第十六届全国大学生

智能汽车竞赛

**RT-Thread创新论文**

基于RT-Thread的全向赛车算法开发

学 校： 山东大学

队伍名称： TSH

参赛队员： 芦子晟 曹晓东 刘家宝

带队教师： 陈桂友 吴皓

关于研究论文使用授权的说明

本人完全了解全国大学生智能汽车竞赛关保留、使用研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：

带队教师签名：

日 期：

目 录

[第1章 引言 1](#_Toc214816195)

[1.1 概述 1](#_Toc214816196)

[1.2 整车设计 1](#_Toc214816197)

[1.3 RT-Thread技术 2](#_Toc214816197)

[1.4 本文结构安排 3](#_Toc214816198)

[第2章 移植RT-Thread Nano到RISC-V 4](#_Toc214816200)

[2.1 添加RT-Thread Nano到工程 4](#_Toc214816201)

[2.2 适配RT-Thread Nano 4](#_Toc214816204)

[2.3 配置RT-Thread Nano 5](#_Toc214816205)

[第3章 RT-Thread在调参寻迹等方面的资源使用 6](#_Toc214816200)

[3.1 线程 6](#_Toc214816201)

[3.2 信号量的使用 8](#_Toc214816201)

[3.3 邮箱的使用 8](#_Toc214816202)

[3.4 软件定时器的使用 9](#_Toc214816203)

[第4章 RT-Thread为单核全向行进车模带来的益处 1](#_Toc214816200)0

[4.1 节约硬件定时器资源 1](#_Toc214816201)0

[4.2 减少全局标志位 提升封装性 简化复杂的控制逻辑 1](#_Toc214816203)0

[4.3 提高实时性 1](#_Toc214816203)4

[第5章 结论 1](#_Toc214816200)5

[附 录A 程序源代码](#_Toc214816209) I

第一章 引言

1.1 概述

飞思卡尔杯全国大学生智能车竞赛以“立足培养，重在参与，鼓励探索，追求卓越”为指导思想，反映当代大学生综合运用所学知识和探索创新的精神。

今年的全向行进组使用麦克纳姆轮，综合应用电磁、摄像头和陀螺仪等传感器完成从车库触发沿着赛道运行两周，然后在返回车库。车模需要分别通过三岔路口两条岔路。车模每经过三岔路口时，需要改变一次行进方向，即在车模前后行进和左右行进两种模式下切换。

由于今年赛道元素多样，传感器类型多样，并且全向行进组需要在三岔中实现横行，所以使用传统的裸机开发逻辑复杂，大量应用全局变量作为标志位，层层使用条件语句，代码可读性下降，也不便于实现模块化。故在车模制作过程中本组参赛人员移植了由上海睿赛德电子科技有限公司拥有并负责开发维护和运营RT-Thread操作系统，最终使得车模软件逻辑结构简单明了，程序运行井然有序，实时性和封装性得到提高。

1.2 整车设计

车模主控芯片使用基于开源内核RISC - V的沁恒CH32V103国产芯片，通过应用麦克纳姆轮并改变前后左右四个电机的旋转方向和转速能够实现通过车模全方位移动，两圈运行过程中，在三岔路以外赛道四个电机同方向转动前行，通过三岔路时车模左或右旋转90°并分别横行通过三岔路的两条支路。

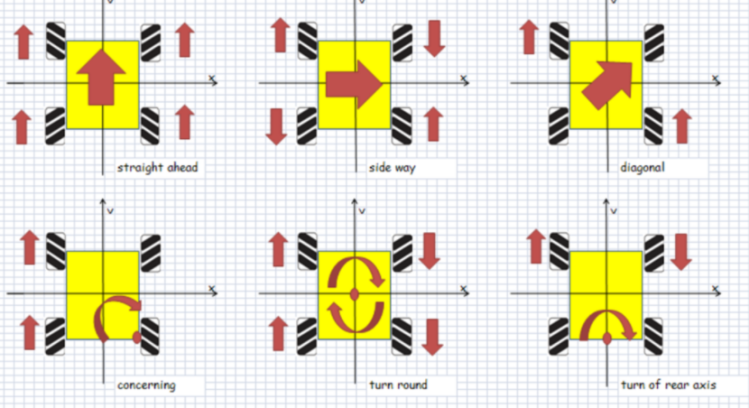


图1.1 麦轮运行规律

对于传感器的应用，摄像头主要应用在寻迹中进行数据采集和控制电机差速实现车模转向寻迹，电感辅助进行环形赛道和三岔路口的识别，陀螺仪在车模直行和横行相互转换时自身旋转90°中实现闭环控制。

