第十七届全国大学生
智能汽车竞赛

技 术 报 告

学　　校：中国计量大学

队伍名称：仰仪 9 队

参赛队员：赵宇龙 李育婵 杨炅坤 董明泽 冼国森

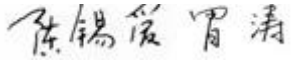
带队教师：　陈锡爱、曾涛

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：

带队教师签名：



日　　期：2022 年 8 月 19 日

目录

摘要

本文主要介绍了基于 RT-Thread 操作系统的完全模型组智能车的相关硬件设计方案以及车辆控制方案，对车辆控制系统的介绍包括车模机械结构的设计、电路模块的设计、图像处理算法、控制算法、AI 模型部署以及上下位机配合的方法等。设计难点在于上位机与下位机相配合，本项目以百度 EdgeBoard 计算卡（ FZ3B 赛事定制版）为上位机，用于图像处理和 AI 模型的部署，其中 AI 模型使用飞桨 PaddlePaddle 框架搭建和训练完成；以 TC264 为下位机核心控制器，用于实现车模运动控制，上位机与下位机通过串口进行通讯，使用编码器获取小车的速度，陀螺仪获取小车姿态。下位机程序基于 RT-Thread 操作系统，ADS 作为集成编译环境，使用无限网卡、串口、oled、拨码开关和按键作为辅助调试手段。经过大量的软硬件测试，较好实现了智能车在完成循迹任务的同时进行基于飞桨深度学习模型的目标检测算法和相关任务的完成。

关键词：Infineon 智能车 RT-Thread 操作系统 线程 定时器 TC264 芯片
百度 Paddle 框架 PID 控制 系统辨识

第一章 引言

1.1 全国大学生智能车竞赛介绍

全国大学生智能汽车竞赛是一项以“立足培养、重在参与、鼓励探索、追求卓越”为指导思想，面向全国大学生开展的具有探索性的工程实践活动。它以设计制作在特定赛道上能自主行驶且具有优越性能的智能模型汽车这类复杂工程问题为任务，鼓励大学生组成团队，综合运用多学科知识，提出、分析、设计、开发并研究智能汽车的机械结构、电子线路、运动控制和开发与调试工具等相关问题，激发大学生从事工程技术开发和科学研究探索的兴趣和潜能，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。

加强大学生实践、创新能力和团队精神的培养，是当前高等教育教学改革的重要内容之一。进一步促进高等学校加强对学生创新精神、协作精神和工程实践能力的培养，提高学生解决实际问题的能力，充分利用面向大学生的群众性科技活动，为优秀人才的脱颖而出创造条件，不断提高人才培养质量。该竞赛融合科学性、趣味性和观赏性为一体，是以迅猛发展、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。

1.2 全国大学生智能车竞赛完全模型组制作情况

完全模型组比赛赛道以室内循环赛道为基础，赛道材质，赛道规格均保持一致。在导引方式上完全保留室内循环赛道的导引方式，并在此基础上添加完全模型组任务导引标志和锥桶，引导车模完成完全模型组赛道任务。

完全模型组的基本比赛任务为：

选手制作的车模完成从车库出发沿着赛道运行两周。车模需要分别通过道路设置的各种元素，识别道路中心的标志完成特殊路段通行。比赛时间从车模

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

本次比赛采用大赛组委会统一指定的 I 型车模，以百度 EdgeBoard 计算卡（ FZ3B 赛事定制版）为上位机，用于赛道检测识别，要求模型算法必须使用百度 Paddle 框架搭建，即必须使用百度深度学习框架的人工智能算法实现；以 TC264 为下位机核心控制器，用于实现车模运动控制，上位机与下位机通过串口进行通讯，要求智能车识别白色赛道和车库等完成循迹任务的同时进行模型训练以完成模型任务，以最快速度完成整个比赛，智能车采用 120 帧彩色 uvc 摄像头对赛道信息进行检测，采用逆透视方案和边线平移算法得到赛道中线；

通过编码器检测智能车的实时速度，利用陀螺仪辅助检测坡道；使用增量式 PID 控制算法调节电机的转速和使用位置式 PID 控制算法调节舵机的打角，实现智能车在运动过程中速度和方向的闭环控制；为了提高模型车的速度和稳定性，使用无线网卡、按键、OLED 模块等调试工具，进行了大量硬件与软件测试。实验结果表明，该系统设计方案确实可行。

经过一年的准备，我们在小车机械结构设计、3D 打印设计、硬件电路设计、软件算法设计等不断改进，灵活应用了 RT-Thread 系统的多线程并发、软定时器、线程间同步——信号量、线程间通信——邮箱、时间片轮转等特性，制作出了结构合理、系统稳定、可以顺利完成循迹和模型比赛任务的完全模型组小车，在第十七届全国大学生智能汽车比赛浙江省选拔赛中获得第三名的成绩。

第二章 智能车系统总体设计

智能车控制系统的每一个参数，都对整个智能车系统的稳定运行起着至关重要的作用,其中机械结构和控制电路的设计对车的稳定性有很大影响。通过对车辆整体布局的合理规划，寻找合适的重心，使其左右平衡，保证车辆在高速状态下稳定运行。同时，既要保证车辆轮胎可靠地抓牢地面，又要保证舵机、电机有较快的响应。在满足各个要求的情况下，尽量使电路设计简洁，PCB 布局美观。

2.1 智能车整体布局

我们组采用的是 I 车模，对称性好，整车尺寸为：316*190*360mm(含摄像头碳纤杆)，摄像头为 120 帧彩色 uvc 摄像头，驱动电机为 RS-555 微型直流电机，额定电压 12V，额定转速 12000rpm；舵机为 CS-3120 数字舵机，反应速度快，空载转速在 5.0V 时可以达到 0.16 秒/60 度，空载转速在 6.8V 时可以达到 0.14 秒/60 度；超大扭矩，5.0V 供电时额定扭矩可以达到 15kg • cm，6.8V 供电时额定扭矩可以达到 17kg • cm。智能车的外形大致如图 1.2。

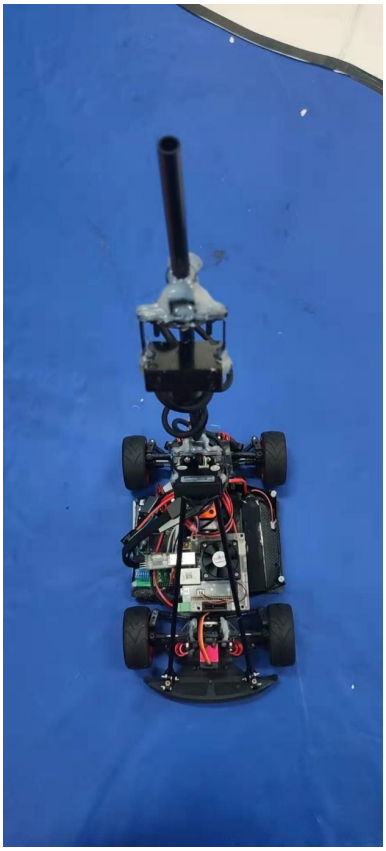


图 1.2 智能车俯视图

车模布局特点：

(1) 车模底盘降低，用来降低重心。

(2) 舵机采用竖直姿态，方便控制车体。

(3) 对前轮倾角进行调节，主销后倾，主销内倾，保证车直线行驶的稳定性。

(4) 调整各器件位置使车体左右平衡。

2.2 智能车机械设计

2.2.1 摄像头的安装

作为最主要的道路信息采集传感器，摄像头的安装方式是否稳固，是否合理，是其能够得到清晰稳定的图像的关键。经过多次尝试调整，利用三角形稳定原理，最终使用一根竖直碳素杆，两根碳素杆辅助稳定，以达到较为稳定的

效果。使用防松螺母将摄像头支撑杆底部固定在车模转向的轴线上，保证杆子与车身保持垂直。同时，碳素杆固定在靠近车头位置，使车模转向时不存在水平分量，保证图像的稳定性。

2.2.2 车模前轮倾角的调整

在调试过程中，我们发现前轮对于车模的运行影响较大，并且由于前轮轴和车轮之间的间隙较大，对车高速时转向中心的影响较大，会引起高速转向下模型车的转向不足。为了尽可能降低转向舵机负载，我们对前轮的安装角度，即前轮定位进行了调整。

前轮定位的作用是保障汽车直线行驶的稳定性，转向轻便和减少轮胎的磨损。前轮是转向轮，它的安装位置由主销内倾、主销后倾、前轮外倾和前轮前束等 4 种可调参数决定，实现转向轮、主销和前轴等三者在车架上的位置关系。

在实际调试中，我们发现适当增大内倾角可以增大转弯时车轮和地面的接触面积，从而增大车了地面的摩擦程度，使车转向更灵活，减小因摩擦不够而引起的转向不足的情况。并且我们智能视觉组设计的车模相对于往届车模来说重心更高，更靠前，所以我们对于主销后倾做了一些调整。

2.2.3 底盘高度的调整

在保证顺利通过坡道的前提下，底盘尽量降低以降低重心，从整体上降低模型车的重心，可使模型车转弯时更加稳定、高速。

2.2.4 编码器的安装

编码器用来测量小车速度，实现小车转速闭环控制，就必须保证齿轮完全啮合才能使得测量结果准确，同时在实际调试过程中我们还发现，编码器掉齿、卡顿等现象时有发生，为提高编码器测速的稳定性及准确性，用齿轮进行了安装配合，尽量使得传动齿轮轴保持平行，传动部分轻松、流畅，不存在过大噪

音，卡齿，丢数情况。

2.3 智能车硬件设计

2.3.1 电源模块设计

电源模块的稳定是保证各模块正常运行的前提。

硬件供电电源为 12V，由于电路中的不同电路模块所需要的工作电压和电流容量各不相同，主控板设计有 4 种供电电压：

（1）12V 直接由锂电池（带保护板）供电 12V，用于给上位机和驱动板直接供电

（2）5V 使用稳压芯片 LM2596S，输入电压 12V（电源电压），输出电压 5V，用于核心板、编码器、3.3v 稳压芯片 TPS7333 供电。原理图如图所示图 1.3

