第十九届全国大学生
智能汽车竞赛

技 术 报 告

学　　校：　　盐城工学院

队伍名称：　　西伏河睡务局

参赛队员：　　杨顺理　杨烨　詹文博

带队老师：　　孟海涛　陈中

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：杨顺理 杨烨 詹文博

带队教师签名：孟海涛 陈中

日　　期：2024 年 8 月 16 日

摘 要

本文介绍了盐城工学院西伏河睡务局在第十九届全国大学生智能汽车竞赛镜头 NXP 组的详细制作方案。省赛车模为逐飞科技公司的 mini 车模、国赛车模为自制车模。以逐飞助手作为上位机，以恩智浦公司生产的微控制器 RT1021 作为下位机，自主构思控制方案及系统设计，包括传感器、信号采集处理、控制算法及执行、动力电机驱动等，最终实现了车辆自主识别路线，并可以实时控制车体状态的智能汽车控制系统。

在制作小车的过程中，我们对小车的整体构架进行了深入的研究，分别在机械结构、硬件和软件上都进行过改进，硬件上主要是考虑并实践各种传感器的驱动电路，软件上我们使用 Python 语言，通过 CCD 摄像头，采用了多种算法结合的方式，采用鲁棒性较强的算法。控制上，通过编码器监测车的实时速度，使用 PID 控制算法调节电机的转速，实现了车模在运动过程中速度的闭环控制。

关键词：智能汽车；恩智浦；Python；CCD 摄像头；PID 控制

Abstract

This article introduces the detailed production plan of the NXP camera group for the 19th National College Student Intelligent Vehicle Competition by the Xifuhe Shuiwuju of Yancheng Institute of Technology. The provincial racing model is a mini car model from Zhufei Technology Company, and the national racing model is a self-made car model. Using the NXP Semiconductors computer and the RT1021 micro-controller produced by NXP Corporation as the lower computer, we independently conceived control schemes and system designs, including sensors, signal acquisition and processing, control algorithms and execution, power motor drive, etc. Finally, we achieved an intelligent automotive control system that enables vehicles to autonomously recognize routes and control the vehicle's status in real time.

In the process of making the car, we conducted in-depth research on the overall structure of the car, and made improvements in mechanical structure, hardware, and software. In terms of hardware, we mainly considered and practiced the driving circuits of various sensors. In terms of software, we used Python and CCD camera, a combination of multiple algorithms and adopted robust algorithms. In terms of control, the real-time speed of the vehicle is monitored through an encoder, and the PID control algorithm is used to adjust the motor speed, achieving closed-loop control of the speed of the vehicle model during motion.

Keywords: intelligent cars; NXP;Python;CCD camera; PID control

目录

第一章 引言

1.1 全国大学生智能汽车竞赛简介

全国大学生智能车竞赛是从 2006 开始，由教育部高等教育司委托高等学校自动化类教学指导委员会举办的旨在加强学生实践、创新能力和培养团队精神的一项创意性科技竞赛，至今已经成功举办了十八届。竞赛是以智能汽车为研究对象，面向全国大学生的一种具有探索性工程实践活动。

该竞赛以“立足培养，重在参与，鼓励探索，追求卓越”为指导思想，旨在促进高等学校素质教育，培养大学生的综合知识运用能力、基本工程实践能力和创新意识，激发大学生从事科学研究与探索的兴趣和潜能，倡导理论联系实际、求真务实的学风和团队协作的人文精神，为优秀人才的脱颖而出创造条件。

该竞赛由竞赛秘书处设计、规范标准硬软件技术平台，竞赛过程包括理论设计、实际制作、整车调试、现场比赛等环节，要求学生组成团队，协同工作，初步体会一个工程性的研究开发项目从设计到实现的全过程。该竞赛融科学性、趣味性和观赏性为一体，以迅猛发展、前景广阔的汽车电子为背景，涵盖自动控制、模式识别、传感技术、电子、电气、计算机、机械与汽车等多学科专业的创意性比赛。该竞赛规则透明，评价标准客观，坚持公开、公平、公正的原则，力求向健康、普康、持续的方向发展。

在继承和总结前十八届比赛实践的基础上，竞赛组委会努力拓展新的竞赛内涵，设计新的竞赛内容，创造新的比赛模式，使得智能车竞赛在新时代更加适应新工科大学生培养理念。

图 1 十九届智能汽车竞赛 LOGO

1.2 智能汽车制作要求及任务

在本次比赛中，小车需要完成绕行赛道任务，基本要求是车模作品从赛道上起跑线出发，绕行赛道一周之后，返回到起跑线。要求车模严格在赛道内运行， 不允许车轮冲出赛道。

其中“镜头组”（分别为 Infineon 赛道、 STC 赛道、 NXP-MicroPython 赛道） NXP-MicroPython 赛道因为编程方式和传统的用 C 语言对单片机进行编程有所不同。 MicroPython 是一个小型的开源 Python 编程语言解释器，运行在微控制器上。使用 MicroPython ，可以编写干净、简单的 Python 代码来控制硬件，而不必使用复杂的底层语言，如 C 或 C++用于编程。 MicroPython 功能非常齐全，支持大多数 Python 语法，一方面可以减轻编程的负担，另一方面能够锻炼当代大学生的专业交叉运用能力。

本组对车模不做限制，我们在华东赛区省赛使用逐飞科技公司的 mini 车模、全国总决赛使用自制差速车模。采用恩智浦公司的单片机作为处理器，通过光电类传感器识别赛道，使得小车自主的在赛道上行驶一圈，速度最快者胜。车模所使用的电路板必须为自行设计制作。比赛必须采用光电类传感器识别赛道。

智能车制作过程中，主要任务分为机械、硬件电路、软件程序三大块。需要对车模的机械结构进行调校，车模调校的结果直接影响着车模的运行状态；硬件电路方面需要自行设计电路板，电路板需要包含电源单元，处理单元，电机控制单元，人机交互单元。必须在电路板铜层上刻上学校名称以及队伍名称；软件方面需要自行设计软件方案，编写程序，控制小车在赛道上行驶。三大部分设计好后需要对智能车进行组装，各部件的安装位置需要经过测量，必须安装对称，否则造成车模重量分配不均，影响运行效果。

第二章 系统总体方案设计

RT1021 微处理器通过采集线性 CCD 摄像头的模拟信号，通过处理得到赛道边沿信息；通过采集磁编码器对车轮转速的脉冲计数，得到车行进的速度数据；通过微处理器对图像处理，对角度、速度进行 PID 控制，最后以 PWM 波输出驱动舵机，电机。在调试过程中，通过无线通信模块、示波器、上位机观察摄像头采集的图像、舵机、电机等实时数据，便于对车模运行状态的监控和调试。

第三章 机械结构设计

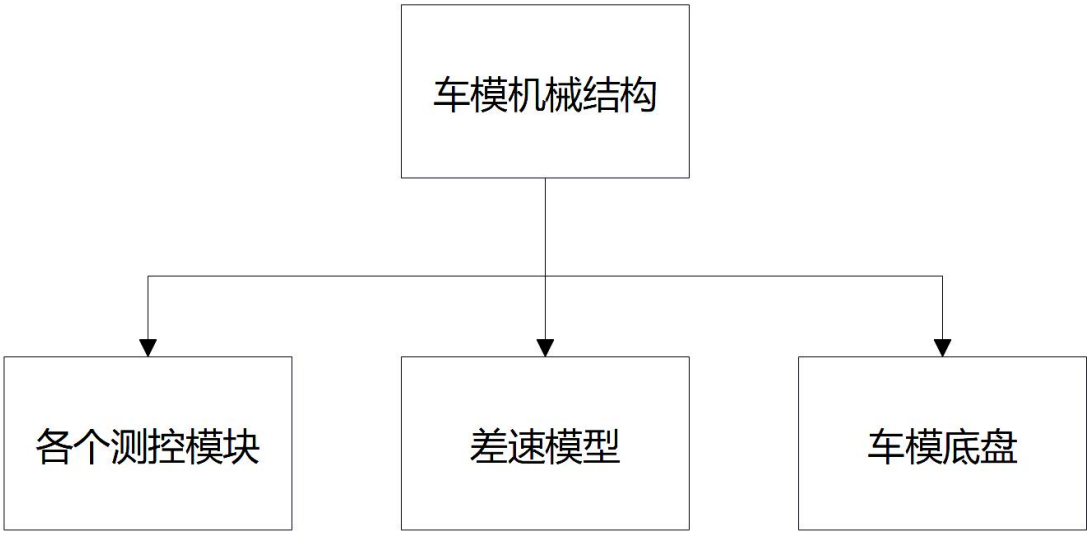

图 2 差速车模机械结构设计

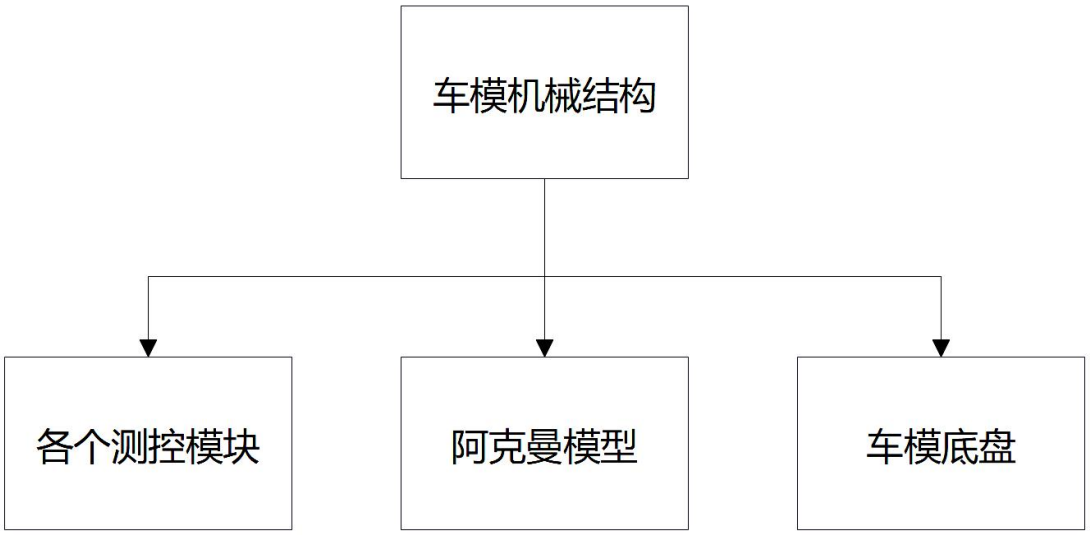

图 3 mini 车模机械结构设计

在华东赛区省赛使用逐飞科技公司的 mini 车模、全国总决赛使用自制差速车模，其中 mini 车模包含了驱动机构、转向机构、车模的底盘三个部分，自制差速车模包含了驱动机构、车模的底盘两个部分。

