;

        // 限幅
        if (result >= 8000) result = 8000;
        if (result <= -8000) result = -8000;

        // 速度输出
        if (result >= 0) {
```

```
            gpio_set(MOTOR_PH_PIN, 1);
            pwm_duty(MOTOR_PIN, result);
        } else {
            gpio_set(MOTOR_PH_PIN, 0);
            pwm_duty(MOTOR_PIN, -result);
        }

        // rt_thread_mdelay(5);
        // 等待邮件（高精度定时）
        rt_ubase_t dat;
        rt_mb_recv(motor_delay_mb, &dat, RT_WAITING_FOREVER);
    }
}

static bool motor_inited;

bool motor_init(void) {
    my_assert(!motor_inited);
    motor_inited = true;

    motor_delay_mb = rt_mb_create("motor_delay_mb", 1,
RT_IPC_FLAG_FIFO);

    rt_thread_t tid = rt_thread_create("motor", motor_main, RT_NULL,
        1024, 5, 5); // 创建线程 // 优先级最高，确保编码器数据准时采集
    my_assert(tid != RT_NULL);
    rt_thread_startup(tid); // 启动线程
    return false;
}

// usage: mts
int motor_show_cli(int argc, char *argv[]) {
    while (true) {
        printf("speed:%d\n", motor_speed);
        rt_thread_mdelay(5);
    }
    return RT_EOK;
}

MSH_CMD_EXPORT_ALIAS(motor_show_cli, mts, motor show);
```

comm.c

```c
#include "comm.h"
#include "serial.h"
#include <string.h>
#include "zf_assert.h"
#define my_assert ZF_ASSERT
#include "headfile.h"

// 不同类型的数据包大小
int comm_payload_size[] = {
    1, // COMM_TYPE_PING
    1, // COMM_TYPE_PONG
    3, // COMM_TYPE_UPDATE_TO_TC264
    4, // COMM_TYPE_UPDATE_FROM_TC264
    0, // COMM_TYPE_HELLO_FROM_TC264
};

// 堵塞发送一个数据包
// type 传递数据类型，payload 传递实际数据
// 返回是否发生错误
bool comm_send_blocking(comm_type_t type, const uint8_t payload[]) {
    bool ret;
    ret = serial_send_blocking(0x5A); // 0b01011010 as header
    if (ret) return true;
    ret = serial_send_blocking((uint8_t)type); // 0b01011010 as header
    if (ret) return true;
    for (int i = 0; i < comm_payload_size[type]; ++i) {
        ret = serial_send_blocking(payload[i]);
        if (ret) return true;
    }
    return false;
}

static uint8_t recv_buf[2 + COMM_PAYLOAD_SIZE_MAX];
static int recv_buf_p;

// 尝试接收一个数据包
```

```
// 如果没有发生错误，则 *type 内为接收到的数据包类型，payload 为接收到的实际数据
// 返回是否发生错误
// 这个函数发生错误十分正常，在没有接收到数据的时候，就会返回错误（不阻塞）
bool comm_recv_poll(comm_type_t *type, uint8_t payload[]) {
    bool ret;
    while (true) {
        uint8_t buf;
        ret = serial_recv_poll(&buf);
        if (ret) return true; // 没有新数据
        if (recv_buf_p == 0 && buf != 0x5A) continue; // 无效数据
        recv_buf[recv_buf_p++] = buf;
        my_assert(recv_buf_p <= sizeof recv_buf); // 缓冲区太小存不下完整数据包
        my_assert(recv_buf_p != 2 || recv_buf[1] < (sizeof comm_payload_size) / (sizeof comm_payload_size[0])); // 收到未知类型的数据包
        if (recv_buf_p >= 2 && recv_buf_p == 2 + comm_payload_size[recv_buf[1]]) { // 数据包到结尾了
            // 复制数据输出
            *type = recv_buf[1];
            memcpy(payload, recv_buf + 2, sizeof comm_payload_size[recv_buf[1]]);
            // 清空缓冲区
            recv_buf_p = 0;
            break;
        }
        // TODO 增加超时机制
    }
    return false;
}

// 返回是否发生错误
bool comm_init(void) {
    bool ret = serial_init();
    if (ret) return true; // 没有新数据
    recv_buf_p = 0;
    return false;
}
```

```
// 通讯的测试程序
// 将串口助手打开，发送后车子会返回数据以测试通讯是否正常
void comm_test() {
    comm_type_t type;
    uint8_t payload[10];
    bool ret = comm_recv_poll(&type, payload);
    if (!ret) {
        // success
        if (type == COMM_TYPE_PING) {
            // pong back
            printf("pong %02x\n", payload[0]);
            ret = comm_send_blocking(COMM_TYPE_PONG, payload);
            my_assert(!ret);
        }
    }
}
```

serial.c

```
#include "serial.h"
#include "headfile.h"

// 初始化串口
// 返回是否发生错误
bool serial_init(void) {
    uart_init(UART_2, 460800, UART2_TX_P02_0, UART2_RX_P02_1); // 初始化
串口
    return false;
}

// 堵塞输出一个字符
// c 为要发送的字符
// 返回是否发生错误
bool serial_send_blocking(uint8_t c) {
    uart_putchar(UART_2, c); // 实际上为非堵塞输出
    return false;
}

// 非堵塞轮询一个字符
// c 为要接受的字符
```

```
// 返回是否发生错误
bool serial_recv_poll(uint8_t *c) {
    bool ret = uart_query(UART_2, c); // 非阻塞
    return !ret;
}
```

steer.c