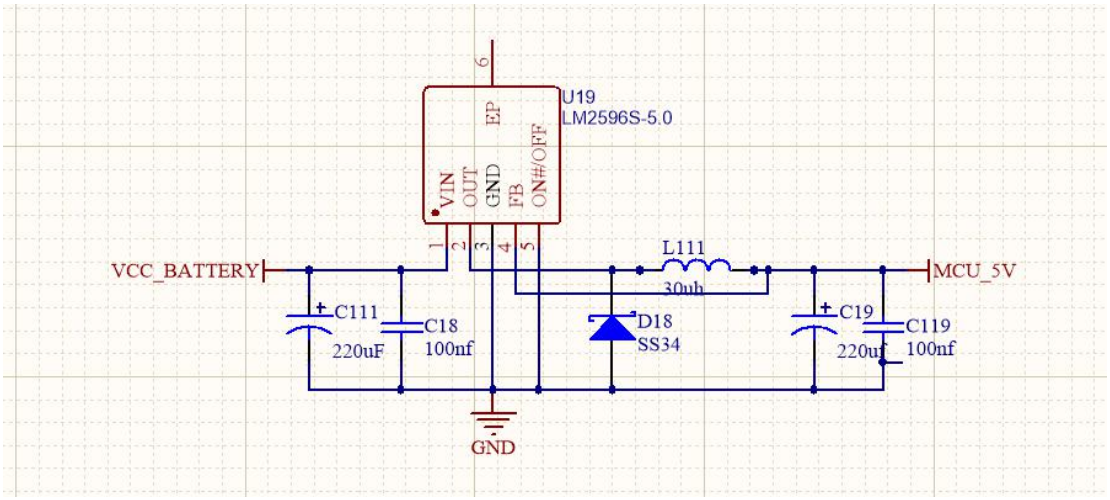


图 1.3 12V 转 5V 稳压电路

（3）3.3V 使用稳压芯片 TPS7333，输入电压 5V，输出电压 3.3V，用于蜂鸣器，串口，陀螺仪，OLED，隔离芯片 EL357，隔离芯片 741v245 供电。原理图如图所示图 1.4

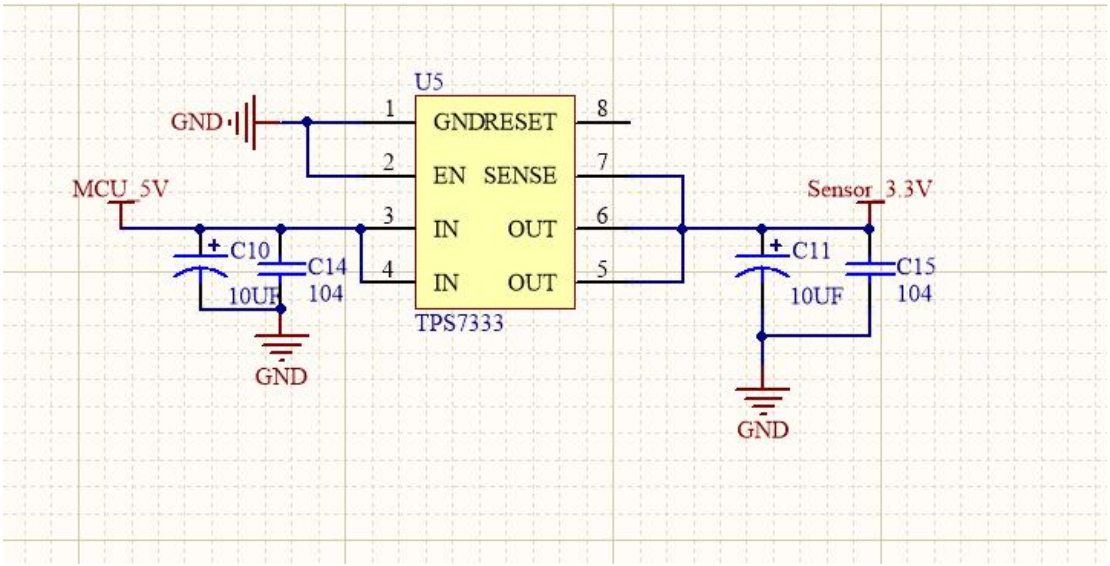


图 1.4 5V 转 3.3V 稳压电路

（4）6V 使用稳压芯片 LT1084，输入电压 12V，输出电压 6V，用于给舵机供电。原理图如图所示图 1.5

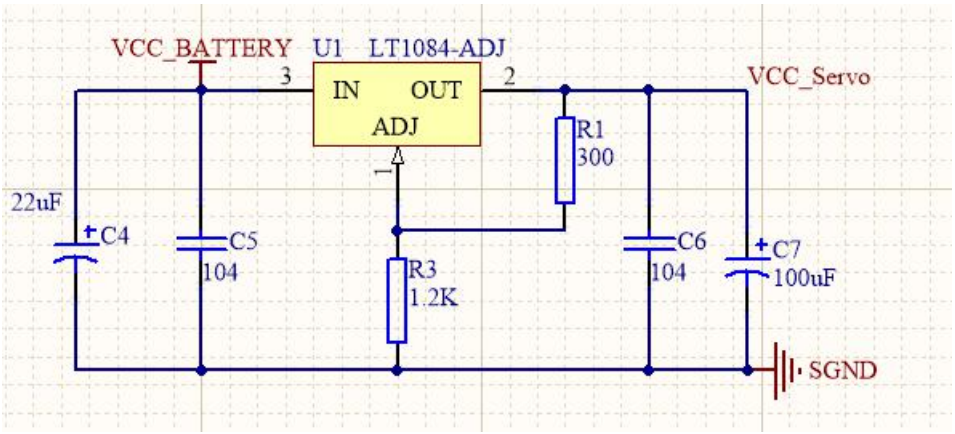


图 1.5 舵机稳压 6V

2.3.1 电机驱动电路

在栅极驱动芯片选择方面，我们选择 IR2104 芯片（用 MC34063 稳压 12V 驱动芯片），IR2104 芯片可以驱动高端和低端两个沟道 MOSFET，提供较大的驱动电流，并有硬件死区，防止同侧桥臂导通。使用两片 IR2104 可以构成一个 MOS 管（IR7843）全桥驱动电路，如图 1.6 所示。

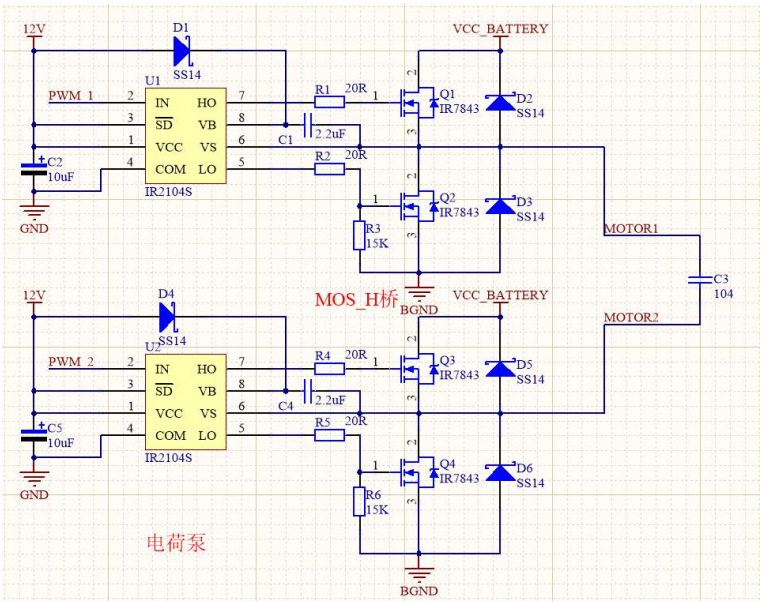


图 1.6 电机驱动电路

2.4 智能车上位机软件设计

2.4.1 基于多进程多线程和状态机的车辆控制方案

为了充分利用 edgeboard 的多核性能，并避免深度学习任务对总体帧率的拖累，我们采用双进程的设计方案，一个进程用于传统图像的寻迹任务，另一个进程用于深度学习模型的预测任务，并且为了避免过多进程间通信的麻烦，我们采用我们采用两个摄像头，每个进程单独读取一个摄像头，省去图像共享的耗时。

其中主进程包含传统图像处理的寻迹任务、状态机的更新任务、主摄像头的读取任务以及串口的读写任务，子进程用于副摄像头的读取任务、副摄像头的传统图像算法任务以及 mobilenetSSD 模型的地标检测任务。而在每个进程中，摄像头图像的采集都单独占用一个线程，避免等待摄像头读取耗时对帧率的拉低。

主进程中的状态机包含三个大状态，即普通寻迹状态、施工区寻迹状态、泛行区寻迹状态，每个大状态下包含若干个小状态位，用于不同寻迹状态下的不同元素、不同阶段的控制。

子进程在循环里运行 mobilenetSSD 模型预测函数，检测地标位置和种类，当地标位置和种类符合要求的时候会更新 Manager 字典里的数据，主进程会在循环里检查这些数据，从而更新主进程的状态位。

2.4.1 基于 OpenCV 和逆透视的智能车图像处理方案

我们凭借上位机操作系统的优势，用 Python 实现了一套基于逆透视的智能车巡线方案。在方案中我们借助 OpenCV 和 skimage 完成我们的图像处理操作，并充分利用 numpy 的广播特性进行高效运算。实测帧率达到 60 帧左右。

我们寻迹用的摄像头是一款全局快门 120 帧的彩色 uvc 摄像头，我们将摄像头对比度调至最高，使白色赛道像素接近 255，然后采用固定阈值二值化。随后我们对图片进行逆透视处理，将图像坐标系对应到笛卡尔坐标系中去，我

们采用 160x120 大小的图片进行处理，其中一像素对应物理世界的一厘米。此外，我们还对二值化图片进行连通域分析，只保留车辆所在赛道的连通域，这样可以减少计算量和避免误判。