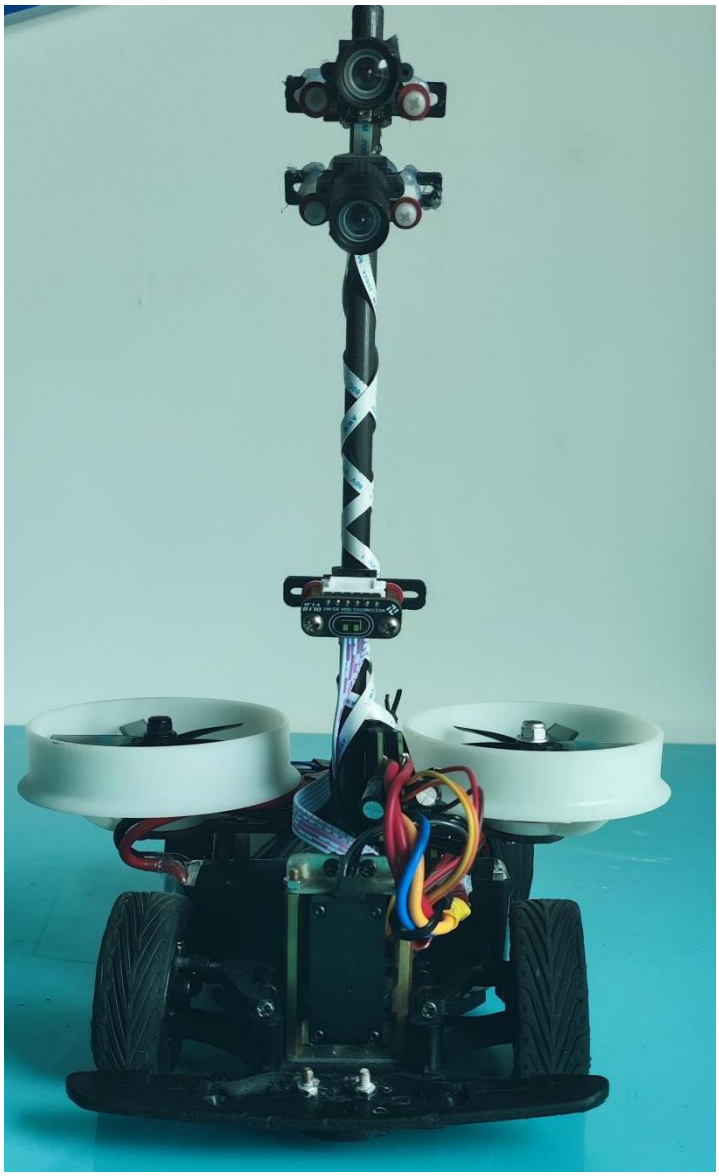

图 4 mini 车模

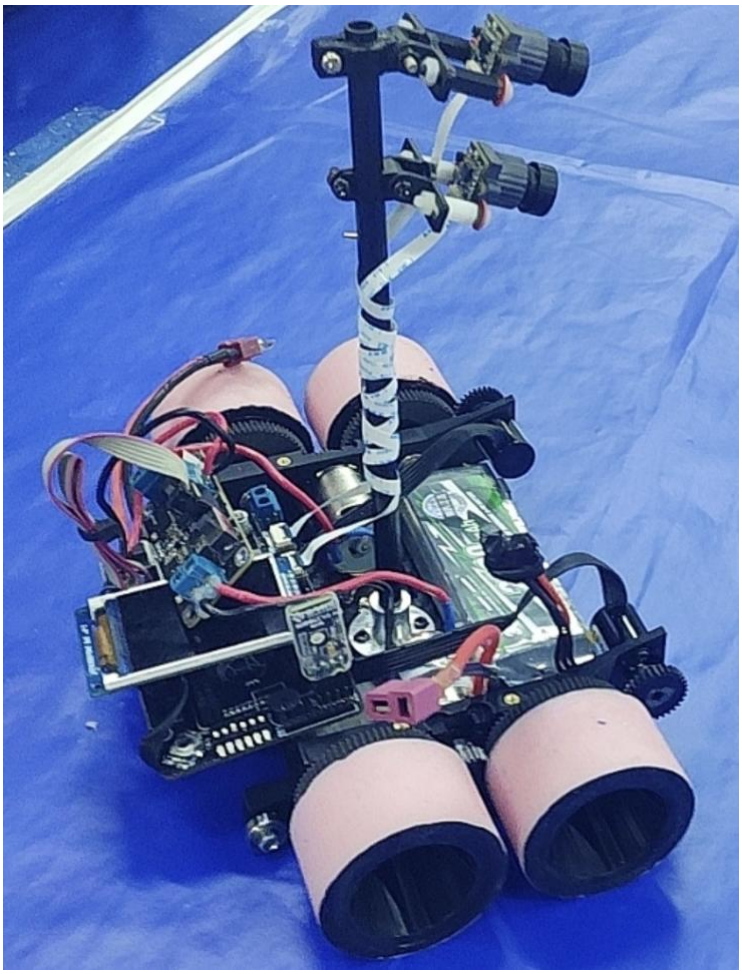

图 5 差速车模

底盘是构成整个车模的主体，其材料由玻璃纤维和碳纤维构成。底盘上搭载着各个模块及零件，是整个车模最为重要的部件之一，底盘的质量直接影响着车模的质量。同时底盘起着支撑整个车体的作用，是整个车模系统的主要承力机构。

底盘上承载了车模的驱动机构，车模采用后轮驱动的形式，驱动电机和传动机构以及驱动轮通过机械连接的方式固定在底盘上。

mini 车模底盘前部承载着车模的转向机构，通过机械连接固定在底盘上。转向机构包含了轮鞘、连杆、轴承几个部分。转向机构安装在底盘的前部，驱动前轮运动，并控制车模的运动方向。其安装角度很有讲究，科学的安装角度直接影响着小车拐弯时的流畅性和车模整体的运行效率。机械上的问题往往都出现在转向机构上，这部分也是车模最脆弱的一部分。

3.1 mini 车模舵机安装

华东赛区省赛我们采用逐飞科技公司生产的 mini 车模，原装的 mini 车模上的转向机构舵机紧贴车模的底盘，并且在我们更换了原装舵机，使用反应速度更快、扭力更大的无刷舵机，发现控制舵机转向的过程中转向幅度较小，在车模高速行驶过程中，势必会造成转向不足情况。我们在前轮传动处固定舵机之家，使用铜柱抬高舵机，在车模高速行驶过程中，舵机转向更容易控制。我们还发现适当增长摆臂的长度和非竖直安装摆臂有助于车模前轮增加倾角，达到前轮内倾，实现前轮的“外八字”的效果。在小车行驶过程中，小车运行在弯道区域，内倾的前轮则能与赛道路面接触更大面积，使得小车过弯时增大摩擦力，不会发生侧翻的风险，也会使得车模运转也会更加平稳。（舵机安装图见图 4）。

在相同转向条件下，转向连杆在舵机一端的连接点离舵机轴心距离越远，转向轮转向变化越快。这相当于增大力臂长度，提高线速度。舵机安装方式有立式和卧式两种，通过比较得立式安装效果更优。舵机采用中置立式安装方式，这样的安装方式使得舵机转向机构、拉力、 行程、反应时间都左右对称，以保证舵机左右转向时力臂相等且处于最大范围，提升舵机响应速度。这样的安装方式也更易于后期机械舵臂拉杆的调试，更利于实现更优转向结果。经理论分析，功率等于速度与扭矩的乘积，加大转向速度必然减少输出扭矩，扭矩过小会造成迟钝，故调试机械舵臂拉杆时须考虑到转向机构的响应速度与舵机扭矩之间的关系，获得最佳转向效果。

因为舵机转向是整个控制系统中延迟最大的一个环节，往往会发生过弯后系统超调难以达到预期目标值的现象，并且阿克曼舵机转向结构需配合后轮差速控制才能达到完美的转向，再加上舵机安装方式和位置具有严格的对称性，且复杂，所以在全国总决赛中我们摒弃了传统的舵机阿克曼转向模型，参考三轮车模的差速结构，设计出仅靠电机差速控制即可实现转向的低延迟差速模型。（差速车模型见图 5）

3.2 差速车模的制作和安装

我们设计的差速模型采用碳纤维板作为小车底盘。传动齿轮使用尼龙材料加工。侧壁和车轮使用 PLA 工程塑料加工，在侧壁挖孔处装填热熔螺母，热熔螺母的安装方法为使用热熔枪缓慢加热螺母，用力缓慢顶入 PLA 工程塑料内部即可。

车轮轮胎为自制硅胶，我们使用树脂光固化打印出倒模模具，对硅胶 AB 胶进行 1：1 混合，使用搅拌器充分搅拌 3 分钟后，放入真空机消除混合胶体的气泡。我们使用的是 15°硅胶，倒入模具后在 16℃空调下冷风吹 8 小时自然晾干 12 小时后，进行脱模处理。轮胎触感柔软，且放在 A4 纸上 10 秒后拿起不粘纸即可。最后使用南大 704 硅橡胶把硅胶胎和车轮紧密连接。车模使用两个 370 有刷电机带动两对传动齿轮和一对编码器齿轮同时转动。通过提升其中一个电机的转速来实现小车的左转和右转。我们还在底板前后各伸出两个轴承，以防止车模高速运行，突然急刹车导致前后倾的问题，同时轴承能够丝滑的滑行减少车模运行的阻力，达到高速行驶的目的。

图 6 硅胶胎制作过程

图 7 硅胶胎安装效果

3.3 减速齿轮和编码器的安装

由于 mini 车模结构的特殊性，以及两个编码器的使用，使得齿轮的安装变得至关重要。齿轮安装的过紧，会加大电机的负载，对于低扭矩的有刷电机，长时间运行会导致过热，有烧坏电机的风险；齿轮安装的过松，会导致齿轮啮合打滑，从而磨损齿轮，也会造成编码器读数不准确，影响电机闭环行驶。因此安装齿轮的标准如下：传动轴和电机轴保持平行。开动电机时，若发出很刺耳的声音，说明间隙过小，有卡齿；若感觉有明显的迟滞现象，则表明齿轮间隙过大。调整好的齿轮传动结构噪声很小，且不会有卡齿的声音。编码器通常车模安装在电机的上方。而差速车模则直接安装在编码器安装槽中，但也要确保齿轮的啮合情况，防止造成不可逆的损伤。

3.4 mini 车模轮胎的保养与使用

mini 车模使用的轮胎为橡胶材质的带花纹轮胎，该轮胎材料太过紧实，而且粘上灰尘抓地力会大大下降，长期使用容易发粘、磨损，但相应的抓地力则会大大加强。虽然在湿滑的路面轮胎纹路能够有效的将水分“锁住”在胎纹内，但对于室内赛道来说胎纹基本上是负提升，因此我们人为磨损轮胎，达到一种长期使用的效果，尽量减少轮胎纹路。

我们将胎皮取出，浸泡在橡胶油内 12 小时，充分软化轮胎表面，然后在车床上通过砂纸进行打磨。我们先使用低目数的砂纸粗磨，后用目数大的细磨，磨轮胎过程中会产生大量白色烟雾（此为轮胎的橡胶颗粒烟尘，有毒需佩戴口罩），轮胎会严重发烫具有粘性，当磨损到胎纹肉眼可见但摸不出来时，即可停止。在日常调试小车后，每天涂抹轮胎软化剂，并且用保鲜膜密封一个晚上。经过一个月左右时间，轮胎胎纹基本不可见，轮胎摩擦力大大增加。