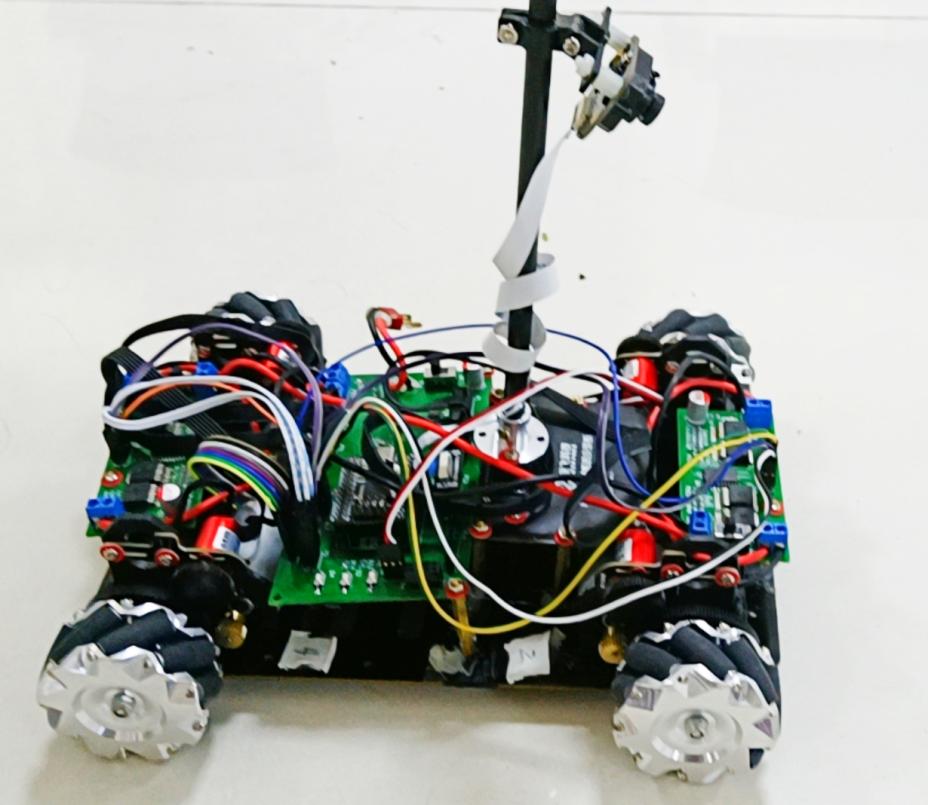


图1.2 TSH队车模

1.3 RT-Thread技术

RT-Thread是近年来发展迅速的嵌入式操作系统，是市面上装机量大、开发者数量最多、软硬件生态最好的物联网操作系统之一。RT-Thread诞生于2006年，由上海睿赛德电子科技有限公司拥有并负责开发维护和运营。因其十四年的沉淀积累，其高可靠性、超低功耗、高可伸缩性和中间组件丰富易用等特性极大地满足了物联网市场的需求。

在车模软件设计中，由于微控制系统资源受限，故使用了RT-Thread Nano内核版本，通过代码仓Gitee下载资源。



图1.3 RT-Thread Nano

1.4 本文结构安排

本文从RT-Thread Nano的移植开始介绍，给出车模在调参，寻迹，处理特殊赛道如三岔路时横行、进入环岛和出入车库等等中规划和使用的内核资源。重点是对比了裸机运行与加入RT-Thread操作系统，详细分析出使用RT-Thread操作系统的益处。最后在结论部分进行总结，并给出程序源代码以供参考。

第二章 移植RT-Thread Nano到RISC-V

2.1 添加RT-Thread Nano到工程

添加 Nano 源文件：

打开MounRiver软件开发环境，在工程下board中配置修改示例代码 rtthread-nano/bsp中的文件board.c，并在Libraries中新建 rtthread\_libraries 文件夹，并在该文件中修改添加以下文件：