接下来进行 canny 边缘检测，提取等高线、端点截断等操作，得到边缘线的数组。接着根据边线与车辆的相对位置筛选出左右边线，依据三点计算角度的原则计算出边线上的角点，同时计算边线的长度、曲率等基本信息，然后依据角点的位置和角度大小，以及边线的一系列信息，判断赛道元素，并作出相应的处理。

我们将边线数组按半径方向平移得到车辆行驶的参考线，考虑到完全模型组的车模是单电机驱动的阿克曼模型车模，我们采用 pure pursuit 算法计算出车辆行驶到预瞄点的转弯半径和舵机打角。

2.4.1 基于骨架提取和连通域分析的任务处理方案

对于施工区的锥桶，我们将红色的锥桶从 hsv 颜色空间中分离出来，得到红色色块，然后将这些色块膨胀到互相连在一起，得到红色长条，对这些长条进行骨架提取，将复杂的连通域变成单像素线条。然后我们对骨架图提取等高线。

在学术界，骨架提取算法的毛刺是一个十分困扰大家的问题，我们利用自创的端点检测算法去毛刺，效果很好，鲁棒性也很强。最终我们得到边线数组，然后将边线平移作为车辆行驶的参考线，再利用 pure pursuit 算法得出舵机打角。

对于泛行区，我们采用精度更高的角点计算方式计算出三叉出入口的 120 度角点，作为出入口处的预瞄点，在三岔内部根据连通域的面积检测障碍物，将障碍物坐标平移一段距离作为预瞄点，同样利用 pure pursuit 算法得出舵机打角。

地标检测基于目标检测模型，识别准确率高，泛化性能好；而任务处理基于传统图像处理，算法稳定，帧率较高。

2.5 智能车下位机算法设计

2.5.1 位置式 PID 和增量式 PID 的区别

（1）位置式 PID 控制的输出与整个过去的状态有关，用到了误差的累加值;而增量式 PID 的输出只与当前拍和前两拍的误差有关，因此位置式 PID 控制的累积误差相对更大;

（2）增量式 PID 控制输出的是控制量增量，并无积分作用，因此该方法适用于执行机构带积分部件的对象，如直流电机等，而位置式 PID 适用于执行机构不带积分部件的对象，如舵机。

（3）由于增量式 PID 输出的是控制量增量，如果计算机出现故障，误动作影响较小，而执行机构本身有记忆功能，可仍保持原位，不会严重影响系统的工作，而位置式的输出直接对应对象的输出，因此对系统影响较大。

增加 P 作用：提高系统响应速度　存在稳态误差 P 过大系统会震荡

增加 I 作用：减少或消除稳态误差　帮助达到目标值　I 过大误差累积很多

增加 D 作用：减少震荡 防止超过目标值

速度闭环控制（采用增量式 PID）就是根据单位时间获取的脉冲数测量电机的速度信息，并与目标值进行比较，得到控制偏差，然后通过对偏差的比例、积分、微分进行控制，使偏差趋向于零的过程。

所谓增量式 PID 是指数字控制器的输出只是控制量的增量△u(k)。增量式 PID 优点在于，由于输出的是增量所以误动作时影响小，必要时可用逻辑判断的方法关掉。另外由于控制增量△u(k)的确定仅与最近 k 次的采样值有关，所以较容易通过加权处理而获得比较好的控制效果。

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

增量式 PID 得出的控制量▲u(k)对应的是近几次位置误差的增量，而不是对应

与实际位置的偏差，没有误差累加

本次增量式 pid 的输出是由本次位置式 pid 的输出减去上次位置式的输出得到的

增量式 PID 的比例项相当于位置式的微分项，积分项其实是位置式的比例项。比例项是误差，积分项是误差的累加，微分项是误差的变化率

2.5.1 闭环控制原理

闭环控制

给定占空比 -> 测转速 -> 比较实际转速和目标转速 -> 重新调整占空比，这样的过程其实就是一个闭环控制，我们发现这个过程形成了一个回环：每次调整的占空比大小都是基于上一次结果得到的。相比开环控制，闭环控制多了信息反馈环节（测电机转速），我们根据反馈信息再做出进一步调整，接着获得调整后的反馈信息，再基于更新过的反馈信息进行新一轮的调控。

增量式 PID 的调参方式

首先把比例项和微分项置 0

然后调整积分项，求起振点

接着调节比例项，抑制振荡

最后调节微分项，削弱超调

位置式 PID 的调参方式

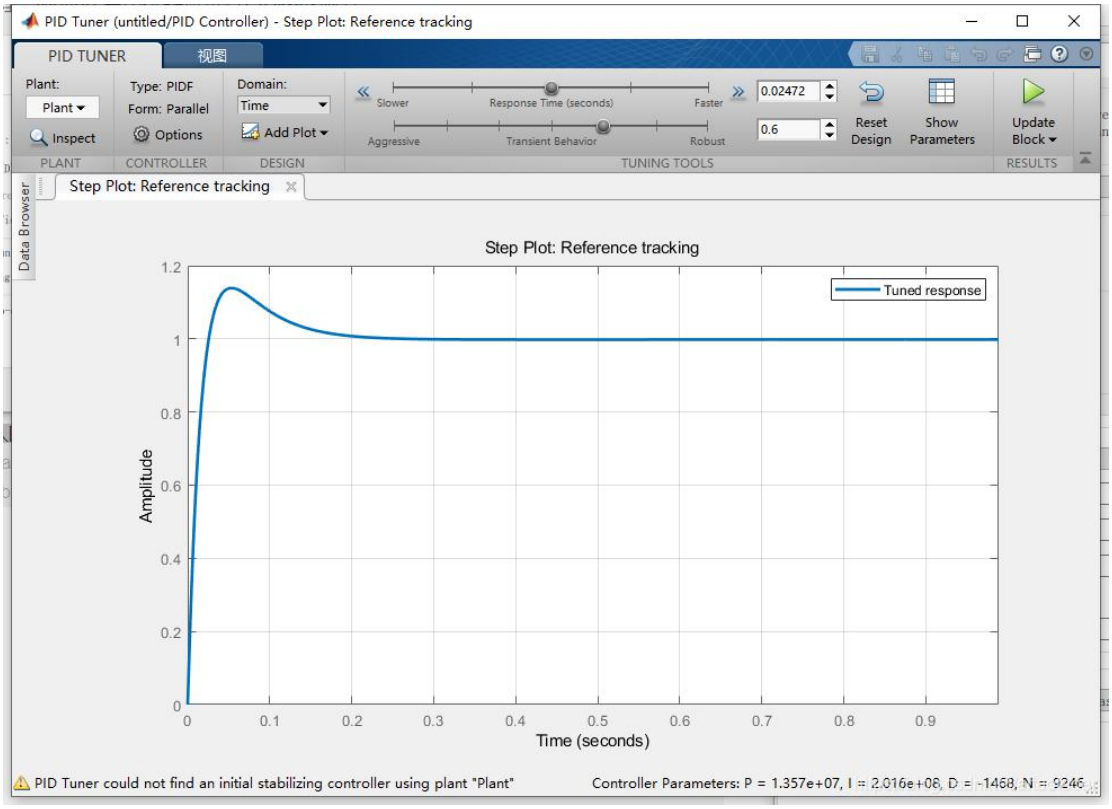
首先把积分项和微分项置 0

然后调整比例项，求起振点

接着调节微分项，抑制振荡

最后调节积分项，削弱超调

通过系统辨识初步计算增量式 pid 参数


PID Tuner (untitled/PID Controller) - Step Plot: Reference tracking
PID TUNER
视图
Plant:
Plant
Inspect
PLANT
Type: PIDF
Form: Parallel
Options
CONTROLLER
Domain:
Time
Add Plot
DESIGN
Slower
Response Time (seconds)
Faster
Aggressive
Transient Behavior
Robust
0.02472
0.6
TUNING TOOLS
Reset Design
Show Parameters
Update Block
RESULTS
Data Browser
Step Plot: Reference tracking
Step Plot: Reference tracking
Tuned response
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0.1
0.3
0.5
0.7
0.9
PID Tuner could not find an initial stabilizing controller using plant "Plant"
Controller Parameters: P = 1.357e+07, I = 2.016e+08, D = -1468, N = 9246

第三章 基于 RT-Thread 的下位机软件设计方案

3.1 应用 RT-Thread 原因

针对前后台代码结构存在的CPU利用效率低、实时性差等缺点，RT-Thread实时操作系统通过夺取CPU控制权，然后根据系统定时器列表中的当前最高优先级分配给相应的任务，从而增加了CPU利用率，提升了实时性。围绕着CPU控制权夺取产生了一系列函数完成切换上下文环境、触发中断、保存任务中各种参数等工作。操作系统移植的目的就是将这些函数加入到相应的开发程序中，使之能够适应该硬件平台并且正常运行。其中任务调度功能[2]要求不同优先级任务能够根据优先级进行抢占，同一优先级的任务按时间片轮转执行。

完全模型组以核心板TC264和百度 EdgeBoard 计算卡为主相配合完成任务，多种传感器来帮助完成任务，其中包括：uvc摄像头、增量式测速编码器、陀螺仪块。系统运行过程中，上位机与下位机通过串口通讯来交互配合，检测模块需要与运动控制模块互相配合进行工作，因此，我们需要应用RT-Thread系统对该系统的各个模块协调。

3.2 RT-Thread 移植

在移植[3]源码过程中，尝试了两种移植方法，一种是添加 RT-Thread 源码到原工程中，一种是把原工程代码添加到 RT-Thread 开源库中，经过尝试，第一种方法移植问题较少，移植较顺利，所以采用第一种方法移植程序。

1. 首先建一个文件夹，用来保存 RT-Thread 的源码，在 RT-Thread 的官网下载 RT-Thread 的相关资料,接着先将 src 文件夹里的所有 C 语言文件添加进来，然后添加头文件，分别添加 bsp 文件、include 文件、include\libc 文件为头文件，将他们分别添加进开源 TC264 工程文件相应位置。由于组件部分代码并不涉及到内核部分代码，可在移植成功后进行添加。至此，代码已经添加到了工程之中。

将 RT-Thread 相关文件成功添加到开源 TC2644 工程文件后，还需要对开源库做一定的修改，使之适用于 RT-Thread 操作系统。移植过程中发现并解决了一些问题。

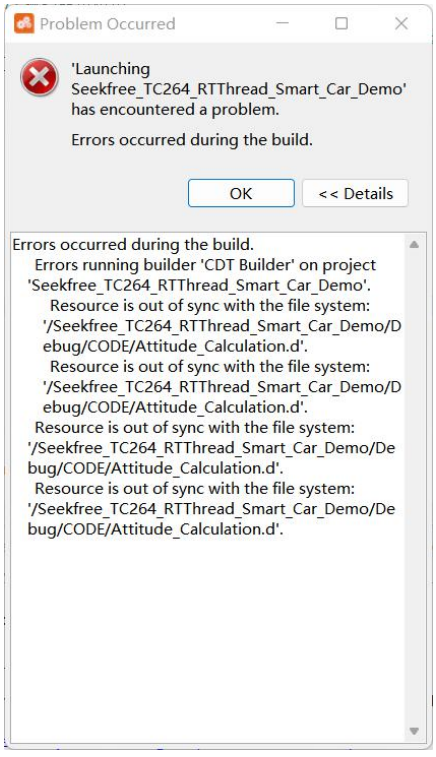


图 1.7 问题一

(1) 原因：原版本为 1.2.2 版本，不适配 RT-Thread 相关文件

解决方法：更新 AURIX™ Development Studio 版本（1.6.0 版本及以上）

(2) USER/Cpu0_Main.d:3: *** multiple target patterns. Stop.