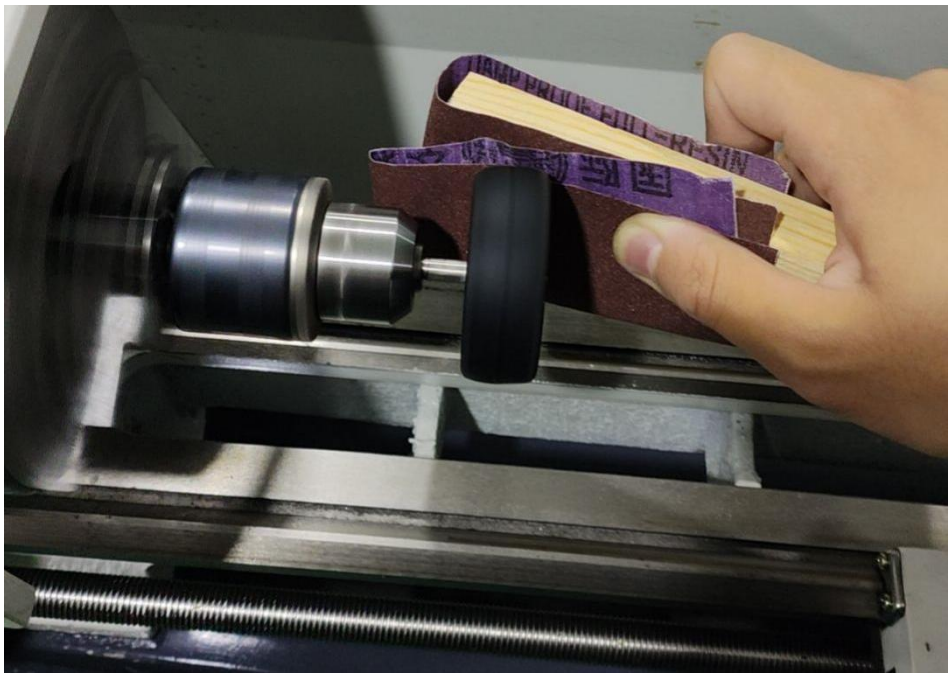

图 8 使用车床磨轮胎

3.5 重心位置的调整、电池的放置和负压的安装

智能车在做圆周运动时，会产生一定的离心力，若智能车转弯半径一定时，速度越快，离心力越大，会导致智能车的漂移甚至翻车，严重影响智能车的稳定性。为了提高安全性以及稳定性，需降低智能车重心，使底盘最大限度降低，

且不能触碰到地面。智能车上的器件安装也应尽可能放低。且保证整车的质量分布均匀。

电池的放置方式有两种，电池前置和电池后置。由于车身尾部的质量要远远大于前端的质量，导致后轮转向的转动转矩过大，这样的转矩会使车模在急弯处发生偏移、侧滑，影响其稳定性。由于我队电机和电路板安装在小车尾部，为了保证质量分布合理因此采用电池前置的方式。使得整体重心处于小车正中心也就是摄像头杆子的安装位置，大大提高了车轮行驶的稳定性。

为了防止高速运行的小车因气流影响产生远离地面的效果，我们在小车重心的位置，横向安装了两个负压风扇，通过反向吸风，将小车牢牢压在地面，减小车轮的打滑，实现小车的稳定运行大大提高系统的可靠性。

图 9 负压安装示图

3.6 摄像头的安装

降低整车重心，需严格控制线性 CCD 摄像头的安装位置和重量，我们使用了轻巧的塑料三通组件连接碳素杆作为安装摄像头的主桅，且主桅的高度不超过 25cm，这样可以获得最大的刚度质量比，整套装置具有很高的刚度。摄像头便于拆卸和维修，同时具有硬件运放旋钮调节整体曝光度，具有赛场快速保障能力。为了防止车模在过颠簸路段时，碳素杆发生抖动，我们在碳素杆底部，使用铝合金固定件，牢牢固定在车模底板上。

第四章 系统硬件设计

我们主要从系统的稳定性、可靠性、高效性、实用性、简洁性等方面来考虑硬件的整体设计。从最初方案设定到最终方案的敲定，我们经历各种讨论与大的改动才有了如下的硬件。

这辆小车电路部分主要的模块包括：电源管理模块、传感器模块、驱动模块以及人机交互模块。各模块的总体设计原则是：稳定可靠且符合车模自身结构。

为了尽量减轻车模的负载，降低模型车的重心位置，应使电路设计尽量简洁，尽量减少元器件使用数量，缩小电路板面积，使电路部分使电路部分重量轻，易于安装。在设计完原理图后，注重 PCB 板的布局，优化电路的走线，整齐排列元器件。

4.1 传感器的选择

4.1.1 摄像头的选择

相对于 CMOS 而言， CCD 摄像头体积小、功耗低、使用较少的引脚资源、图像简单易处理等特点，有利于车模的图像处理和高速运行。

目前市面上的摄像头主要分为硬件运放和软件曝光两种。大多数摄像头都可以很好的实现单片机与摄像头之间的交互。

CCD 摄像头 TSL1401 介绍：

我们选用的摄像头是 CCD 摄像头，这是一款基于 TSL1401 芯片设计的传感器模块，是一款在智能车竞赛市面上性能最优，最适合高速情况下采集一行图像的线性摄像头。它主要由以下几个优点：

●硬件 AD 调节曝光

我们的摄像头具有可调曝光功能，当环境变亮，我们可以通过调节硬件运放来减小曝光时间，当环境变暗增加曝光时间。所以此款摄像头可以适应不同环境。

●全白底色图像稳定

我们的摄像头图像区分度明显，更有利于我们对采集到的数据进行处理。

●更小更轻更强悍

我们的摄像头尺寸仅有 3（宽度） * 2（长度） * 2.5（高度），方便安装。摄像头质量很小，架在碳素杆上，重心不会特别偏上。

图 10 CCD 摄像头模块

4.1.2编码器的选择

为了使更加精准的反馈速度，我们选用了普地公司生产的 1024 线编码器。

该编码器具有以下特点：

●更小更轻更强悍

mini 编码器，使用 FC 2.54mm 2 * 3 端子，尺寸外壳仅有 15mm 长 18.5mm。

●直接方向输出

输出三线步进脉冲，可分辨正转和反转。

●TMR 技术高转速

使用车规级 DSP 内核，转速可达 15000 RPM。

●抗磁性

编码器使用强钕磁铁外壳，使其具有抗磁性。有效阻隔有刷电机的磁场干扰，即使紧贴电机安装，也能输出正确的时序。

图 11 1024 线编码器

4.2 电路模块设计

4.2.1 电源管理模块的设计

电源管理模块上我们有产生 3.3V 的 LDO 模块和产生 5V 的 DCDC 模块两种，分别使用 RT9013-33GB 和 SCT2450STER 芯片。

RT9013-33GB 是一款输出电压固定的稳压器，最大输入电压可达 7V，使用 SOT-23-5 封装。输出电流 500mA ，输出电流 500mA 时，最小输入输出电压差小于等于 1V。工作温度 -40 ～ +85℃。

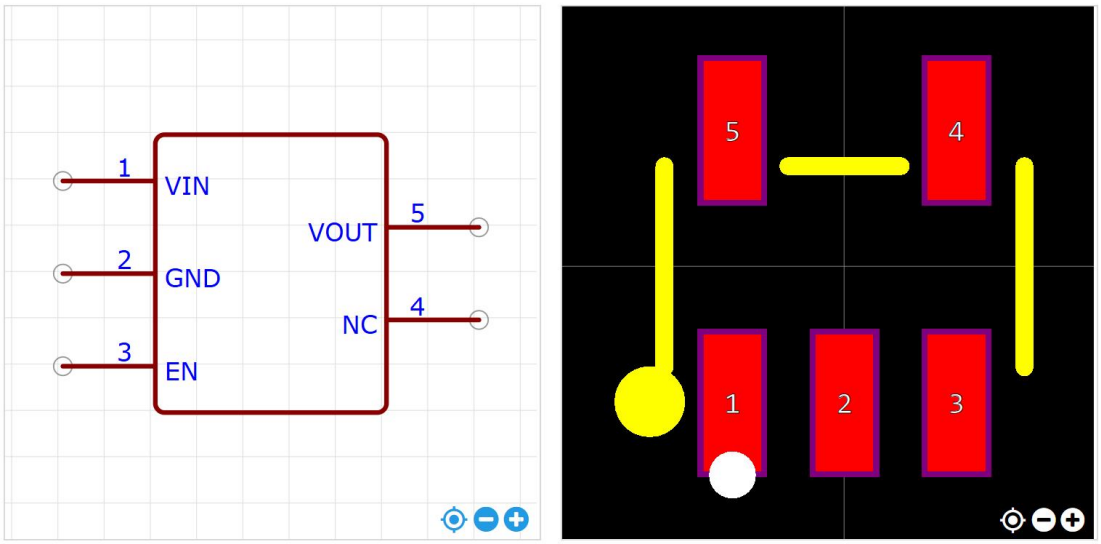

图 12 RT9013-33GB 线性稳压器

SCT2450STER 是一款可编程频率的高效同步降压 DCDC 转换器，最大输入电压 3.8V - 36V，使用 ESOP-8 封装。输出 5A 持续电流，工作温度 -40 ～ +150℃。输出电压范围 0.8V - 36V。

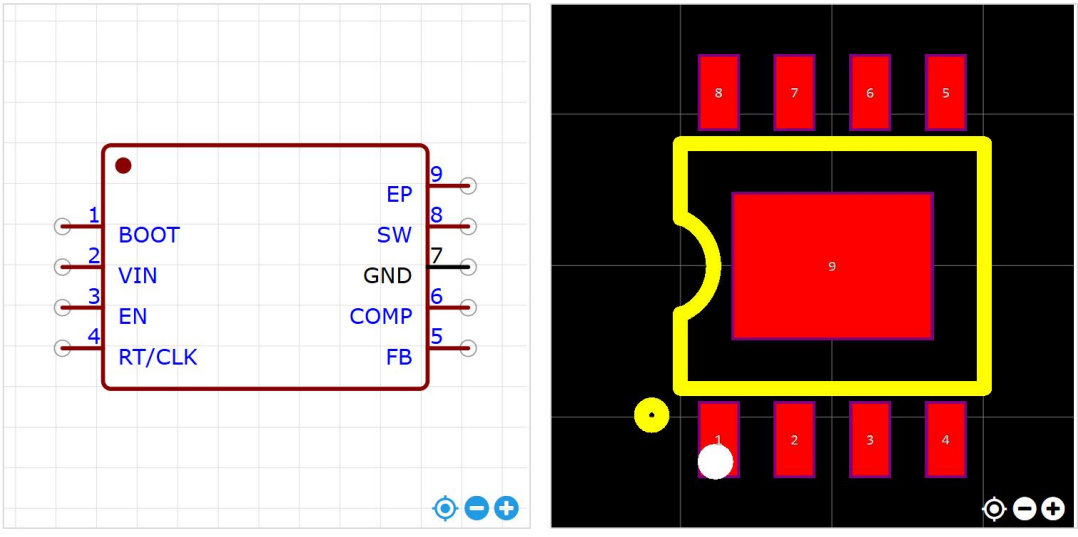

图 13 SCT2450STER 可编程频率的高效同步降压 DCDC 转换器

4. 2. 2 图像采集模块

图像采集模块我们使用 CCD 摄像头采集赛道图像，摄像头数据以模拟信号的形式输出，摄像头接口和单片机之间使用串行通信的方式传送数据。为了增大传输速度，图像数据通过单片机的 DMA 接口传输。图像采集模块电路如图 4.2 所示。