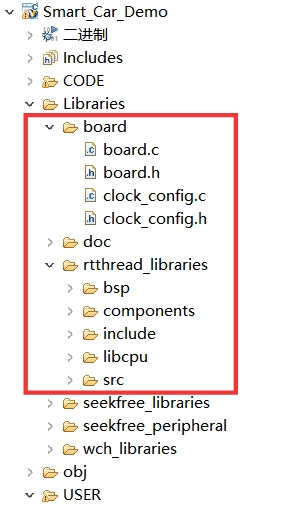


图2.1 添加位置

添加头文件路径：

右击工程，点击 properties 进入下图所示界面，点击 C/C++ Build -> settings ，分别添加汇编与 C 的头文件路径：添加 rtconfig.h 头文件所在位置的路径，添加 include 文件夹下的头文件路径。然后点击 C/C++ General -> Path and Symbols ，添加相应的头文件，最后点击应用即可。

2.2 适配RT-Thread Nano

修改 start.S：

修改启动文件，实现 RT-Thread 的启动：由于 RT-Thread Nano 在 GCC 环境下的启动是由 entry() 函数调用了启动函数 rt\_thread\_startup()，所以需要修改启动文件 start.S，使其在启动时先跳转至 entry() 函数执行，而不是跳转至 main()，这样就实现了 RT-Thread 的启动。

中断与异常处理：

RT-Thread 提供中断管理方法，当系统没有实现类似中断向量表的功能，物理中断要和用户的中断服务例程相关联，就需要使用中断管理接口对中断进行管理，这样当发生中断时就可以触发相应的中断，执行中断服务例程。

系统时钟配置：

需要在 board.c 中实现 系统时钟配置与OS Tick的配置，如下图所示。

内存堆初始化：

系统内存堆的初始化在 board.c 中的 rt\_hw\_board\_init() 函数中完成，内存堆功能是否使用取决于宏 RT\_USING\_HEAP 是否开启，RT-Thread Nano 默认不开启内存堆功能，这样可以保持一个较小的体积，不用为内存堆开辟空间。开启系统 heap 将可以使用动态内存功能。

由于车模使用动态线程的创建，所以打开RT\_USING\_HEAP 宏定义，并且根据创建线程数量和使用堆栈空间之和，将其堆栈的空间设置为如下图所示：

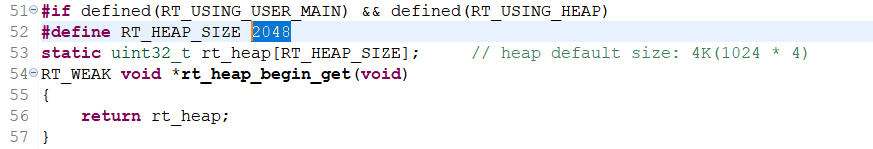


图2.2 修改heap大小

2.3 配置RT-Thread Nano

RT-Thread Nano 的配置在 rtconfig.h 中进行，通过开关宏定义来使能或关闭某些功能：

基础配置：设置系统最大优先级、设置 RT-Thread 操作系统节拍表示多少 tick 每秒、字节对齐时设定对齐的字节个数、设置对象名称的最大长度......

软件定时器是否使用。

IPC 配置：是否使用信号量、互斥量、事件集、邮箱、消息队列。

......