图 1.8 问题二

原因：路径最好只有英文、数字、下划线、文件名开头不能是数字

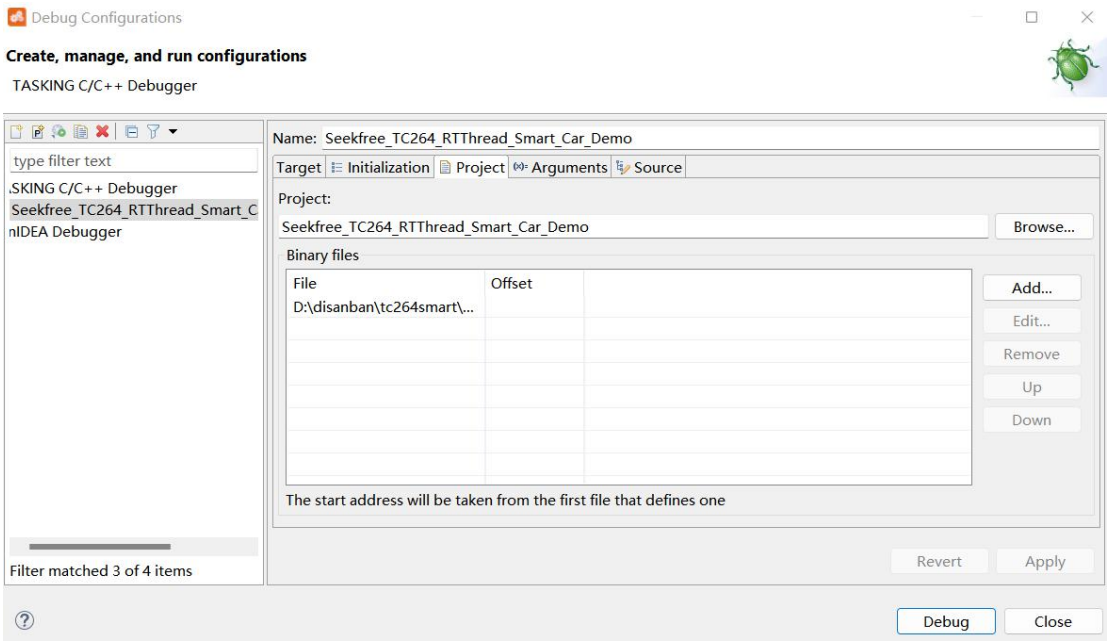
解决方法：按路径、文件命名方法更改路径、文件命名

(3) 使用 AURIX Development Studio 过程中报错（Debug\XXX.elf ‘ is not a binary file supported within this project ）

解决方法：

首先先成功编译一遍，右键工程名字找到 Debug as－>Debug Configurations 或者点击类似小蜘蛛那样的一个按钮右边有一个向下的小箭头找到 Debug Configurations。

然后点击 Project，然后跟着步骤走，点击 File 那个路径，点 Remove,然后点 Add



点击 Browse，在你对应的工作区间下有一个 Debug 文件夹下有一个.elf 文件，把这个路径添加上去

最后点 Apply，点击 Debug，问题就解决了。

(4)

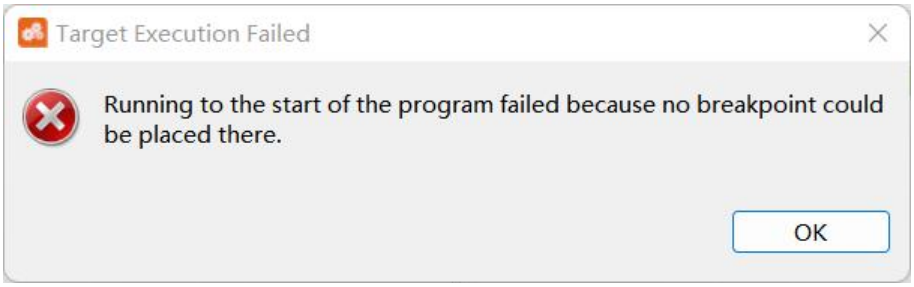


图 1.9 问题三

原因：断点数据失效

解决方法：跳过重启，多重启软件几次

（5）

“*** target pattern contains no '%'. stop”

原因：Cpu0_Main.d 中的内容：Cpu0_Main.c 路径可能没进行配置，路径最好只有英文、数字、下划线、文件名开头不能是数字

解决方法：路径最好只有英文、数字、下划线、文件名开头不能是数字

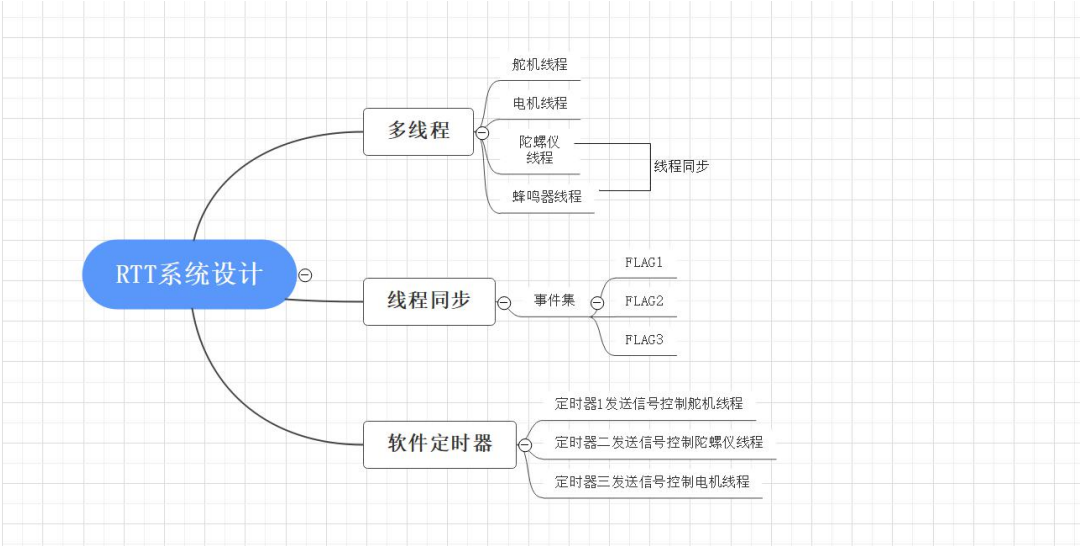
Cpu0_Main.d 中的内容：Cpu0_Main.c 路径写相对路径，编译进行匹配

2.将 RT-Thread 相关文件成功添加到开源库后，并不报错后，还需要对开源库做一定的修改，使之适用于 RT-Thread 操作系统。首先，在 board.c 中添加 rtthread.h 的头文件，在使用了操作系统后，蜂鸣器的使用方式也可以进行相应的优化，我们采用邮箱的方式进行串口的通信：创建 uart_mb 邮箱控制块，串口通信方式 变为串口中断内发送邮件，fish 组件接收邮件。然后，修改 zf_systick.h 文件， 使用操作系统后，硬件延时不再使用，而是使用线程的延时，将开源库中的硬件延时删除掉，均改为线程延时。

3.最后，在 headfile.h 文件中添加 rtthrad.h，以供其他文件调用。 经过以上 4 个步骤以后，RT-Thread 的移植就基本上完成了。

3.3 方案简述

下面为 RTT 多线程同步操作系统在完全模型组 TC264 芯片上的部署方案：



3.4 软件定时器与线程初始化

在 RT-Thread 中，线程是一个非常重要的概念。线程[4]，即任务的载体。一般被设计成 while(1) 的循环模式，但在循环中一定要有让出 CPU 使用权的动作。如果是可以执行完毕的线程，则系统会自动将执行完毕的线程进行删除 / 脱离。通过多线程，就可以把原本串行完成的任务改为并行完成，大大提高了工作效率。

线程一共有 5 种状态：初始状态、运行状态、就绪状态、挂起状态、结束状态。线程又分为静态线程和动态线程，主要使用动态进程。

eg:rt_thread_init()是创建一个静态线程;rt_thread_create()是创建一个动态线程。（静态线程需要事先定义好栈空间）其中，rt_thread 是一个线程控制块，该线程控制块内包含了线程的各种信息，如：名字、优先级、时间片等，而 rt_thread_t 是该线程控制块的指针。

对各个硬件初始化完毕后，创建了以下线程：舵机控制线程，陀螺仪线程，电机控制线程，开启软件定时器并设置触发周期，循环触发。由于在 RTT 中可以设置任意数量的软件定时器，不会再担心定时器够不够用，同时对线程的控制方便我们对代码进行模块性划分，使代码更易修改，更具有可读性。