4.2.3 电机驱动模块

我们需要驱动 2 个有刷电机驱动行进轮，2 个无刷电机驱动负压，我们使用四个 MOS 管构成 H 桥式电路以及 DRV8701ERGER 和 FD6288Q 两个电机驱动芯片来驱动有刷和无刷电机。由于我们的两个车模行进轮电机都是 370 有刷电机、负压都是 RS2205 无刷电机，它们功率较大，工作电流也较大。为了防止电机驱动电路对单片机造成干扰，在驱动电路的输入级接入了逻辑芯片进行隔离。如图 14 所示。

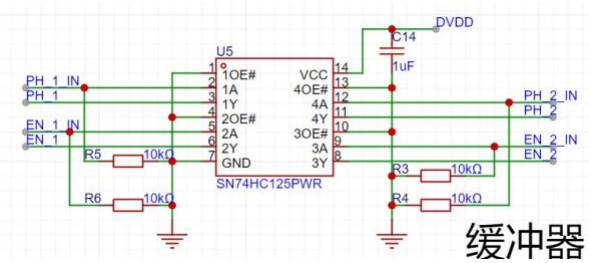

图 14 隔离电路

4.2.4 人机交互模块

我们为了方便控制，在硬件电路中增添了人机交互模块。包括：TFT 屏幕、蜂鸣器、五向开关、拨码开关以及无线串口，用于通过按键和上位机与单片机交互，达到快捷控制的目的。

第五章 智能车控制软件设计说明

我们的智能车系统分为赛道的图像处理和速度控制两个主要的部分。图像处理的合理性为小车系统在赛道上提供了连续有效的偏差，速度控制的准确性为小车行驶的稳定性提供了保障，这两者的有机结合，使得智能车有更优的行驶路径和更好的鲁棒性。

5.1 赛道中线提取及优化

5.1.1 图像特征

CCD 摄像头可以采集线阵信息，能够很具体显示出一行赛道的特征。但有时因为赛道材质或外界光线影响，原始图像会出现诸多的尖峰，会干扰对赛道图像的处理。因此在获取到原始图像后需要对图像进行简单的滤波处理或者加上偏振片来减少不必要的干扰。

图像的有效信息主要包括：赛道边沿，赛道的中线和赛道的类型。

将摄像头采集的图像经过软件处理：将得到各个点与阈值比较，实现软件的二值化处理。

该届赛道的复杂元素包括十字及环岛。CCD 摄像头对其处理较困难，因此我们采取使用两个摄像头的方法。

5.1.2 赛道中线获取

赛道边沿提取方式如下：

（1）遍历 128 个像素点，选取合理的数值区间，取出一个最大值和一个最小值，进行一次算数平均处理，得到一个动态阈值。

（2）将 128 个像素点的值分别与阈值进行比较，大于等于阈值的设定为白（0xFFFF）小于等于阈值的设定为黑（0x0000）。

（3）第一次将图像由标定中点向左右两边进行扫描，从第二次开始，从上一次中点向左右两边扫描。确定赛道的左右边界位置后，向两边扫描得到黑白跳变点，作为原始边沿，若扫描不到黑白跳点，则视为丢线，人为补定边界为 0（左边点）或 127（右边点）。

（4）在得到原始边沿的同时得到赛道的宽度，用于作为特殊元素的情况判断。

（5）利用得到的左右边点计算赛道中点。

（6）通过得到的赛道中点与人为标定中点对比，得到两个 CCD 摄像头的图像误差，方便接下来的操作。

5.2 赛道路径处理

5.2.1 有效偏差提取

为了很好的获得赛道偏差，需要拟合出各个赛道下的中心线。当两边都不丢线时通过简单的边沿求和平均取中点即可。当出现一边丢一边存在的情况需要做特别的处理。

1.两边边沿都存在的情况处理如下：

计算方式： (左边点 + 右边点) / 2 = 中点;

2.单边存在情况处理如下：

计算公式如下：

●若左丢线：

(其中一个未丢线 CCD 的左边点 +0) / 2 = 左边点

(一个 CCD 的右边点 + 另一个 CCD 的右边点) / 2 = 右边点;

●若右丢线：

(一个 CCD 的左边点 + 另一个 CCD 的左边点) / 2 = 左边点

(其中一个未丢线 CCD 的右边点 +127) / 2 = 右边点;

在拟合到合理的中线后需要对中线进行处理提取有效的偏差量，进而送入 PD 控制器实现对舵机或者差速轮的控制。为了获得更好的运动路径都需要满足：偏差在一定区间需要有一定的连续性。换而言之，猛然的阶跃偏差会导致小车运动的剧烈抖动，在高速下很有可能冲出赛道。

5.2.2 元素判断

小车能否以最短的时间完成比赛，与小车的速度和元素识别都有着密切的关系，因此，如何使小车准确识别出对应的元素完成比赛是提高平均速度的关键。

以下是我们图像处理的思路：

（1）斑马线

1.扫描像素，寻找该行像素中的黑白跳变点，记录黑白交界点的个数和它们的横坐标，用于后续判定；

2.逐行判定当黑白跳变是否大于一定次数，若是则预判定该行具有斑马线特征，进入均匀性判断；

3.判断该行的黑白交界段数是否在指定范围内，若是则判定该行具有斑马线特征，行特征计数器+1；

4.若斑马线特征行数计数器大于设定阈值，则判定该帧存在斑马线。

（2）十字

1.检测是否有一个图像两侧是否存在丢线；

2.检测图像中线上的黑色部分宽度，如果很长就终止判断；

3.检测两行图像左右边点差值是否相差很大（至少有一个摄像头两侧丢线，此时两个摄像头左边点差值和右边点差值均会差别很大）；

（3）弯道与直道

1.通过两个图像中点误差的绝对值之和和绝对值之和的变化率来判断。

2.在弯道时，绝对值之和是非常大的，但绝对值之和的变化率很小几乎为零。

3.在直道时，绝对值之和是很小几乎为零。

4.在直道进入弯道时，绝对值之和以及绝对值之和的变化率同时发生变化，且都有增大的趋势。

（4）坡道

1.坡道可分为上坡和下坡，在坡道时，IMU 陀螺仪加速度数据变化很大。同时通过解算出来的 Pitch 数据，上坡时大于零，下坡小于零。

2.即将进入坡道时候，可以通过 TOF 测距模块进行预判断。（由于我们对丢线进行了特殊处理，此过程也可以忽略）

（5）环岛

1.上面摄像头单边丢线，记作状态 1。

2.下面摄像头单边丢线，记作状态 2，此时小车刚到达环岛第一路口，对丢线端进行补线操作（丢线点 = 有线点 ± 赛道宽度）。同时用编码器记录脉冲为一个标准赛道宽度的距离，记下状态 3。

3.在状态 3 时，小车行驶到环岛中间，此时对圆环记作有线端，然后对另一边虚空补线。小车即可自行驶入环岛，同时对 IMU 陀螺仪解算的 Yaw 轴积分。

4.当积分值达到固定阈值时停止积分，对上述所有标志位清零。小车此时应该处在环岛中间，此时记作状态 4。

5.对最后一个路口进行补线处理，补线时用编码器记录距离为一个标准赛道宽度，完成出环岛。

（6）路障

由于室内赛道上的路障是由和标准砖头尺寸（240mm×115mm×53mm）相同的长方体构成。颜色为黑色。距离赛道中心距离 10cm 。因为我们的车模宽度较小，我们通过中点误差即可避开，也可以稳定车身，直接形式过去。

第六章PID控制算法

6.1 PID控制算法介绍

在工程实际中，应用最为广泛的调节器控制规律为比例、积分、微分控制，简称 PID 控制，又称 PID 调节。PID 控制器问世至今已有近 70 年历史，它以其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一。当被控对象的结构和参数不能完全掌握，或得不到精确的数学模型时，控制理论的其它技术难以采用时，系统控制器的结构和参数必须依靠经验和现场调试来确定，这时应用 PID 控制技术最为方便。即当我们不完全了解一个系统和被控对象，或不能通过有效的测量手段来获得系统参数时，最适合用 PID 控制技术。

PID 控制属于一种线性控制方式，通过给设定的目标值与实际值进行相减算出偏差。将偏差进行比例、积分和微分算出控制量控制对象。原理框图如图 15 所示。

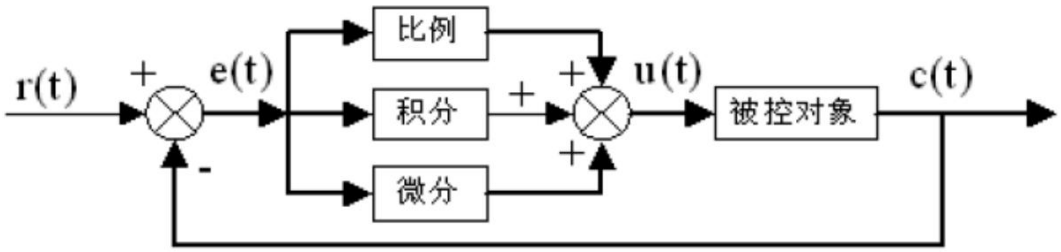

图 15 PID 控制原理框架

PID 基本理想公式为：

$$U(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(t)\, dt + T_d \frac{de(t)}{dt} \right] \qquad (6\text{-}1)$$

U(t) 为计算的输出结果；

e(t) 为实际值与设定值的偏差，及输入量；

KP 为 PID 中 P 的值即比例系数；

TI 为 PID 系统时间积分常数；

TD 为 PID 系统微分时间常数；

PID 控制器各环节作用如下：

1.比例环节：将实际值与目标值相减获得的偏差进行比例放大或缩小获得偏差信号，一旦系统中产生偏差，控制器立即进行控制减少偏差；

2.积分环节：用于消除静态误差，减少系统误差。PID 系统中积分时间常数越大，积分作用越小，积分时间常数越小，积分作用越大；

3.微分环节：能够预测变化趋势，及时修正系统输入信号量，提高系统的反应速度。

6.2 车模速度控制

经过查询和咨询，在速度控制上，采用左右轮分别使用增量式 PID 控制。使用增量式 PI 算法，可将速度迅速达到自己设定的目标速度，实现智能车在直道加速，弯道减速的功能。两轮分开使用增量式 PID 可消除左右电机自身误差，且可用于左右轮主动差速控制，优化过弯路径。

增量式 PID 公式如下：

$$\Delta U(k) = U(k) - U(k-1) = K_p \Delta e(k) + K_i e(k) + K_d[\Delta e(k) - \Delta e(k-1)] \quad (5\text{-}4)$$

式中：Δe(k) = e(k) - e(k - 1)

在实际测试中发现，智能车只有在弯道以及部分特殊路段会加减速，其余路段均为匀速，即速度稳定变动不大，所以简化增量式 PID 公式，将 KD 参数置 0，无需微分部分。其增量式 PI 控制流程如图 16 所示。