第三章 RT-Thread在调参寻迹等方面的资源使用

3.1 线程

根据需要和ch32v103芯片资源大小，创建线程如下：

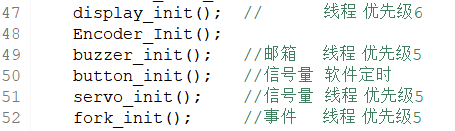


图3.1 所有线程

根据需要和ch32v103芯片资源大小，创建线程如下：

其中display线程用于屏幕显示摄像头图像，耗时最长，优先级最低。

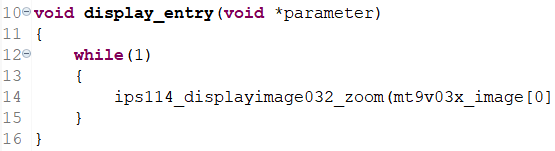


图3.2 display线程

buzzer线程用于根据不同程序识别出不同赛道元素，控制蜂鸣器发出不同时长声音，用于检测是否识别出赛道元素等。

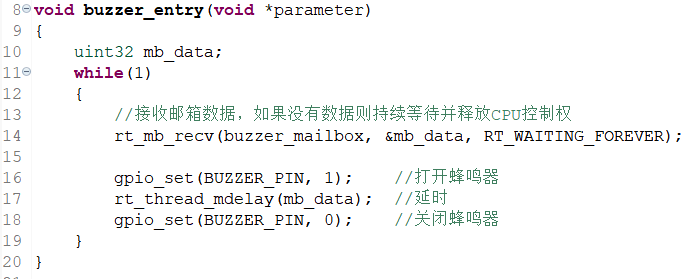


图3.3 buzzer线程

servo线程用于接收三岔路给出的信号，控制舵机带动摄像头旋转，从而使车模横行时摄像头依然朝向正前方寻迹。



图3.4 servo线程

fork线程用于在识别为环岛时总体调度，计算三岔次数、改变行进方式、给出舵机旋转信号量等等



图3.5 fork线程

3.2 信号量的使用

在舵机旋转和按键操作时使用信号量作为触发信号。

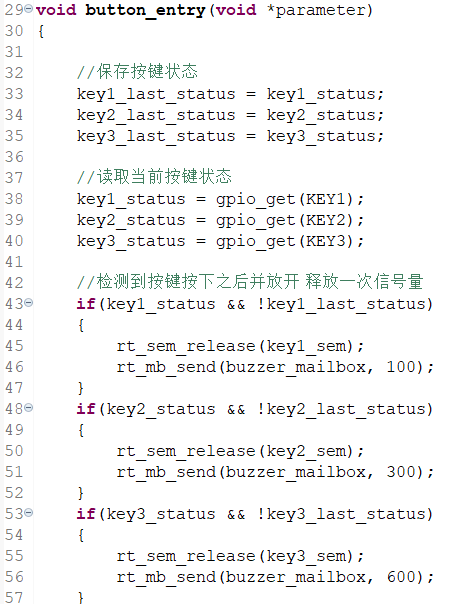


图3.6 button信号量

3.3 邮箱的使用

通过识别不同的赛道元素类型从而给蜂鸣器发送不同声音长度的信号，可以在调参时辅助判断。

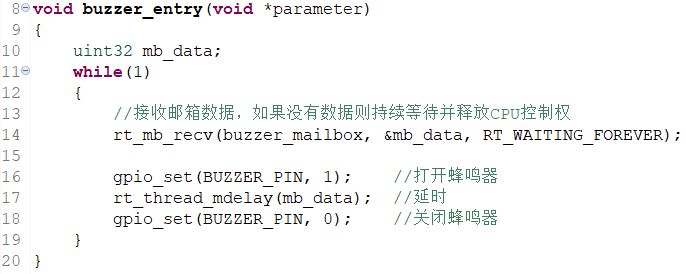


图3.7 buzzer邮箱

3.4 软件定时器的使用

软件定时器有两个应用：定时检测按键是否按下 和 周期处理赛道寻迹信息、计算PID并控制电机PWM输出等。



图3.8 周期中断软件定时器

第四章 RT-Thread为单核全向行进车模带来的益处

4.1 节约硬件定时器资源

车模使用单核ch32v103芯片，该芯片具有4个定时器可供使用，在前期裸机运行时，4个定时器分别供给：周期定时中断、摄像头数据采集、电机PWM信号、舵机PWM信号。资源非常紧张，后期换电机驱动为BTN7971时，需要多使用一个定时器产生多一路PWM驱动电机，幸运的是此时我们队伍已经使用了RT-Thread操作系统，并使用了基于系统滴答定时器的软件定时器，用于供给周期中断，从而节省了一个硬件定时器，使得资源刚好够用。