定时器与线程初始化：

```
                th1=rt_thread_create("th1",th1_entry,NULL,512,10,5);
                th2=rt_thread_create("th2",th2_entry,NULL,512,11,5);
                th3=rt_thread_create("th3",th3_entry,NULL,512,12,5);
                rt_thread_startup(th1);
                rt_thread_startup(th2);
                rt_thread_startup(th3);
//////////////使能三定时器
        timer6_0_0 = rt_timer_create("CCU6_0_CH0",CCU6_0_CH0,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_0_0);
        timer6_0_1 = rt_timer_create("CCU6_0_CH1",CCU6_0_CH1,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_0_1);
        timer6_1_0 = rt_timer_create("CCU6_1_CH0",CCU6_1_CH0,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_1_0);
```

定时器里面调用事件集 FLAG 控制线程：

```
void CCU6_0_CH0(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG1);
    rt_thread_mdelay(3);

}

void CCU6_0_CH1(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG2);
    rt_thread_mdelay(3);

}

void CCU6_1_CH0(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG3);
    rt_thread_mdelay(3);
}
```

3.5 邮箱与蜂鸣器线程

创建一个名为“buzzer”线程，入口函数为 buzzer_entry，函数参数为

RT_NULL，栈大小为 256，优先级为 20，使用 RT_IPC_FLAG_PRIO 优先级 flag 创建的 IPC 对象，在多个线程等待消息队列资源时，优先级高的线程将优先获得资源。而 使用 RT_IPC_FLAG_FIFO 先进先出 flag 创建的 IPC 对象，在多个线程等待消息队列资 源时，将按照先来先得的顺序获得资源。我们选择的是 FIFO 创建 IPC 对象。当需要调用蜂鸣器线程时， 发送邮箱将延时时间作为参数，线程得到参数之后延时时间到就释放 CPU 控制权。

创建邮箱，创建蜂鸣器线程：

```
void buzzer_init(void)
{
    rt_thread_t tid; //用于接收线程返回值

    //初始化蜂鸣器所使用的GPIO
    gpio_init(BUZZER_PIN, GPO, 0, PUSHPULL);

    //创建邮箱
    buzzer_mailbox = rt_mb_create("buzzer", 5, RT_IPC_FLAG_FIFO);//名字自定义

    //创建蜂鸣器的线程
    tid = rt_thread_create("buzzer", buzzer_entry, RT_NULL, 256, 20, 2);  /

    //启动线程
    if(RT_NULL != tid)  //创建线程成功
    {
        rt_thread_startup(tid); //进入就绪状态
    }
}
```

入口函数获取邮箱数据并对蜂鸣器进行控制，由传入数据控制蜂鸣器开启时间。

入口函数定义：

```
void buzzer_entry(void *parameter)
{
    uint32 mb_data;
    while(1)
    {
        //接收邮箱数据，如果没有数据则持续等待并释放CPU控制权
        rt_mb_recv(buzzer_mailbox, &mb_data, RT_WAITING_FOREVER);

        gpio_set(BUZZER_PIN, 1);     //打开蜂鸣器
        rt_thread_mdelay(mb_data);   //延时
        gpio_set(BUZZER_PIN, 0);     //关闭蜂鸣器
    }
}
```

3.6 舵机线程

舵机控制采用位置式 PID，尽量减少舵机延时，使舵机快速响应。

```
void th1_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG1,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    SteerControl();
    count++;
     if(count>=1000)
     {
           count=0;
           pd=1;
           acount=count1;
           count1=0;
     }
     rt_thread_mdelay(3);
    }
}
```

3.7 陀螺仪线程

陀螺仪用于获取车模姿态，为坡道检测起辅助作用。

```
void th2_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG2,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    Read_DMP(&mpu_Pitch,&mpu_Roll,&mpu_Yaw);    //直接使用这三个变量即可
           get_gyro();
           if((sign==2)&&(sign!=last_sign))//圆环
               begin_yaw=mpu_Yaw;
           if((mpu_Pitch>=pitch_max1)&&(mpu_Pitch!=0))
           {
               pitch_max1=pitch;
           }
           if((mpu_Pitch<=pitch_min1)&&(mpu_Pitch!=0))
           {
               pitch_min1=pitch;
           }
           if((mpu_gyro_z>=pitch_ratemax))
           {
               pitch_ratemax=mpu_gyro_z;
           }
           if((mpu_gyro_z<=pitch_ratemin))
           {
               pitch_ratemin=mpu_gyro_z;
           }
          else
          {
              df=0;
          }
          if(sf||df||(sign!=0))
          {
              rt_mb_send(buzzer_mailbox,50);
          }
          else
          {
              gpio_set(P00_12, 0);
          }
          if((sf==1)&&(sign!=90))
           {
               speedset=sps;
           }
          if((df==1)&&(sign!=90))
          {
              speedset=dps;
          }
          if(((sf==1)||(df==1))&&(sign!=90))
          {
              kp1=spkp;
              ki1=spki;
          }
          else
          {
              kp1=50;
              ki1=10;
          }
          rt_thread_mdelay(3);
    }
}
```

3.8 电机线程

电机控制采用增量式 PID，使电机转速快速响应到目标值，并稳定在目标值。

```
void th3_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG3,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    SpeedGet();
    SpeedControl();
    rt_thread_mdelay(3);
    }
}
```

3.9 事件集进行同步

多个事件的集合用一个 32 位无符号整型变量来表示，变量的每一位代表一个事件，线程通过 “逻辑与” 或 “逻辑或” 将一个或多个事件关联起来，形成事件组合。

RT-Thread 中的事件集的特点：1、事件只与线程相关，事件间相互独立。2、事件仅用于同步，不提供数据传输功能 3、事件无排队性，即多次发送同一个事件（且线程还未读走），其效果等同于发送了一次。

当线程等待事件同步时，可以通过 32 个事件标志和这个事件信息标记来判断当前接收的事件是否满足同步条件。

定时器与事件绑定进行联动触发：

```
//////////////////////////////////////事件集与线程
                event1=rt_event_create("event1",RT_IPC_FLAG_FIFO);

void th1_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG1,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
```

```
    void th2_entry(void *parameter)
    {
        while(1)
        {
        rt_event_recv(event1 , EVENT_FLAG2,
                       RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                         RT_WAITING_FOREVER, NULL);

void th3_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG3,
                   RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                     RT_WAITING_FOREVER, NULL);
```

第四章 对 RT-Thread 系统的研究及理解

4.1 对 RT-Thread 系统同步与通信研究

RT-Thread 系统通过对多线程的配合使用，我们得以井然有序地运行本系统，再借助邮箱系统的使用，成功地避免了资源的浪费，如果像裸跑时单纯地一直通过定时器中断去进行信号采集和控制，在运行过程中会出现明显的资源浪费的现象（如陀螺仪读取一次信号值后，在下一次读取之前未能进入陀螺仪的控制程序中，多次读取后才对数据进行使用），为了避免这种情况的发生，我们在这两个线程中加入了邮箱进行通信，在进入控制程序之前通过邮箱发送“Read Angel”对陀螺仪进行采集，采集完成后再回复“Angle Finish”运行控制程序，在程序的多处我们均采用了类似的方法，既降低了代码量，又避免了控制不及时的情况。

4.2 英飞凌 TC264 之双核互斥

TC264 是双核单片机，在用两个和运行同一资源时会出现竞争的问题，例如 CPU0，CPU1 在同时调用一个串口时会出现竞争，导致从串口输出值出现乱码，我们可以通过软件来解决这一问题，这就类似于互斥锁。

解决方案：双方在在开同一扇的同时，谁先拿到钥匙谁就开门，开完门后放回钥匙，等待下一次开门，这样双方就不会出现资源竞争了。按照这个思路我们也可以设置一个全局变量来充当这个互斥量，令这个全局变量为 1（钥匙），其中一核拿到 1（钥匙），让这个全局变量为 0，此时另一核判断全局变量为 0（没拿到钥匙），拿到钥匙的核中执行完相应的程序后，再令设置的全局变量为 1，这样就是先抢钥匙，抢到钥匙在进行相应的程序，就不会存在资源竞争的问题。

4.3 总结

本文从机械、硬件、软件等方面详细介绍了车模的设计和制作方案。从机械和硬件方面，我们力求电路的稳定可靠，尽量使车模结构稳定对称；在软件方面，我们努力追求最优的行驶路径，并不断提高鲁棒性。在做完全模型组这个新组别中，我们学到很多知识，逐渐对 PaddlePaddle 框架，英飞凌单片机、PCB 电路设计等有了比较好的认识，在移植 RT-Thread 过程中，遇到较多问题，但我们凭借坚持与耐心解决了一个一个的问题中。

本文在单片机上完成了 RT-Thread 操作系统的移植，并详细介绍了具体的移植过程。车模实际运行起来的整体效果较传统裸机操作好，主要是因为 RT-Thread 的可裁剪性和实时性，统一的接口，强稳定性和良好的移植性。这些优点极大程度上便利了嵌入式应用程序的开发、管理和维护。我们在对 RT-Thread 的学习过程中，较为熟练的应用了多线程、定时器、事件集的同步、邮箱、信号量，从而对于操作系统不同优先级线程的抢占、同一优先级对线程

进行时间片轮询的机制，有了较为深刻的理解。

RT-Thread 作为一种嵌入式实时操作系统，具有占用资源少、组件丰富、可移植性强的优点，内核代码完全由国内团队自主研发，完全开源[5]。在本届完全模型组的备赛过程中，我们充分体会到了嵌入式实时操作系统的优势，以及团队协作、坚持不懈的重要性。

参考文献：

[1]陈瑞雪,王宜怀,王庭琛.实时操作系统 RT-Thread 启动流程剖析[J].现代电子技术,2022,45(12):36-42.DOI:10.16652/j.issn.1004-373x.2022.12.007.

[2] 王兆滨,韩鹏程.MSP432 的 RT-Thread 操作系统移植[J].单片机与嵌入式系统应用,2021,21(05):39-42.

[3]熊谱翔,全召.RT-Thread Smart 微内核操作系统概述[J].单片机与嵌入式系统应用,2021,21(03):9-12+17.