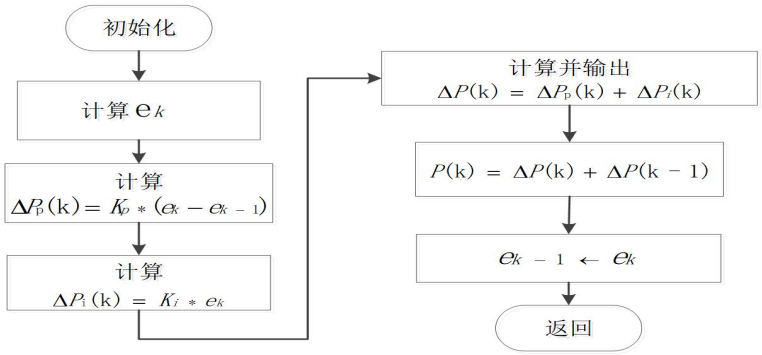

图 16 增量式 PI 控制流程

因使用左右轮分别控制，便加入主动差速控制。主动差速可根据转向控制

数值计算出左右后轮的速度，实现差速控制。

6.3 车模转向控制

智能车的方向控制是根据图像获得的数据控制舵机打角（差速车模则是通过获得的数据传入 PID 控制器得到左右轮的速度偏差）。根据查询相关资料，获知融合处理后的数据与输出值成一次线性关系，最简单的方法为，建立数据与角度的函数关系，但实际测量发现，只有速度较慢的情况下才能跑完赛道，速度较快时，智能车左右晃动严重，不符合智能车需求。

经过多种方式测试以及老师指导，发现位置式 PID 适用于获取的数据与输出值关系的计算。

设 U(k)为第 k 次计算的输出结果，根据式（6-1）推出位置式 PID 公式为：

$$\mathrm{U}(k) = K_p e(k) + K_i \sum_{j=0}^{k} e(j) + K_d[e(k) - e(k-1)] \qquad (6\text{-}2)$$

式中：

$K_i = \frac{K_p T}{K_i}$ 为积分系数；

$K_d = \frac{K_p T_d}{T}$ 为微分系数；

由于不存在静态误差，所以在实际使用中积分系数 Ki 设为 0。根据舵机实际情况，将位置式 PID 公式变化为位置式 PD 公式，见式 6-3。

$$\mathrm{U}(k) = K_p e(k) + K_d[e(k) - e(k-1)] + X \qquad (6\text{-}3)$$

式中：

X 为偏差为 0 时，控制量的基础值，即输出为 0。

第七章 系统的调试与运行

7.1 Thonny调试软件

Thonny 是基于 python 内置图形库 tkinter 开发出来的支持多平台包括：windows, Mac, Linux 等的 python IDE，支持语法着色、代码自动补全、debug 等功能。在软件中可以通过 Shell 窗口实时监测到程序中各个变量的值，这对于项目的开发起到了很大的作用，缩短了项目开发的周期。

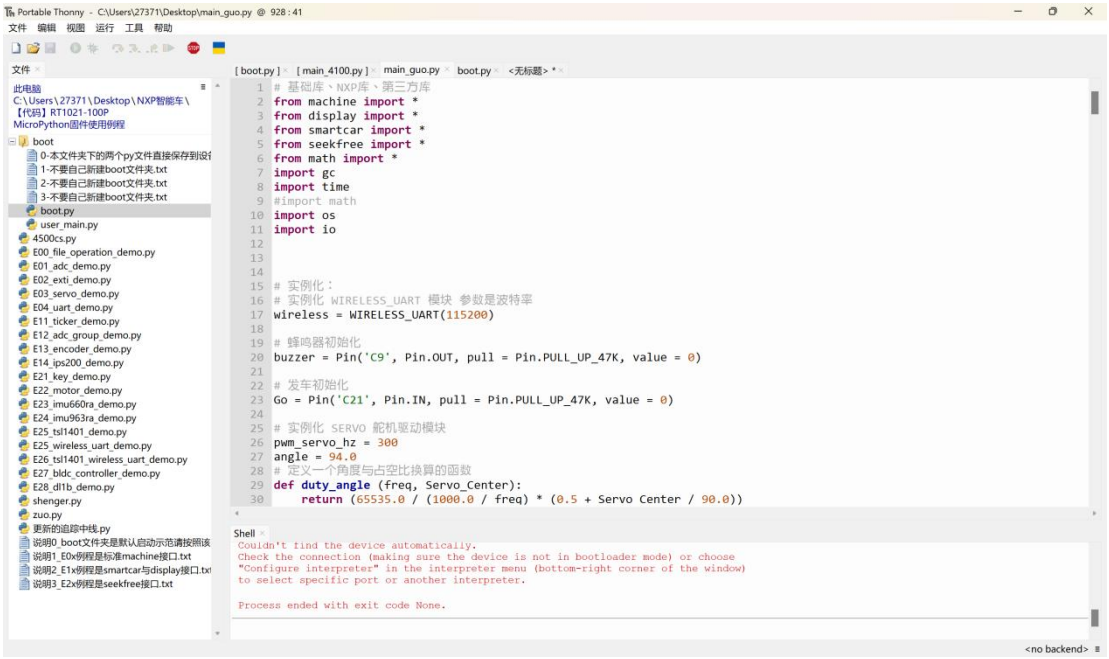

图 17 Thonny 编程界面

7.2 串口调试

为了能获取 CCD 摄像头采集到的图像和输出的虚拟示波器图像，采用无线串口连接电脑的上位机软件——逐飞助手。上位机软件如图 18 所示。

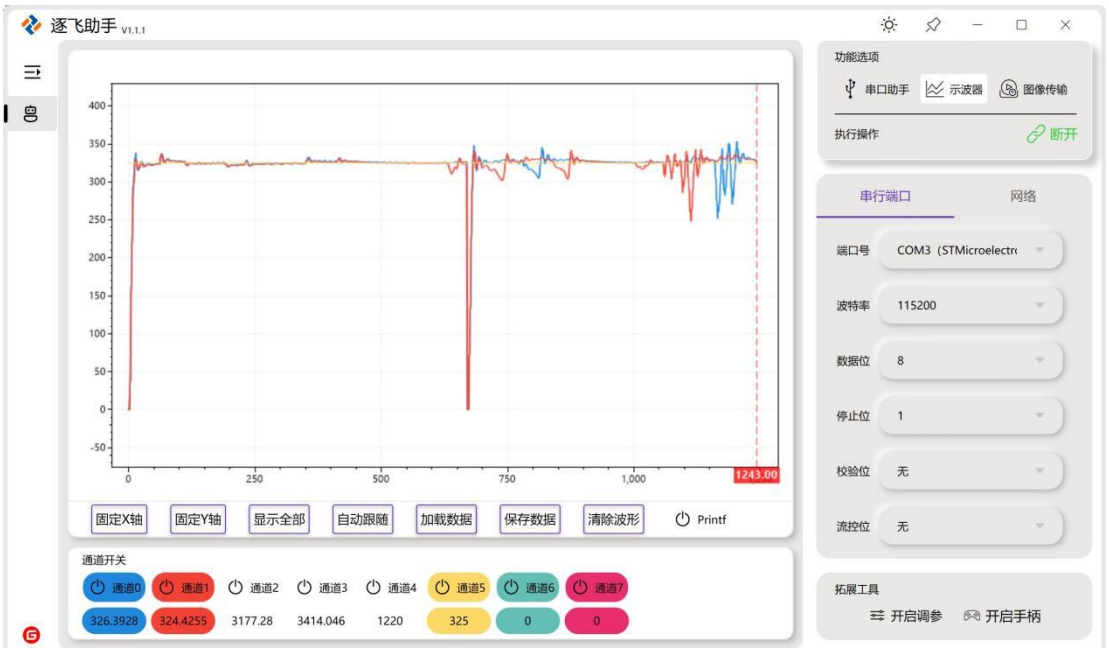

图 18 逐飞助手上位机

7.3 UI 界面调试

为了方便比赛时调试程序部分参数的便利，设计了一款可供调试的菜单，菜单中设有一些比赛时需要经常修改的参数，如下图所示。

图 19 UI 界面

第八章 模型车的主要技术参数说明

车模整体尺寸	长	215
	宽	200
	高	280

表 1 车模整体尺寸

传感器	Mini 车模	差速车模
	CCD 摄像头	CCD 摄像头
	IMU 陀螺仪	IMU 陀螺仪
	编码器	编码器
		灰度传感器

表 2 传感器类型

PCB 电路板	Mini 车模	差速车模
	主板	主板
	有刷电机驱动板	有刷电机驱动板
	无刷电机驱动板	无刷电机驱动板

表 3 PCB 电路板

总结

自十九届全国大学生智能车竞赛发布以来，我和小组成员反复研究，多次实 验最终确定了这个方案。在这份技术报告中，包含了这一年来的调试的思路和方法，包括机械、硬件、软件等方面。机械方面，充分考虑了车模运行可能出现的受力和力矩变化情况。硬件电路的设计稳定。软件算法上力求创新，在保证其稳定的同时，力求突破车模运行的极限。编程语言为 python 语言，开发工具为 Thonny。感谢这几个月来学院和老师的大力支持，同时也十分感谢参与指导的学长和学院领导。在此特别感谢比赛组委会能组织这样一项有意义的比赛。通过这个比赛学到的东西有很多，但还是远远不够的。我们要在今后的生活里继续钻研，努力学习，为祖国智能化，自动化的发展做出贡献。

参考文献

[1] 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.

[2]胡健等.单片机原理及接口技术实践教程[J].机械工业出版社，2004.

[3]李心广等.电路与电子技术基础.机械工业出版社[J]，2008.

[4]阎石.数字电子技术基础[J].高等教育出版社，2006.

[5]李小坚.郝晓丽.Protel DXP 电路设计与制版使用教材[M],人民邮电出版社，2015.

[6] 仲志丹，张洛平，张青霞. PID 调节器参数自寻优控制在运动伺服中的应用[J] . 洛阳工学院学报，2000，21（1）：57～60.

[7]熊中华. 基于摄像头传感器的智能车循迹算法设计方案[J]. 电子产品世界,2022,29(7):69-73.

[8]漆小华.基于模糊 PID 算法的三轮智能车应用[J].长江信息通信,2024,37(02):141-144.DOI:10.20153/j.issn.2096-9759.2024.02.043.

[9]邹华跃. 《数字集成电路基础学习参考》[M]. 南京大学出版社，2011.

[10]童诗白，华成英. 《模拟电子技术基础》[M]. 高等教育出版社，2014.