可见操作系统不仅能够锦上添花，就算是在资源不够充足的情况下，依然能够雪中送炭，以占用很少的资源解决部分其他资源不足所带来的冲突。

4.2 减少全局标志位 提升封装性 简化复杂的控制逻辑

本队程序是基于裸机开发，中期才移植于操作系统，我们认为使用RT-Thread操作系统开发，由于存在线程、信号量、事件等等的概念，可以减少大量全局标志位的使用转而使用这些不同概念，而不是单纯的一律为全局变量。而尽量少地使用全局标志位是非常重要的，同时由于将整个车模的控制分解为一个个的线程，自然而然地提高了系统的封装性，大大降低了开发难度和复杂程度。

以三岔识别和控制算法为例。三岔路是本届智能车比赛非常重要的赛道元素，对于全向行进组尤其是这样，因为在三岔路段涉及到：识别三岔路、控制车模原地旋转、控制舵机带动摄像头反向旋转、将前进由麦轮直行转变为麦轮横行或相反、两次经过三岔分别选择不同岔路等等一系列复杂运算。

在裸机情况下，需要使用的标志位如下：

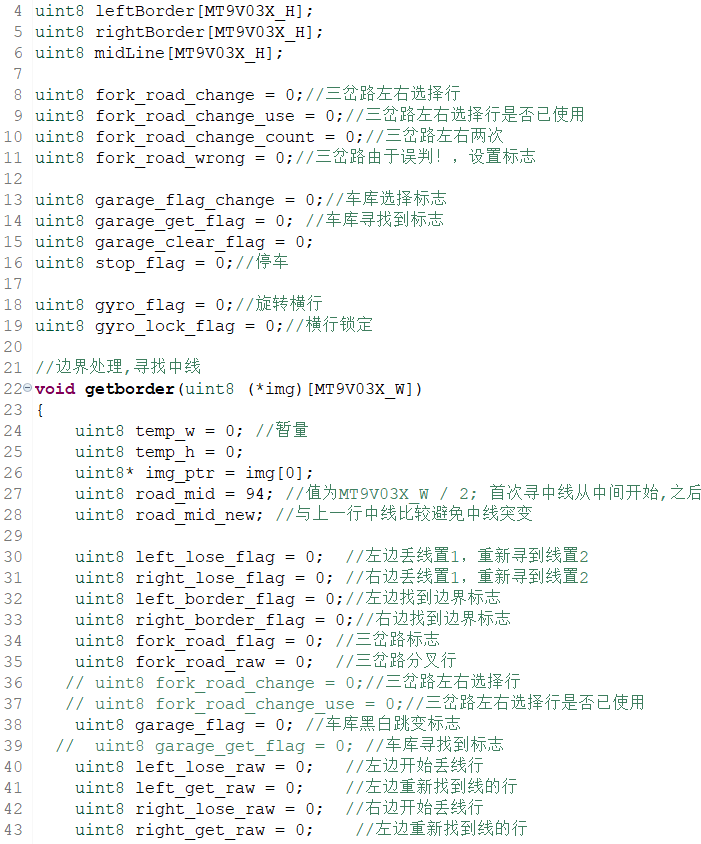


图4.1 裸机全局变量标志位

而在使用RT-Thread操作系统后，将判断出三岔后的一系列控制提取为一个线程，使用一直等待信号量的方式执行三岔内程序：

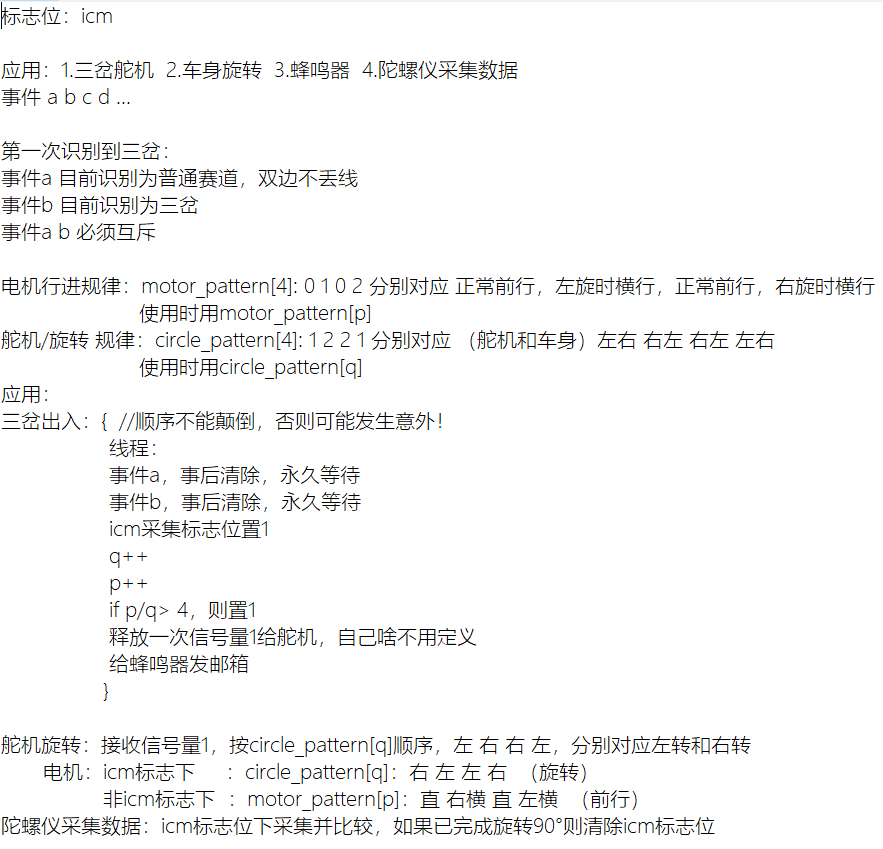


图4.2 操作系统逻辑