[4]崔业梅,杨焕峥.基于 RT-Thread 的多线程任务实时运行仿真及应用[J].数字技术与应用,2021,39(10):23-26.DOI:10.19695/j.cnki.cn12-1369.2021.10.08.

[5]宫健,裴焕斗,唐道光.RT-Thread 操作系统的可信验证平台设计[J].单片机与嵌入式系统应用,2022,22(04):34-37.

附录

附录 A 核心算法和子程序及其源码

Main 函数： 各种初始化，核心算法调用

```
#include "headfile.h"
#include "rtthread.h"
#include "display.h"
#include "timer_pit.h"
#include "Encoder.h"
#include "buzzer.h"
#include "button.h"
#include "Motor.h"
#include "elec.h"

#pragma section all "cpu0_dsram"

struct Car_Information  myCar;
struct Ele_Information  myEle;
volatile CAR_STATUS_e   carStatus = CAR_START;
volatile CAR_STATUS_e   Last_carStatus = CAR_STOP;

rt_sem_t camera_sem;
void CarInformationInit(int mode)
{
   if(mode==0)
   {
    //舵机参数
    myCar.g_iSteeringCenter     = 458;//舵机中值  430
    myCar.g_iSteeringLeftLimit  = 110;//左极限
    myCar.g_iSteeringRightLimit = 115;//右极限



    myCar.g_fSteerKD[0]  = 0; //自定义的舵机的D值 5.9
    myCar.g_fSteerKD[1]  = 0;//4.6
    /////电机PID参数
    myCar.g_fSpeedKP = 8;//10.6
    myCar.g_fSpeedKI = 30; //58
    myCar.g_fSpeedKD = 0.22;
```

```
    //左转P值
    myCar.g_fSteerKP[0] = 0.9;//用
    myCar.g_fSteerKP[1] = 0.95;//1.89
    myCar.g_fSteerKP[2] = 1.05;//2.02
    myCar.g_fSteerKP[3] = 1;//2.19
    myCar.g_fSteerKP[4] = 1;//2.25
    myCar.g_fSteerKP[5] = 1;//2.31
    myCar.g_fSteerKP[6] = 1;//2.05
    myCar.g_fSteerKP[7] = 1;//2.17
    myCar.g_fSteerKP[8] = 1.2;  //用 P最大为15.8  2.24

    ////////////////////////////////


    //右转P值
    myCar.g_fSteerKP[10] = 0.9;//用  1.87
    myCar.g_fSteerKP[11] = 0.95;//1.98
    myCar.g_fSteerKP[12] = 1.05;//2.07
    myCar.g_fSteerKP[13] = 1;//2.20
    myCar.g_fSteerKP[14] = 1;//2.32
    myCar.g_fSteerKP[15] = 1;//2.30  大弯
    myCar.g_fSteerKP[16] = 1;//2.06  圆环
    myCar.g_fSteerKP[17] = 1;//2.16
    myCar.g_fSteerKP[18] = 1.2;  //用 P最大为17.8 2.20
//    ki1=10;
//    kp1=50;


     ////////////////////////////////
  }
```

```
    else
    {
      int FlashSector_Now = 10;
      int Flash_Addr_Now  = 920;

      //舵机参数
       myCar.g_iSteeringCenter     = 458;//舵机中值  444
       myCar.g_iSteeringLeftLimit  = 110;//左极限
       myCar.g_iSteeringRightLimit = 115;//右极限
       //左转P值
            myCar.g_fSteerKP[0] = 1;//用
            myCar.g_fSteerKP[1] = 1;//1.89
            myCar.g_fSteerKP[2] = 1;//2.02
            myCar.g_fSteerKP[3] = 1;//2.19
            myCar.g_fSteerKP[4] = 1;//2.25
            myCar.g_fSteerKP[5] = 1;//2.31

            myCar.g_fSteerKD[0]  = 0; //自定义的舵机的D值 5.9
       //右转P值
            myCar.g_fSteerKP[10] = 1;//用  1.87
            myCar.g_fSteerKP[11] = 1;//1.98
            myCar.g_fSteerKP[12] = 1;//2.07
            myCar.g_fSteerKP[13] = 1;//2.20
            myCar.g_fSteerKP[14] = 1;//2.32
            myCar.g_fSteerKP[15] = 1;//2.30  大弯
//          myCar.g_fSteerKP[16] = 1;//2.06  圆环



       //int类型
       spflag  = flash_read(FlashSector_Now,Flash_Addr_Now,uint32);Flash_Addr_Now++;
       xpflag  = flash_read(FlashSector_Now,Flash_Addr_Now,uint32);Flash_Addr_Now++;
       sps     = flash_read(FlashSector_Now,Flash_Addr_Now,uint32);Flash_Addr_Now++;
       dps     = flash_read(FlashSector_Now,Flash_Addr_Now,uint32);Flash_Addr_Now++;
           Flash_Addr_Now=930;

          Flash_Addr_Now=930;
//电机 float
          spki    = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          spkp    = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
//左 float类型
          Flash_Addr_Now=940;

          myCar.g_fSteerKP[0]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[1]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[2]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[3]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[4]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[5]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKD[0]         = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
//右 float类型
          Flash_Addr_Now=950;

          myCar.g_fSteerKP[10]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[11]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[12]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[13]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[14]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          myCar.g_fSteerKP[15]        = ((float)flash_read(FlashSector_Now,Flash_Addr_Now,uint32))/100; Flash_Addr_Now++;
          Flash_Addr_Now=960;


    }
  }
  int check = 0;
```

```
//RTT自定义变量
static rt_timer_t timer6_0_0;
static rt_timer_t timer6_0_1;
static rt_timer_t timer6_1_0;

static rt_thread_t th1;
static rt_thread_t th2;
static rt_thread_t th3;

////////////////////////////////////////////////////////////////////////////////////////
int main(void)
{
    ////////////////////////////////////////////////////////////////////////////////////
    uint32 mb_time=5;
    ////////////////////////////////////////////////////////////////////////////////////
    disableInterrupts();
    int b;
   char bomasign[10]={1,1,1,1,1,1,1,1,1,1};
   char lastbomasign[10]={1,1,1,1,1,1,1,1,1,1};
//   int flAg;
   int flag=1;

   int ii=0;
   char yy=0;
   char bmwz=0;
   IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
   get_clk();//获取时钟频率  务必保留
    //等待所有核心初始化完毕
    IfxCpu_emitEvent(&g_cpuSyncEvent);
    IfxCpu_waitEvent(&g_cpuSyncEvent, 0xFFFF);

    CarInformationInit(1);//车辆参数初始化,1为从flash赋值，0为程序烧写

        oled_init();//OLED初始化

        SteeringInit();//舵机初始化
        MotorInit(13000);//电磁线频率20k，会有5k倍数的谐波，在13k时最小
        MyKeyInit();//按键初始化
        DialInit();//拨码初始化
        EncoderInit();//编码器初始化
        uart_init(UART_0,115200,UART0_TX_P14_0,UART0_RX_P14_1);//串口初始化
        gpio_init(P00_12, GPO, 0, PUSHPULL);//P00_0初始化为GPIO功能、输出模式、输出高电平、推挽输出
////////////////////////////////////////////事件集与线程
                event1=rt_event_create("event1",RT_IPC_FLAG_FIFO);
                th1=rt_thread_create("th1",th1_entry,NULL,512,10,5);
                th2=rt_thread_create("th2",th2_entry,NULL,512,20,5);
                th3=rt_thread_create("th3",th3_entry,NULL,512,12,5);
                rt_thread_startup(th1);
                rt_thread_startup(th2);
                rt_thread_startup(th3);
///////////////使能三定时器
        timer6_0_0 = rt_timer_create("CCU6_0_CH0",CCU6_0_CH0,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_0_0);
        timer6_0_1 = rt_timer_create("CCU6_0_CH1",CCU6_0_CH1,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_0_1);
        timer6_1_0 = rt_timer_create("CCU6_1_CH0",CCU6_1_CH0,
                                     RT_NULL, 5,
                                     RT_TIMER_FLAG_PERIODIC);
        rt_timer_start(timer6_1_0);
//////////////////////使能邮箱蜂鸣器
        buzzer_init();
```

```
while(1)
{
    //处理完成需要将标志位置0
    mt9v03x_finish_flag = 0;
    oled_p8x16str(0,0,"abc");
    gpio_set(P00_12, 1);

    if((gpio_get(P20_10)==0))//6
         {
           a = GetMyKey();
               if(a)
               {
                   b++;   //按键消抖
                   if(b>=100)
                   {   b=0;
                       a = GetMyKey();   //获取按键状态
                       if(a)
                       {
                               switch(a)
                               {
                                   case 1:{uart_putchar(UART_0,182);rt_mb_send(buzzer_mailbox,50);ii++;break;}
                                   case 2:{uart_putchar(UART_0,181);rt_mb_send(buzzer_mailbox,50);startrun++;break;}
                                   case 3:{kd11=kd11+0.5;rt_mb_send(buzzer_mailbox,50);break;}
                                   case 4:{kd11=kd11-0.5;rt_mb_send(buzzer_mailbox,50);break;}
          //
                               }
                       }
                   }
               }
               oled_p6x8str(1,1,"carge");
               oled_printf_int32(61,1,carge,5);
               oled_p6x8str(1,2,"pidcarge");
               oled_printf_int32(61,2,carge1-90,5);
               oled_p6x8str(1,3,"speedset");
               oled_printf_int32(61,3,speedset,5);
               oled_p6x8str(1,4,"rSpeedGet");
                       oled_printf_int32(61,4,g_SpeedGet,5);
                       oled_p6x8str(1,5,"sign");
                       oled_printf_int32(61,5,sign,5);
                       oled_p6x8str(1,6,"startrun");
                       oled_printf_int32(61,6,startrun,5);
                       oled_p6x8str(1,7,"kd11");
                       oled_printf_float(61,7,kd11,3,2);

           }
           else
           {
               DebugUI();
           }
             if(startrun>=10)
                 startrun=0;
             if(ii>=10)
                 ii=0;
             bomasign[0]=gpio_get(P15_0);
             bomasign[1]=gpio_get(P15_1);
             bomasign[2]=gpio_get(P20_14);
             bomasign[3]=gpio_get(P20_12);
             bomasign[4]=gpio_get(P20_13);
             bomasign[5]=gpio_get(P20_10);
             for(yy=0;yy<=5;yy++)
             {
                 if((bomasign[yy]!=lastbomasign[yy]))
                 {
                     oled_fill(0x00); //清屏
                     break;
                 }
             }

             lastbomasign[0]=bomasign[0];
             lastbomasign[1]=bomasign[1];
             lastbomasign[2]=bomasign[2];
                   lastbomasign[3]=bomasign[3];
                   lastbomasign[4]=bomasign[4];
                   lastbomasign[5]=bomasign[5];
     }
}

#pragma section all restore
```