```
#include "steer.h"
#include "headfile.h"
#include "zf_assert.h"
#define my_assert ZF_ASSERT
#include "comm.h"
#include "motor.h"

rt_mutex_t midline_mutex = RT_NULL; // 运行参数锁，多线程间访问数据的时候
一定要先上锁再用!
int midline, yuansu, qianzhan;
bool fache_flag, tingche_flag;
int baohu_count;

#define STEER_PIN ATOM0_CH1_P33_9

//舵机控制相关变量
float steer_kp, steer_kd;
float steer_error;
float steer_kp_base;
float steer_ksteer;
float steer_kp_cone, steer_kd_cone;
int steer_middle_duty;
int steer_ruku_midline = 70;

static void steer_main(void *param) {
    // 舵机
    gtm_pwm_init(STEER_PIN, 200, 3000); // 舵机 200Hz 5ms //ATOM 0 模块的
通道 4 使用 P02_4 引脚输出 PWM  PWM 频率 50HZ  占空比百分之
0/GTM_ATOM0_PWM_DUTY_MAX*100  GTM_ATOM0_PWM_DUTY_MAX 宏定义在
zf_gtm_pwm.h

    // 蜂鸣器
```

```
#define BEEP_PIN P00_8
    gpio_init(BEEP_PIN, GPO, 0, PULLUP); // 蜂鸣器

    while (!fache_flag) {
        pwm_duty(STEER_PIN, 3000 + steer_middle_duty); // 舵机调试用
        rt_thread_mdelay(100); // 等待发车
    }

    // 通讯
    comm_init();

    // 上电通知
    uint8_t payload[] = { 0 };
    bool ret = comm_send_blocking(COMM_TYPE_HELLO_FROM_TC264, payload);
    my_assert(!ret);

    while (1) {
        int last_yuansu, my_midline, my_yuansu, my_qianzhan;
        // 不断尝试获取 EB 回传的打角数据，只有成功获取后才会继续运行
        while (true) {
            comm_type_t type;
            uint8_t payload[10];

            bool last_ret = true;
            bool ret = comm_recv_poll(&type, payload);
            while (!ret) {
                // 清空剩下数据
                last_ret = ret;
                ret = comm_recv_poll(&type, payload);
            }
            ret = last_ret;

            if (!ret) {
                // success
                if (type == COMM_TYPE_PING) {
                    // pong back
                    // printf("pong %02x\n", payload[0]);
                    ret = comm_send_blocking(COMM_TYPE_PONG, payload);
                    my_assert(!ret);
                } else if (type == COMM_TYPE_UPDATE_TO_TC264) {
```

```
                    rt_mutex_take(midline_mutex, RT_WAITING_FOREVER);
                    midline = payload[0];
                    my_midline = midline;
                    last_yuansu = yuansu;
                    yuansu = payload[1];
                    my_yuansu = yuansu;
                    qianzhan = payload[2];
                    my_qianzhan = qianzhan;
                    rt_mutex_release(midline_mutex);

                    break;
                }
            }
            rt_thread_mdelay(1);
        }

        if (tingche_flag) while (1) rt_thread_mdelay(100);
        baohu_count = 0;

        if (my_qianzhan) {// 蜂鸣器
            gpio_set(BEEP_PIN, 1);
        } else {
            gpio_set(BEEP_PIN, 0);
        }

        // 方向策略
        if (my_yuansu == 2) {
            my_midline = steer_ruku_midline;
        }

        // 更新 PID
        // 位置式 pd
        float error_pre = steer_error;
        steer_error = (float)(70 - my_midline);

        // 随速度可变 p
        // int error_strategy = steer_error - error_pre;
        // if (error_strategy <= 25) error_strategy = 25;
        // steer_kp = steer_kp_base + steer_error * steer_error / 1500 *
steer_ksteer * (motor_speed_aim / 1000) * error_strategy / 70 / 3.8;
```

```
        // if (steer_kp >= 25) steer_kp = 25;
        steer_kp = steer_kp_base;

        if (my_yuansu == 3) { // 锥桶
            steer_kp = steer_kp_cone;
            steer_kd = steer_kd_cone;
        } else if (my_yuansu == 4 || my_yuansu == 5) { // 三岔
            steer_kd = 0;
        }

        int result = steer_kp * steer_error + steer_kd * (steer_error -
error_pre);
        int duty = 3000 + steer_middle_duty + result; // 舵机实际输出 pwm
值

        // 限幅
        if (duty > 3800) duty = 3800; // 舵机限幅
        if (duty < 2200) duty = 2200; // 舵机限幅

        // 输出
        pwm_duty(STEER_PIN, duty);

        // 回传速度和里程
        uint8_t speed_send = (double)motor_speed / 6000 * 256; // 压缩一
下
        uint8_t payload[] = {
            (uint8_t)motor_mileage, // 小端序
            motor_mileage >> 8,
            motor_mileage >> 16,
            speed_send
        };
        bool ret = comm_send_blocking(COMM_TYPE_UPDATE_FROM_TC264,
payload);
        my_assert(!ret);
    }
}

static bool steer_inited;

bool steer_init(void) {
```

```
    my_assert(!steer_inited);
    steer_inited = true;

    // 初始化锁
    midline_mutex = rt_mutex_create("midline_mutex", RT_IPC_FLAG_FIFO);
    my_assert(midline_mutex != RT_NULL);

    rt_thread_t tid = rt_thread_create("steer", steer_main, RT_NULL,
        1024, 7, 5); // 创建线程 // 优先级在 motor 和 main 之间
    my_assert(tid != RT_NULL);
    rt_thread_startup(tid); // 启动线程
    return false;
}
```