附录A：源代码

```
# 基础库、NXP 库、第三方库
from machine import *
from display import *
from smartcar import *
from seekfree import *
from math import *
import gc
import time
#import math
import os
import io



# 实例化：
# 实例化 WIRELESS_UART 模块 参数是波特率
wireless = WIRELESS_UART(115200)
# 蜂鸣器初始化
buzzer = Pin('C9', Pin.OUT, pull = Pin.PULL_UP_47K, value = 0)
# 发车初始化
Go = Pin('C21', Pin.IN, pull = Pin.PULL_UP_47K, value = 0)
# 实例化 SERVO 舵机驱动模块
pwm_servo_hz = 300
angle = 94.0
# 定义一个角度与占空比换算的函数
def duty_angle (freq, Servo_Center):
    return (65535.0 / (1000.0 / freq) * (0.5 + Servo_Center / 90.0))
duty = int(duty_angle(pwm_servo_hz, angle))
pwm_servo = PWM("C20", pwm_servo_hz, duty_u16 = duty)
# 实例化 MOTOR_CONTROLLER 电机驱动模块
motor_l =
MOTOR_CONTROLLER(MOTOR_CONTROLLER.PWM_C25_DIR_C27, 13000,
duty = 0, invert = True)
motor_r =
MOTOR_CONTROLLER(MOTOR_CONTROLLER.PWM_C24_DIR_C26, 13000,
duty = 0, invert = True)
# 实例化 encoder 模块
encoder_l = encoder("D0", "D1", True)
encoder_r = encoder("D2", "D3")
```

```
# 实例化 bldc 模块
bldc1 = BLDC_CONTROLLER(BLDC_CONTROLLER.PWM_B26, freq = 300,
highlevel_us = 1100)
bldc2 = BLDC_CONTROLLER(BLDC_CONTROLLER.PWM_B27, freq = 300,
highlevel_us = 1100)
high_level_us = 1300
dir = 1
# 实例化 lcd 模块
cs = Pin('C5', Pin.OUT, pull = Pin.PULL_UP_47K, value = 1)
cs.high()
cs.low()
rst = Pin('B9', Pin.OUT, pull = Pin.PULL_UP_47K, value = 1)
dc   = Pin('B8', Pin.OUT, pull = Pin.PULL_UP_47K, value = 1)
blk = Pin('C4', Pin.OUT, pull = Pin.PULL_UP_47K, value = 1)
drv = LCD_Drv(SPI_INDEX = 1, BAUDRATE = 60000000, DC_PIN = dc,
RST_PIN = rst, LCD_TYPE = LCD_Drv.LCD200_TYPE)
lcd = LCD(drv)
lcd.color(0xFFFF, 0x0000)
lcd.mode(1)
lcd.clear(0x0000)
# 实例化 IMU660RA 模块
imu = IMU660RA()

# 核心板上的 LED
led1 = Pin('C4' , Pin.OUT, pull = Pin.PULL_UP_47K, value = True)
# 拨码开关 2
end_switch = Pin('C19', Pin.IN, pull=Pin.PULL_UP_47K, value = True)
# 拨码开关 4
switch_3 = Pin('B14', Pin.IN, pull=Pin.PULL_UP_47K, value = True)
# 拨码开关 3
switch_4 = Pin('B15', Pin.IN, pull=Pin.PULL_UP_47K, value = True)
# 调用 TSL1401 模块获取 CCD 实例
ccd = TSL1401(3)
# 实例化 KEY_HANDLER 模块
key = KEY_HANDLER(10)
ticker_flag = False
# 定义一个回调函数
def time_pit_handler(time):
    global ticker_flag
    ticker_flag = True
```

```
# 实例化 PIT ticker 模块
pit1 = ticker(1)
pit1.capture_list(ccd, encoder_l, encoder_r)
pit1.callback(time_pit_handler)
pit1.start(5)
ticker_flag_3ms = False
# 定义一个回调函数
def time_pit_3ms_handler(time):
    global ticker_flag_3ms
    ticker_flag_3ms = True

# 实例化 PIT ticker 模块
pit2 = ticker(2)
pit2.capture_list(imu, key)
pit2.callback(time_pit_3ms_handler)
pit2.start(3)
ticker_flag_1000ms = False
# 定义一个回调函数
def time_pit_1000ms_handler(time):
    global ticker_flag_1000ms
    ticker_flag_1000ms = True

# 实例化 PIT ticker 模块
pit3 = ticker(3)
pit3.capture_list()
pit3.callback(time_pit_1000ms_handler)
pit3.start(1000)
# 初始化变量
ccd_data1 = [0] * 128          # ccd1 原始数组
ccd_data2 = [0] * 128          # ccd2 原始数组
encl_data = 0                  # 左编码器数据
encr_data = 0                  # 右数据编码器
tof_data = 0                   # TOF 数据
key_data = [0] * 4             # 按键数据
threshold1 = 0                 # ccd1 阈值
threshold2 = 0                 # ccd2 阈值
image_value1 = [0] * 128       # ccd1 的二值化数组
image_value2 = [0] * 128       # ccd2 的二值化数组
Mid_point1 = 0                 # ccd1 的中点
```

```
Mid_point2 = 0                  # ccd2 的中点
error1 = 0                      # ccd1 的误差
error2 = 0                      # ccd2 的误差
out = 0                         # 舵机输出值
aim_speed_l = 0                 # 左轮期望速度
aim_speed_r = 0                 # 右轮期望速度
output_encl = 0                 # 左轮编码器滤波
output_encr = 0                 # 右轮编码器滤波
out_l = 0                       # 左轮输出值
out_r = 0                       # 右轮输出值
data_change_flag = 0            # 数据修改标志位
main_menu_item = 1              # 主菜单选择第几行
sec_menu_item = 2               # 二级菜单选择的第几行

n = N = 0                   # 左圆环标志
m = M = 0                   # 右圆环标志
distance_l_1 = 0            # 左圆环路程标志
distance_l_2 = 0            # 左圆环路程标志
distance_r_1 = 0            # 右圆环路程标志
distance_r_2 = 0            # 右圆环路程标志
distance_T = 0              # 总路程标志
grzo_z = 0                  # 陀螺仪 z 轴值
grzo_z1 = 0
distance_a = 0
# 用户函数：
# 二值化处理
def ccd_image_value(ccd_data, value):
    ccd_value = [0] * len(ccd_data)    # 定义一个空二值化列表
    for i in range(len(ccd_data)):     # 遍历 0-127 点
        temp_num = ccd_data[i]         # 暂存列表
        if temp_num > value:           # 比较阈值大小
            ccd_value[i] = 1           # 大于阈值为 1
        else:
            ccd_value[i] = 0           # 小于为 0
    return ccd_value
#获得动态阈值
def ccd_get_threshold(ccd_data):
    value_max = ccd_data[4]            # 从第 5 个元素开始考虑最大值
    value_min = ccd_data[4]            # 从第 5 个元素开始考虑最小值
```

```
    #遍历 5-122
    for i in range(5, 123):
        value_max = max(value_max, ccd_data[i])    # 限幅在最大传入数据和第 5
个元素值上
        value_min = min(value_min, ccd_data[i])    # 限幅在最小传入数据和第 5
个元素值上

    threshold = (value_max + value_min) / 2        # 计算阈值
    threshold = min(max(75, threshold), 255)       # 阈值限幅在 75-256 之间
    return threshold


# 得到中点
left_point_1 = False
right_point_1 = False
last_mid_point_1 = 0  # 上次中点，应在函数外部初始化
mid_point_1 = 64        # 当前中点，应在函数外部初始化
def get_ccd1_mid_point(bin_ccd):
    global last_mid_point_1, mid_point_1, left_point_1, right_point_1
    # 搜索左边点，以上次中点作为这次的起搜点
    for l_point1 in range(last_mid_point_1, 1, -1):
        if bin_ccd[l_point1 - 1] == 0 and bin_ccd[l_point1] == 1:     # 判断是否与
左边点一致
            left_point_1 = l_point1       # 左边点找到
            break
        elif l_point1 == 1:       # 如果找到 1 都没找到
            left_point_1 = 0     # 强制令左边点为 0
            break

    # 搜索右边点，以上次中点作为这次的起搜点
    for r_point1 in range(last_mid_point_1, 126):  # 注意这里应该是 128，因为索
引是从 0 开始的
        if bin_ccd[r_point1] == 1 and bin_ccd[r_point1 + 1] == 0:     # 判断是否与
右边点一致
            right_point_1 = r_point1     # 右边点找到
            break
        elif r_point1 == 126:        # 如果找到 126 都没找到
            right_point_1 = 127     # 强制右左边点为 127
            break
```

```
    # 计算中点
    mid_point_1 = (left_point_1 + right_point_1) / 2
    # 以这次中点作为下次的上次中点
    last_mid_point_1 = int(mid_point_1)
    # 返回当前中点
    return mid_point_1
left_point_2 = False
right_point_2 = False
last_mid_point_2 = 0  # 上次中点，应在函数外部初始化
mid_point_2 = 64        # 当前中点，应在函数外部初始化
def get_ccd2_mid_point(bin_ccd):
    global last_mid_point_2, mid_point_2, left_point_2, right_point_2, n, N

    # 搜索左边点，以上次中点作为这次的起搜点
    for l_point2 in range(last_mid_point_2, 1, -1):
        if bin_ccd[l_point2 - 1] == 0 and bin_ccd[l_point2] == 1:    # 判断是否与
左边点一致
            left_point_2 = l_point2      # 左边点找到
            break
        elif l_point2 == 1:      # 如果找到 1 都没找到
            left_point_2 = 0     # 强制令左边点为 0
            break

    # 搜索右边点，以上次中点作为这次的起搜点
    for r_point2 in range(last_mid_point_2, 126):  # 注意这里应该是 128，因为索
引是从 0 开始的
        if bin_ccd[r_point2] == 1 and bin_ccd[r_point2 + 1] == 0:    # 判断是否与
右边点一致
            right_point_2 = r_point2     # 右边点找到
            break
        elif r_point2 == 126:       # 如果找到 126 都没找到
            right_point_2 = 127     # 强制右左边点为 127
            break

    # ccd2 左圆环补线
    if (n == 2):
        left_point_2 = right_point_2 - 55
    if (N == 2):
        right_point_2 = left_point_2 + 68
    if (n == 4):
```

```
            left_point_2 = right_point_2 - 55


        # ccd2 右圆环补线
        if (m == 2):
            right_point_2 = left_point_2 + 55
        if (M == 2):
            left_point_2 = right_point_2 - 68
        if (m == 4):
            right_point_2 = left_point_2 + 55


        # 计算中点
        mid_point_2 = (left_point_2 + right_point_2) / 2
        # 以这次中点作为下次的上次中点
        last_mid_point_2 = int(mid_point_2)
        # 返回当前中点
        return mid_point_2


# 偏差计算
def get_offset(mid_point):
    # 计算误差并返回结果
    offset = 64.0 - mid_point
    return offset
# 舵机位置式 PID
class turn_PID:
    def __init__(self, kp_turn, kd_turn):
        self.kp_turn = kp_turn      # kp 系数
        self.kd_turn = kd_turn      # kd 系数
        self.err = 0                # 误差
        self.last_err = 0           # 上次误差
        self.out = 0                # 输出值
    def Direction_pid(self, err):
        #比例项
        P = err * self.kp_turn
        #微分项
        D = self.kd_turn * (err - self.last_err)

        # 使用公式计算转角值
```

```
        self.out = P + D
        #更新 err
        self.last_err = err

        # 限幅在 13.0°-10.5°之间
        if self.out >= 10.5:
            self.out = 10.5
        elif self.out <= -13.0:
            self.out = -13.0

        return self.out
# 电机增量式 PID
class motor_PID:
    def __init__(self, kp_motor, ki_motor, kd_motor):
        self.kp_motor = kp_motor    # kp 系数
        self.ki_motor = ki_motor    # ki 系数
        self.kd_motor = kd_motor    # ki 系数
        self.aim_speed = 0          # 期望速度
        self.speed = 0              # 实际速度
        self.speed_err = 0          # 误差
        self.last_speed = 0         # 上次误差
        self.last_speed2 = 0
        self.out = 0                # 输出值
    def motor_control(self, aim_speed, speed):
        self.speed_err = aim_speed - speed    # 计算误差

        # 比例项