Control.h

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//RTT函数
extern rt_event_t event1;
# define EVENT_FLAG1 (1 << 0)
# define EVENT_FLAG2 (1 << 1)
# define EVENT_FLAG3 (1 << 2)
void CCU6_0_CH0(void *parameter);
void CCU6_0_CH1(void *parameter);
void CCU6_1_CH0(void *parameter);
void th1_entry(void *parameter);
void th2_entry(void *parameter);
void th3_entry(void *parameter);
```

Control.c：定时器内容

RTT 函数：多线程、邮箱、事件集同步

```
////////////////////////////////////////////////////////////////////////////////
uint16 uart_buff = 0;
uint16 last_buff = 0;
uint16 last_time = 0;
int8 last_sign=0;
static int count;
static int count1;
int last1;
int juli;
rt_event_t event1;

char pd=0;
void CCU6_0_CH0(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG1);
    rt_thread_mdelay(3);

}

char yuanhuanflag;
char sf=0;
char df=0;
void CCU6_0_CH1(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG2);
    rt_thread_mdelay(3);

}

void CCU6_1_CH0(void *parameter)
{
    rt_event_send(event1, EVENT_FLAG3);
    rt_thread_mdelay(3);
}
```

```
void th1_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG1,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    SteerControl();
    count++;
     if(count>=1000)
     {
           count=0;
           pd=1;
           acount=count1;
           count1=0;
     }
     rt_thread_mdelay(3);
    }
}
void th2_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG2,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    Read_DMP(&mpu_Pitch,&mpu_Roll,&mpu_Yaw);    //直接使用这三个变量即可
           get_gyro();
           if((sign==2)&&(sign!=last_sign))//圆环
               begin_yaw=mpu_Yaw;
           if((mpu_Pitch>=pitch_max1)&&(mpu_Pitch!=0))
           {
               pitch_max1=pitch;
           }

           if((mpu_Pitch<=pitch_min1)&&(mpu_Pitch!=0))
           {
               pitch_min1=pitch;
           }
           if((mpu_gyro_z>=pitch_ratemax))
           {
               pitch_ratemax=mpu_gyro_z;
           }
           if((mpu_gyro_z<=pitch_ratemin))
           {
               pitch_ratemin=mpu_gyro_z;
           }
           if((mpu_Pitch<=spflag)&&(mpu_Pitch!=0)&&(pd==1))//上坡    //-
           {
               uart_putchar(UART_0, (1));
               sf=1;
           }
           else
           {
               sf=0;
           }
           if((mpu_Pitch>=xpflag))
           {
               uart_putchar(UART_0, (2));
               df=1;
           }
           else
           {
               df=0;
           }
           if(sf||df||(sign!=0))
           {
               rt_mb_send(buzzer_mailbox,50);
           }
```

```
            else
            {
                gpio_set(P00_12, 0);
            }
            if((sf==1)&&(sign!=90))
             {
                 speedset=sps;
             }
            if((df==1)&&(sign!=90))
            {
                speedset=dps;
            }
            if(((sf==1)||(df==1))&&(sign!=90))
            {
                kp1=spkp;
                ki1=spki;
            }
            else
            {
                kp1=50;
                ki1=10;
            }
            rt_thread_mdelay(3);
    }
}
void th3_entry(void *parameter)
{
    while(1)
    {
    rt_event_recv(event1 , EVENT_FLAG3,
                  RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                    RT_WAITING_FOREVER, NULL);
    SpeedGet();
    SpeedControl();
    rt_thread_mdelay(3);
    }
}
```

isr.c:上位机与下位机通过串口中断进行交互

```
//串口中断函数
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);
    if(uart_query(UART_0, &uart_buff))
       {

   //        if(uart_buff != last_buff)
   //          {

                    if(uart_buff==255) //帧头
                        flAg=1;
                    if(flAg==1)
                    {
                        shuju[i++]=uart_buff;
                        if(uart_buff==254)  //帧尾
                        {
                            m=1; //接收到一组数据
   //                         uart_putchar(UART_0, (-g_SpeedGet));
                            flAg=0;
                            i=0;
                        }

                    }
                    count1++;

                            last_buff = uart_buff;
   //          }
       }

    //单角度
     carge=shuju[1];
     speedset=shuju[2];
     sign=shuju[3];
     last_sign=sign;
     if(sf==1)
     {
         speedset=35;
     }
     if(df==1)
     {
         speedset=30;
     }
    rt_interrupt_leave();   //退出中断
}
```

Control.c：各种控制具体代码

```
#include "control.h"
#include "isr.h"
#include "isr_config.h"
#include "headfile.h"
#include "buzzer.h"
/*********最小二乘法定义***********/
#define Sample_Num1 10
#define Sample_Num2 3
float   Slope_steer = 0;        //打角变化率
float   Slope_error = 0;        //误差变化率
int     Angle_store[Sample_Num1];
int     Error_store[Sample_Num2];
/*********参数设置与定义***********/
//float        Road = img_column/2;       //摄像头中值
//int        g_centre_line=img_column/2;
float pitch = 0;
float roll  = 0;
float yaw   = 0;
float pitch_rate = 0;
float roll_rate  = 0;
float yaw_rate   = 0;
float calcarge;
float begin_yaw=0;
float pitch_max1=0;
float pitch_min1=180;
float pitch_ratemax=0;
float pitch_ratemin=0;
int32 spflag=-15;
uint32 spflag1;
int32 xpflag=3;
uint32 xpflag1;
int32 carge=90;
uint32     runMode = 0;
uint32     stop_time;//停车时间
/***********舵机输出变量**********/
float g_SteerControlOutput;
int32 carge1;
float g_LastSteerControlOutput;
float g_fSteerKP;
float g_fSteerKD;
uint32 Steer_AccLimit;
/************单电机变量**********/
int32 g_SpeedSet;//单电机双电机共有
float g_CarSpeed;
float g_SpeedOutput;
int Derror= 0;
//int speed=2500;
///***********双电机变量***********/
//int16 g_SpeedSet;//设置速度
//float g_CarSpeed;//编码器测得速度
/*********差速输出变量**********/

/************函数声明START***********/
void SpeedStrategy();
void SpeedControl();
int32 Speed_turn(uint32 Speed);
/*************函数声明END*************/
float Line_k,Line_b;
float rate_debug;
float R_f=0;

int Run_time = 0;
int Speed_statu = 0;
int32 shuju[10];
int8 sign;
int8 angle[3];
int8 angle1;
uint32 stkp[10];
```

```
uint32 skd[10];
int startrun=0;
float spki=68;
float spkp=20;
void LinearFitting_k_b()//最小二乘法线性拟合
{
    float  dataX[8] = {10.0,15.0,20.0,25.0,30.0,35.0,40.0,45.0};
    float  dataY[8] = {myCar.g_fSteerKP[0],myCar.g_fSteerKP[1],
    myCar.g_fSteerKP[2],myCar.g_fSteerKP[3],myCar.g_fSteerKP[4],
    myCar.g_fSteerKP[5],myCar.g_fSteerKP[6],myCar.g_fSteerKP[7]};

    float x, y, xAverage , yAverage;
    float sum_x = 0.0f , sum_y = 0.0f , xySum = 0.0f, x2sum = 0.0f;
    uint8 len = 8;
    for(int i = 0; i < len; i++)
    {
        x=dataX[i];
        y=dataY[i];
        sum_x += x;
    sum_y += y;
    xySum += x*y;
    x2sum += x*x;                                //x的平方和
    }
    xAverage=sum_x/len;
    yAverage=sum_y/len;
    Line_k = (sum_x*sum_y/len-xySum)/(sum_x*sum_x/len-x2sum);
    Line_b = (sum_y-Line_k*sum_x)/len;

}
/////////////模糊化控制/////////////
float LinearFitting(float x)
{
  float kp;
  kp = x * Line_k + Line_b;
  return kp;
}

float Line_Fitting(float x1, float x2, float y1, float y2, float x3)
{
  float k = (y2 - y1) / (x2 - x1);
  float y3 = y1 + k * (x3 - x1);
  return y3;
}

float Error_div[9]={10,13,20,16,20,24,28};//摄像头p分段
float kd11=4;
//float Error_you[9]={-4,-8,-12,-16,-20,-24,-28};  // 71-68大弯
//float Error_div[9]={5,10,15,20,25,30,35,40,45};//摄像头p分段
uint32 g_SteerControlOutput_tempp=0;
//float calcarge;
/**********原基于摄像头的程序**************/
void SteerControl()
{
    //static float g_SteerControlOutput_temp[5]={1125,1125,1125,1125,1125};//缓存数组
    //static float g_SteerControlOutput_last=0;
  /***************旧方法(分段PD)******************/

    int g_iLeftLimit  = 0;
    g_iLeftLimit = (myCar.g_iSteeringCenter + myCar.g_iSteeringLeftLimit);
    int g_iRightLimit = 0;
    g_iRightLimit = (myCar.g_iSteeringCenter - myCar.g_iSteeringRightLimit);
    static float Error = 0,DError = 0,Last_Error = 0;//一般偏差乘的系数都是浮点类型的，所以偏差不管是整形的还是浮点型的最终都是变成浮点型进行运算，所以干脆直接把偏
    Last_Error = Error;
    float avcarge;
    static int a=0;
    static int sum=0;