        P = self.kp_motor * (self.speed_err - self.last_speed)
        # 积分项
        I = self.ki_motor * self.speed_err
        # 微分项
        D = self.kd_motor * (self.speed_err - 2 * self.last_speed + self.last_speed2)

        # 进行增量式 PID 运算
        self.out += P + I

        # 保存上次误差
        self.last_speed = self.speed_err
        self.last_speed2 = self.last_speed
```

```
        # 限幅占空比
        if self.out > 3700:
            self.out = 3700
        elif self.out < -3700:
            self.out = -3700

        return int(self.out)

# 全局变量赛道类型和阶段
road_type = {
    '左圆环 1' : '左圆环 1',
    '左圆环 2' : '左圆环 2',
    '右圆环 1' : '右圆环 1',
    '右圆环 2' : '右圆环 2',
    '斑马线' : '斑马线',
}
flag = None
zebra = 0
# 元素识别
def search_element():
    global flag, zebra
    '''
    # 斑马线
    for i in range(54, len(ccd_data2) - 54):
        if (abs(ccd_data2[i] - ccd_data2[i + 2]) >= 50):
            zebra += 1
    if zebra >= 8:
        flag = road_type['斑马线']
    else:
        zebra = 0
    '''



    return flag

# UI 总显示选择总程序
def menu():
    if (main_menu_item == 1):          # 菜单指针 1
        main_menu()                    # 主菜单
    if (main_menu_item == 2):          # 菜单指针 2
```

```
        Sec_Menu_01()                   # 进入发车相关选择
    if (main_menu_item == 3):           # 菜单指针 3
        Sec_Menu_02()                   # 速度调控选择
    if (main_menu_item == 4):           # 菜单指针 4
        Sec_Menu_03()                   # 元素选择
    if (main_menu_item == 5):           # 菜单指针 5
        Sec_Menu_04()                   # 舵机 PD 系数
    if (main_menu_item == 6):           # 菜单指针 6
        Sec_Menu_05()                   # 电机 PI 系数
    if (main_menu_item == 7):           # 菜单指针 7
        Sec_Menu_06()                   # ccd 图像显示
    if (main_menu_item == 8):           # 菜单指针 8
        Sec_Menu_07()                   # 常用参数
    if (main_menu_item == 9):           # 菜单指针 9
        Sec_Menu_08()                   # 屏幕一键关闭
    if (main_menu_item == 10):          # 菜单指针 10
        Sec_Menu_09()                   # 屏幕一键保存
# 主菜单显示
def main_menu():
    global sec_menu_item, main_menu_item,car_run
    lcd.str24(60, 0, "main_menu", 0x07E0)       # 主菜单标题
    lcd.str16(16, 30, "car_go", 0xFFFF)         # 进入发车相关选择
    lcd.str16(16, 46, "speed", 0xFFFF)          # 速度调控选择
    lcd.str16(16, 62, "element", 0xFFFF)        # 元素选择
    lcd.str16(16, 78, "turn_pd", 0xFFFF)        # 舵机 PD 系数
    lcd.str16(16, 94, "motor_pi", 0xFFFF)       # 电机 PI 系数
    lcd.str16(16, 110, "ccd_image", 0xFFFF)     # ccd 图像显示
    lcd.str16(16, 126, "parameter", 0xFFFF)     # 常用参数
    lcd.str16(16, 142, "screen_off", 0xFFFF)    # 屏幕一键关闭
    lcd.str16(16, 158, "save_para", 0xFFFF)     # 屏幕一键保存

    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标 “>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
```

```
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标“*”
        lcd.clear(0x0000)       # 清屏

    # 向上
    if key_data[0]:
        sec_menu_item -= 1     # 指针减少
        key.clear(1)

    # 向下
    if key_data[1]:
        sec_menu_item += 1     # 指针增加
        key.clear(2)

    # 返回一级菜单
    if(sec_menu_item == 2 and Go.value() == 0):
        car_run += 1
    if(sec_menu_item == 3 and Go.value() == 0):
        main_menu_item = 3
        sec_menu_item = 2
        lcd.clear(0x0000)       # 清屏
    if(sec_menu_item == 4 and Go.value() == 0):
        main_menu_item = 4
        sec_menu_item = 2
        lcd.clear(0x0000)       # 清屏
    if(sec_menu_item == 5 and Go.value() == 0):
        main_menu_item = 5
        sec_menu_item = 2
        lcd.clear(0x0000)       # 清屏
    if(sec_menu_item == 6 and Go.value() == 0):
        main_menu_item = 6
        sec_menu_item = 2
        lcd.clear(0x0000)       # 清屏
    if(sec_menu_item == 7 and Go.value() == 0):
        main_menu_item = 7
```

```
            sec_menu_item = 2
            lcd.clear(0x0000)       # 清屏
        if(sec_menu_item == 8 and Go.value() == 0):
            main_menu_item = 8
            sec_menu_item = 2
            lcd.clear(0x0000)       # 清屏

        gc.collect()
# 发车相关选择
def Sec_Menu_01():
    lcd.str24(60, 0, "car_go", 0x07E0)          # 二级菜单标题
    lcd.str16(16, 30, "return", 0xFFFF)          # 返回主菜单

    lcd.str16(16, 46, "GO", 0xFFFF)               # 发车
    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标“>”
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标“*”

    gc.collect()
# 速度调控选择
def Sec_Menu_02():
    global sec_menu_item, aim_speed_l, aim_speed_r, main_menu_item
    lcd.str24(60, 0, "speed", 0x07E0)           # 二级菜单标题
    lcd.str16(16, 96, "return", 0xFFFF)          # 返回主菜单

    lcd.str16(16, 30, "aim_speed_l = {}".format(aim_speed_l), 0xFFFF)     # 左轮目
标速度
    lcd.str16(16, 62, "aim_speed_r = {}".format(aim_speed_r), 0xFFFF)     # 右轮
目标速度

    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标“>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
```

```
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标“*”
        lcd.clear(0x0000)       # 清屏

    # 向上
    if key_data[0]:
        sec_menu_item -= 2
        key.clear(1)

    # 向下
    if key_data[1]:
        sec_menu_item += 2
        key.clear(2)

    # 限幅指针
    if (sec_menu_item < 2):
        sec_menu_item = 2
    if (sec_menu_item > 6):
        sec_menu_item = 6

    # 按键调参
    if (sec_menu_item == 4):
        if key_data[3]:
            aim_speed_l += 5
        if key_data[2]:
            aim_speed_l -= 5
    if (sec_menu_item == 6):
        if key_data[3]:
            aim_speed_r += 5
        if key_data[2]:
            aim_speed_r -= 5

    # 返回一级菜单
    if(sec_menu_item == 6 and Go.value() == 0):
        main_menu_item = 1
```

```
            sec_menu_item = 2
            lcd.clear(0x0000)       # 清屏

    gc.collect()


# 元素选择
def Sec_Menu_03():
    lcd.str24(60, 0, "element", 0x07E0)       # 二级菜单标题
    lcd.str16(16, 30, "return", 0xFFFF)       # 返回主菜单
    if (flag is '左圆环 1'):
        lcd.str16(16, 110, 'left_round1', 0xFFFF)     # 左圆环 1
    elif (flag is '左圆环 2'):
        lcd.str16(16, 110, 'left_round1', 0xFFFF)     # 左圆环 2
    elif (flag is '右圆环 1'):
        lcd.str16(16, 110, 'right_round1', 0xFFFF)    # 右圆环 1
    elif (flag is '右圆环 2'):
        lcd.str16(16, 126, 'right_round2', 0xFFFF)    # 右圆环 2
    elif (flag is '斑马线'):
        lcd.str16(16, 142, 'zebra', 0xFFFF)           # 斑马线
    gc.collect()

# 舵机 PD 系数
def Sec_Menu_04():
    global sec_menu_item, main_menu_item
    lcd.str24(60, 0, "turn_pd", 0x07E0)       # 二级菜单标题
    lcd.str16(16, 222, "return", 0xFFFF)       # 返回主菜单
    lcd.str16(16, 62, "ccd1_kp_turn = {}".format(kp_turn[0]), 0xFFFF)     # 舵机
kp_ccd1
    lcd.str16(16, 46, "ccd1 p + / -   0.01", 0x001F)
    lcd.str16(16, 110, "ccd1_kd_turn = {}".format(kd_turn[0]), 0xFFFF)     # 舵机
kd_ccd1
    lcd.str16(16, 94, "ccd1 d + / -   0.01", 0x001F)
    lcd.str16(16, 158, "ccd2_kp_turn = {}".format(kp_turn[1]), 0xFFFF)     # 舵机
kp_ccd2
    lcd.str16(16, 142, "ccd2 p + / - 0.01", 0x001F)
    lcd.str16(16, 206, "ccd2_kd_turn = {}".format(kd_turn[1]), 0xFFFF)     # 舵机
kd_ccd2
    lcd.str16(16, 190, "ccd2 d + / - 0.01", 0x001F)
```

```
    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标“>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标“*”
        lcd.clear(0x0000)       # 清屏

    # 向上
    if key_data[0]:
        sec_menu_item -= 1
        key.clear(1)

    # 向下
    if key_data[1]:
        sec_menu_item += 1
        key.clear(2)

    # 指针限幅
    if (sec_menu_item < 2):
        sec_menu_item = 2
    if (sec_menu_item > 14):
        sec_menu_item = 14

    # 按键调参
    if (sec_menu_item == 3):
        if key_data[3]:
            kp_turn[0] += 0.01
        if key_data[2]:
            kp_turn[0] -= 0.01
```

```
    if (sec_menu_item == 4):
        if key_data[3]:
            kp_turn[0] += 0.1
        if key_data[2]:
            kp_turn[0] -= 0.1
    if (sec_menu_item == 6):
        if key_data[3]:
            kd_turn[0] += 0.01
        if key_data[2]:
            kd_turn[0] -= 0.01
    if (sec_menu_item == 7):
        if key_data[3]:
            kd_turn[0] += 0.1
        if key_data[2]:
            kd_turn[0] -= 0.1
    if (sec_menu_item == 9):
        if key_data[3]:
            kp_turn[1] += 0.01
        if key_data[2]:
            kp_turn[1] -= 0.01
    if (sec_menu_item == 10):
        if key_data[3]:
            kp_turn[1] += 0.1
        if key_data[2]:
            kp_turn[1] -= 0.1
    if (sec_menu_item == 12):
        if key_data[3]:
            kd_turn[1] += 0.01
        if key_data[2]:
            kd_turn[1] -= 0.01
    if (sec_menu_item == 13):
        if key_data[3]:
            kd_turn[1] += 0.1
        if key_data[2]:
            kd_turn[1] -= 0.1

    # 返回一级菜单
    if(sec_menu_item == 14 and Go.value() == 0):
        main_menu_item = 1
        sec_menu_item = 2
```

```
        lcd.clear(0x0000)       # 清屏

    gc.collect()

# 电机 PI 系数
def Sec_Menu_05():
    global sec_menu_item, kp_motor, ki_motor, kd_motor, main_menu_item
    lcd.str24(60, 0, "motor_pi", 0x07E0)        # 二级菜单标题
    lcd.str16(16, 158, "return", 0xFFFF)         # 返回主菜单
    lcd.str16(16, 62, "kp_motor = {}".format(kp_motor), 0xFFFF)     # 电机 kp