    Error=carge;
    a=a+1;
```

```
       if(a<=2)
       {
           sum+=Error;
           avcarge=Error;
       }
       else
       {
           avcarge=sum/2.0;
           sum=0;
           a=0;
       }
       Error=avcarge;
       DError = Error - Last_Error;

      if(fabs(Error-90) < Error_div[0])
      {
        g_fSteerKP=myCar.g_fSteerKP[0];
        g_fSteerKD=myCar.g_fSteerKD[0];
      }
      else if(fabs(Error-90) < Error_div[1] && fabs(Error-90) >= Error_div[0])
      {
          g_fSteerKP=myCar.g_fSteerKP[1];
//        g_fSteerKP=Line_Fitting(Error_div[0],Error_div[1],myCar.g_fSteerKP[0],myCar.g_fSteerKP[1],fabs(Error));
        g_fSteerKD=myCar.g_fSteerKD[0];
      }
      else if(fabs(Error-90) < Error_div[2] && fabs(Error-90) >= Error_div[1])
      {
          g_fSteerKP=myCar.g_fSteerKP[2];
//        g_fSteerKP=Line_Fitting(Error_div[1],Error_div[2],myCar.g_fSteerKP[1],myCar.g_fSteerKP[2],fabs(Error));
        g_fSteerKD=myCar.g_fSteerKD[0];
      }
      else if(fabs(Error-90) < Error_div[3] && fabs(Error-90) >= Error_div[2])
      {
          g_fSteerKP=myCar.g_fSteerKP[3];
//        g_fSteerKP=Line_Fitting(Error_div[2],Error_div[3],myCar.g_fSteerKP[2],myCar.g_fSteerKP[3],fabs(Error));
        g_fSteerKD=myCar.g_fSteerKD[0];


       else if(fabs(Error-90) < Error_div[4] && fabs(Error-90) >= Error_div[3])
       {
         g_fSteerKP=myCar.g_fSteerKP[4];
//         g_fSteerKP=Line_Fitting(Error_div[3],Error_div[4],myCar.g_fSteerKP[3],myCar.g_fSteerKP[4],fabs(Error));
        g_fSteerKD=myCar.g_fSteerKD[1];
       }
       else if(fabs(Error-90) < Error_div[5] && fabs(Error-90) >= Error_div[4])
       {
           g_fSteerKP=myCar.g_fSteerKP[5];
//          g_fSteerKP=Line_Fitting(Error_div[4],Error_div[5],myCar.g_fSteerKP[4],myCar.g_fSteerKP[5],fabs(Error));
        g_fSteerKD=myCar.g_fSteerKD[1];
       }

       else
       {

         g_fSteerKP=myCar.g_fSteerKP[6];
         g_fSteerKD=myCar.g_fSteerKD[1];
       }



        carge1=90+(avcarge-90)*1 + DError*4;//g_fSteerKP  g_fSteerKD
        if(avcarge>=88&&avcarge<=92)
          {
              avcarge=90;
              carge1=90;
          }

    g_SteerControlOutput=(1/90.0*carge1+0.5)/3.3*1000;

//    g_SteerControlOutput = myCar.g_iSteeringCenter + Error*g_fSteerKP + DError*g_fSteerKD;
    SteeringOutput(g_SteerControlOutput);
```

```
int pit_time1;
void SpeedStrategy()
{

    /**********************状态判断*************/
    Last_carStatus = carStatus;



    /*********************************状态判断END*********************************/

    switch(carStatus)
    {
        case CAR_STOP :
            g_SpeedSet = 0;
            break;

        case CAR_NORMAL :
            g_SpeedSet = myCar.g_fSpeedNormal;
            break;
        case CAR_RAMPWAY :
            g_SpeedSet = myCar.g_fSpeedRampway;
            break;

        case CAR_Triroad :
            g_SpeedSet = myCar.g_fSpeedTir_road;
            break;

        case CAR_INTriroad :
           g_SpeedSet = Speed_turn(myCar.g_fSpeedINTir_road);
           break;
//
        case CAR_GARAGE :
            g_SpeedSet = myCar.g_fSpeedGarage;
            break;

         case CAR_ENTERTURN :
             g_SpeedSet = myCar.g_fSpeedEnterTurn;
             break;

         case CAR_B_ENTERROUND :
             g_SpeedSet = myCar.g_fSpeedB_EnterRound;
             break;
//
         case CAR_ENTERROUND :
             g_SpeedSet = myCar.g_fSpeedEnterRound;
             break;
//
         case CAR_INROUND :
             g_SpeedSet = myCar.g_fSpeedInRound;
             break;
//
         case CAR_OUTROUND :
             g_SpeedSet = myCar.g_fSpeedOutRound;
             break;

//
         case CAR_LEFTTURN :
             g_SpeedSet = Speed_turn(myCar.g_fSpeedNormal);
             break;

         default :
             g_SpeedSet = myCar.g_fSpeedNormal;
             break;
    }
```

```
  if(0) //开环控制
  {
    MotorOutput(g_SpeedSet);
  }
  else//闭环控制
  {
     SpeedControl();
  }
    MotorOutput(g_SpeedOutput);
}


//转向速度控制
int32 Speed_turn(uint32 Speed)
{

}

int speedset=30;
//speedset=10;
//int time;
int last;
int a;
uint16 dianya;
float ki1=10;
float kp1=50;
int speedsignl;
uint32 sps=45;
uint32 dps=25;

/***********************
函数功能：电机控制增量式/
用到的外部变量： g_SpeedGetLeft 实际速度 SpeedGet(void)采集得到
                 g_SpeedSetLeft 目标速度 DirectionControl()计算得到
                 myCar.g_fSpeedKP        myCar.g_fSpeedKI        myCar.g_fSpeedKD

//可加积分分离
void SpeedControl()
{


   static float Error = 0,DError = 0,Last_Error = 0,Last2_Error = 0;
   float KP,KD,KI;
   g_CarSpeed = -1*g_SpeedGet;
   Last2_Error = Last_Error;
   Last_Error  = Error;
   Error  = speedset - g_CarSpeed;
   DError = Error - Last_Error;
//   int control_diall = GetDial();


    KP = myCar.g_fSpeedKP;//myCar.g_fSpeedKP
    KD = myCar.g_fSpeedKD;
    KI = myCar.g_fSpeedKI;//myCar.g_fSpeedKI

   g_SpeedOutput += kp1*(DError) + ki1*Error+ 0*(Error - 2*Last_Error + Last2_Error);//KD*(Error - 2*Last_Error + Last2_Error)
   if(g_SpeedOutput >  SPEED_MAX)  g_SpeedOutput = SPEED_MAX;
   if(g_SpeedOutput <  SPEED_MIN)  g_SpeedOutput = SPEED_MIN;

   MotorOutput(g_SpeedOutput);
}
```

ImageProcess.zip　　RTT.zip

附录 B 车模技术检查表

队伍名称	仰仪 9 队			
参赛学校	中国计量大学			
赛题组组别	四轮电磁　四轮摄像头　多车编队 平衡单车　无线充电　平衡信标 智能视觉　极速越野　√ 完全模型			
检查项目	规格 （选手自行填写）	符合 （√）	不符合 （×）	备注
1.车模类型是什么？	1.I 车模	√		如果是自制车模，请标明自制。
车模整体尺寸： 1.（包括传感器在内）长，宽，高(mm) 2. 对于无线充电组：显示电能的 LED 板尺寸	长 30cm 宽 19cm 高 60cm	√		在填写是，请将所在组别规则对于车模尺寸限制同时进行填写。
1. 传感器种类、规格(型号)数量。	Infineon 单片机 1 百度的 Edgeboard 1 工业相机 2 Usb 转 ttl 1 陀螺仪 mpu6050　1 AB 相编码器　1	√		

1. 控制转向舵机型号是否自行改装舵机？ 2. 防伪易损标签是否完整？	I 模型车舵机 CS-3120 无改装舵机 是	√		
1. 是否增加伺服电机？ 2. 如果有那么种类、个数和作用？	否	√		
1. 微处理器型号和个数？ 2. 是否复合所在比赛组别要求？	Infineon TC264 单片机 1 块 百度的 Edgeboard 1 块 符合	√		
1. 是否具有其它可编程器件，个数与作用？	否	√		
1. 是否有无线通讯装置？ 2. 如果有，那么种类和个数？	否	√		
1. 电池的种类、规格和数量？	I 模型车电池 CB-22003 标称电压：11.1V； 标称容量：2200mah； 放电倍率：25C； 接头类型：T 型插头。 1 个	√		
1. 是否有升压电路驱动舵机和后轮电机？	无升压模块驱动舵机 有升压模块驱动后轮电机	√		
1. 后轮驱动电机是否是原车模电机？ 2. 是否具有防伪易损标	是 是	√		

签？				
1. 车模轮胎是否原有的纹理可辨析？ 2. 轮胎表面是否具有粘性物质？ 3. 对于麦克纳姆轮是否更换过小轮胶皮？	1.是 2.否	√		
1. 车模底盘是否是原车模底盘？ 2. 是否有大面积切割？	是 否	√		
1. 车轮轴距、轮距是否改装？ 2. 改装参数是什么？	否 无	√		
1. 车模驱动轮传动机构是否改装？ 2. 改装方式是什么？	否 无	√		
1. 车模差速器是否改装？ 2. 改装方式是什么？	否 无	√		
1. 车模零件是否更换或改装？ 2. 更换和改装的方式什么？	否 无	√		。
1. 车模电路板个数及功能。 2. 其中是否有购买成品、哪一些？	车模有六个电路板 百度的 Edgeboard：图像处理 主控：自制，运动控制 串口通讯 驱动：自制，驱动电机 串口：自制和成品，用于串口通讯 陀螺仪：成品，用于姿态检测	√		

	oled：成品，用于查看参数 编码器：成品，用于测速			
1. 自制电路板是否标记有学校名称、队伍名称、制作日期等信息？ 2. 标示信息在PCB的哪一层？	1.是 队伍信息内容：CJLU_仰仪9队_2022 2.pcb的顶层和底层	√		请在表格中注明电路板队伍信息的内容。
其它待说明内容	无			
检查人员签名：	检查意见：			