    lcd.str16(16, 46, "p + / -   0.01", 0x001F)
    lcd.str16(16, 110, "ki_motor = {}".format(ki_motor), 0xFFFF)     # 电机 ki
    lcd.str16(16, 94, "i + / -   0.01", 0x001F)
    lcd.str16(16, 142, "kd_motor = {}".format(kd_motor), 0xFFFF)     # 电机 ki
    lcd.str16(16, 126, "i + / -   0.01", 0x001F)

    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标 “>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标 “*”
        lcd.clear(0x0000)       # 清屏

    # 向上
    if key_data[0]:
        sec_menu_item -= 1
        key.clear(1)

    # 向下
```

```
    if key_data[1]:
        sec_menu_item += 1
        key.clear(2)

    # 指针限幅
    if (sec_menu_item < 2):
        sec_menu_item = 2
    if (sec_menu_item > 10):
        sec_menu_item = 10

    # 按键调参
    if (sec_menu_item == 3):
        if key_data[3]:
            kp_motor += 0.01
        if key_data[2]:
            kp_motor -= 0.01
    if (sec_menu_item == 4):
        if key_data[3]:
            kp_motor += 0.1
        if key_data[2]:
            kp_motor -= 0.1
    if (sec_menu_item == 6):
        if key_data[3]:
            ki_motor += 0.01
        if key_data[2]:
            ki_motor -= 0.01
    if (sec_menu_item == 7):
        if key_data[3]:
            ki_motor += 0.1
        if key_data[2]:
            ki_motor -= 0.1

    # 返回一级菜单
    if(sec_menu_item == 10 and Go.value() == 0):
        main_menu_item = 1
        sec_menu_item = 2
        lcd.clear(0x0000)       # 清屏

    gc.collect()
```

```
# ccd 图像显示
def Sec_Menu_06():
    lcd.str24(60, 0, "ccd_image", 0x07E0)       # 二级菜单标题
    lcd.str16(16, 30, "return", 0xFFFF)         # 返回主菜单

    lcd.wave(0,   64, 128, 64, ccd_data1)       # ccd1 图像
    lcd.wave(0, 128, 128, 64, ccd_data2)        # ccd2 图像
    lcd.line(64, 64, 64, 192, color = 0x001F, thick = 1)     # 实际中线

    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标 “>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标 “*”
        lcd.clear(0x0000)       # 清屏

    gc.collect()


# 常用参数
def Sec_Menu_07():
    global sec_menu_item
    lcd.str24(60, 0, "parameter", 0x07E0)     # 二级菜单标题
    lcd.str16(16, 30, "return", 0xFFFF)         # 返回主菜单

    lcd.str16(16, 46, "tof = {:<4d}".format(tof_data),
0xFFFF)                                 # TOF 数据
    lcd.str16(16, 62, "out_l = {:<4d}".format(out_l),
0xFFFF)                                  # 左环 pid 输出
```

```
    lcd.str16(16, 78, "out_r = {:<4d}".format(out_r),
0xFFFF)                                 # 右环 pid 输出
    lcd.str16(16, 94, "encl = {:<4d}".format(encl_data),
0xFFFF)                                  # 左编码器值
    lcd.str16(16, 110, "encr = {:<4d}".format(encr_data),
0xFFFF)                                 # 右编码器值
    lcd.str16(16, 126, "Mid_point1 = {:<.2f}".format(Mid_point1),
0xFFFF)                     # ccd1 中点
    lcd.str16(16, 142, "Mid_point2 = {:<.2f}".format(Mid_point2),
0xFFFF)                     # ccd2 中点
    lcd.str16(16, 158, "error1 = {:<.2f}".format(error1),
0xFFFF)                                 # ccd1 误差
    lcd.str16(16, 174, "error2 = {:<.2f}".format(error2),
0xFFFF)                                 # ccd2 误差
    lcd.str16(16, 190, "left_point_1 = {:<3d}".format(left_point_1),
0xFFFF)                 # 上摄像头左边点
    lcd.str16(16, 206, "right_point_1 = {:<3d}".format(right_point_1),
0xFFFF)               # 上摄像头右边点
    lcd.str16(16, 222, "left_point_2 = {:<3d}".format(left_point_2),
0xFFFF)                 # 下摄像头左边点
    lcd.str16(16, 238, "right_point_2 = {:<3d}".format(right_point_2),
0xFFFF)               # 下摄像头右边点
    lcd.str16(16, 254, "width_1 = {:<3d}".format(right_point_1 - left_point_1),
0xFFFF)    # ccd1 计算赛道宽度
    lcd.str16(16, 270, "width_2 = {:<3d}".format(right_point_2 - left_point_2),
0xFFFF)    # ccd2 计算赛道宽度
    if (data_change_flag == 0):
        lcd.str12(0, sec_menu_item * 16, ">", 0xF800)    # 光标 “>”
        if key_data[0]:
            lcd.clear(0x0000)       # 清屏
            key.clear(1)
        if key_data[1]:
            lcd.clear(0x0000)       # 清屏
            key.clear(2)
        if key_data[2]:
            lcd.clear(0x0000)       # 清屏
            key.clear(3)
        if key_data[3]:
            lcd.clear(0x0000)       # 清屏
            key.clear(4)
```

```
    elif (data_change_flag == 1):
        lcd.str12(0, sec_menu_item * 16, "*", 0xF800)    # 光标 “*”
        lcd.clear(0x0000)       # 清屏

    gc.collect()


# 屏幕一键关闭
def Sec_Menu_08():
    lcd.clear(0x0000)       # 清屏
    Return

# 屏幕一键保存
def Sec_Menu_09():
    write_flash()           # 写入缓冲区

    buzzer.value(1)         # 蜂鸣器开
    lcd.clear(0xF800)       # 清屏
    time.sleep_ms(100)      # 延时
    main_menu_item = 1      # 返回一级菜单
    buzzer.value(0)         # 蜂鸣器关

# 写入缓冲区
def write_flash():
    os.chdir("/flash")      # 切换到 /flash 目录
    try:
        # 通过 try 尝试打开文件 因为 r+ 读写模式不会新建文件
        user_file = io.open("user_data.txt", "r+")
    except:
        # 如果打开失败证明没有这个文件 所以使用 w+ 读写模式新建文件
        user_file = io.open("user_data.txt", "w+")
    # 将指针移动到文件头 0 偏移的位置
    user_file.seek(0, 0)
    # 使用 write 方法写入数据到缓冲区

    user_file.write("%.4f\n"%(kp_turn))
    user_file.write("%.4f\n"%(kd_turn))
    user_file.write("%.4f\n"%(kp_motor))
    user_file.write("%.4f\n"%(ki_motor))
    user_file.write("%d\n"%(aim_speed_l))
```

```
    user_file.write("%d\n"%(aim_speed_r))

    # 将缓冲区数据写入到文件 清空缓冲区 相当于保存指令
    user_file.flush()
    # 将指针重新移动到文件头
    user_file.seek(0, 0)
    # 读取三行数据 到临时变量 分别强制转换回各自类型
    data1 = float(user_file.readline())
    data2 = float(user_file.readline())
    data3 = float(user_file.readline())
    data4 = float(user_file.readline())
    data5 = int(user_file.readline())
    data6 = int(user_file.readline())
```

```
    # 最后将文件关闭即可
    user_file.close()


speed_pid_l = motor_PID(kp_motor = 10.0, ki_motor = 1, kd_motor = 0)     # 左电机 PID 初始化
speed_pid_r = motor_PID(kp_motor = 10.0, ki_motor = 1, kd_motor = 0)     # 右电机 PID 初始化
servo_pid = turn_PID(kp_turn = 0.45, kd_turn = 0.5)                      # 舵机 PID 初始化
T = 500       # 总路程
while True:
    # 计算路程
    distance_T += (encl_data + encr_data) / 2 / 1024 * 30 / 50 * 0.05 * 3.1415926
    if(distance_T >= 2.5) :
            # 获取电机对应占空比
        motor_l.duty(0)
        motor_r.duty(0)

    else:
            # 获取电机对应占空比
        motor_l.duty(out_r)
        motor_r.duty(out_l)

    # 拨码开关关中断
    if end_switch.value() == 0:
```

```
        pit1.stop()     # pit1 关闭
        pit2.stop()     # pit2 关闭
        pit3.stop()     # pit3 关闭
        break            # 跳出判断

    # 上位机显示
    #wireless.send_oscilloscope(output_encl, output_encr, aim_speed_l,
aim_speed_r)

    # 负压 92.5%起转
    bldc1.highlevel_us(1925)
    bldc2.highlevel_us(1925)

    # 速度控制
    if (error2 >= 10.0):
        aim_speed_l = T #- (servo_pid.Direction_pid(error2) * 4.5)    # 差速：目标
速度 +/- (舵机输出 * 差速系数)
        aim_speed_r = T #+ (servo_pid.Direction_pid(error2) * 6.5)    # 差速：目标
速度 +/- (舵机输出 * 差速系数)
    if (-10.0 <= error2 <= 10.0):
        aim_speed_l = T       # 无差速状态
        aim_speed_r = T       # 无差速状态
    if (error2 <= -10.0):
        aim_speed_l = T #- (servo_pid.Direction_pid(error2) * 4.5)    # 差速：目标
速度 +/- (舵机输出 * 差速系数)
        aim_speed_r = T #+ (servo_pid.Direction_pid(error2) * 6.5)    # 差速：目标
速度 +/- (舵机输出 * 差速系数)


    # 电机 PID 计算
    out_l = speed_pid_l.motor_control(aim_speed = aim_speed_l, speed =
output_encl)
    out_r = speed_pid_r.motor_control(aim_speed = aim_speed_r, speed =
output_encr)

    # 3ms 中断标志位
    if(ticker_flag_3ms):
        #menu()                              # 菜单显示
        lcd.wave(0,   64, 128, 64, ccd_data1, max = 255)
        lcd.wave(0, 128, 128, 64, ccd_data2, max = 255)
```

```
        ccd_data1 = ccd.get(0)              # 读取 ccd1 的数据
        ccd_data2 = ccd.get(1)              # 读取 ccd2 的数据
        imu_data = imu.get()                # 读取陀螺仪的数据

        # 计算动态阈值
        threshold1 = ccd_get_threshold(ccd_data1)
        threshold2 = ccd_get_threshold(ccd_data2)

        # 对 ccd 数据二值化处理
        image_value1 = ccd_image_value(ccd_data1, threshold1)
        image_value2 = ccd_image_value(ccd_data2, threshold2)

        # 进行中点计算
        Mid_point1 = get_ccd1_mid_point(image_value1)
        Mid_point2 = get_ccd2_mid_point(image_value2)

        # 进行偏差计算
        error1 = get_offset(Mid_point1)
        error2 = get_offset(Mid_point2)

        # 元素识别
        search_element()

        gc.collect()
        ticker_flag_3ms = False


    # 1s 中断标志位
    if(ticker_flag_1000ms):
        ccd_data1 = [0] * 128           # 清空 ccd1 原始数据
        ccd_data2 = [0] * 128           # 清空 ccd2 原始数据
        image_value1 = [0] * 128        # 清空 ccd1 二值化数据
        image_value2 = [0] * 128        # 清空 ccd2 二值化数据
        ticker_flag_1000ms = False


    # 10ms 中断标志位
    if (ticker_flag):
        encl_data = encoder_l.get()     # 读取左编码器的数据
        encr_data = encoder_r.get()     # 读取右编码器的数据
```

```
        key_data = key.get()                # 读取按键的数据

        # 舵机 PD 控制
        out = servo_pid.Direction_pid(error2) + angle
        # 获取舵机角度对应占空比
        duty = int(duty_angle(pwm_servo_hz, out))
        pwm_servo.duty_u16(duty)

        # 关中断标志位
        gc.collect()
        ticker_flag = False
    gc.collect()
```