图4.3 操作系统逻辑代码实现

可见使用操作系统后有助于逻辑清晰，也提高了代码的可读性。另一证明就是使用操作系统后，一定程度上避免了条件语句的嵌套等等，代码的编译时间缩短到了原来的二分之一，这是高效的系统调度的结果，可以证明代码的质量得到了很大的提高。

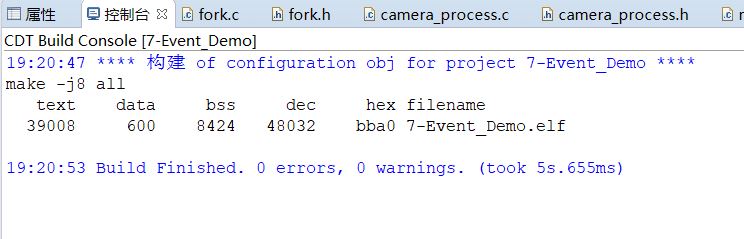


图4.4 使用操作系统后编译完整代码用时

4.3 提高实时性

屏幕显示摄像头图像是调试过程非常重要的一环，但是对于车模正常寻迹则可有可无。屏幕和控制芯片之间虽然采用了SPI这种较为快速的通信方式，但是依然耗时较长，而在裸机控制下，程序显示放在处理摄像头数据之后，二者的权重和地位在程序中是相同的，不仅需要等待屏幕完全显示之后才能再次进入处理摄像头数据，而且还要人为控制摄像头数据处理的时间，如果超出了20ms，则很容易进入hard fault。（非常明显的一点）

而得益于RT-Thread操作系统的线程优先级的概念，main函数中这样安排，看起来井井有条，更重要的是操作系统会为我们调度，只要告诉操作系统线程的实时性排序即可，不必花精力于计算不重要的程序耗时。



图4.5 裸机编程不考虑实时性

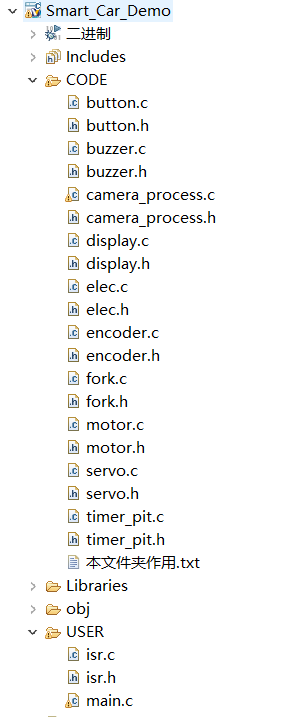


图4.5 操作系统编程考虑实时性

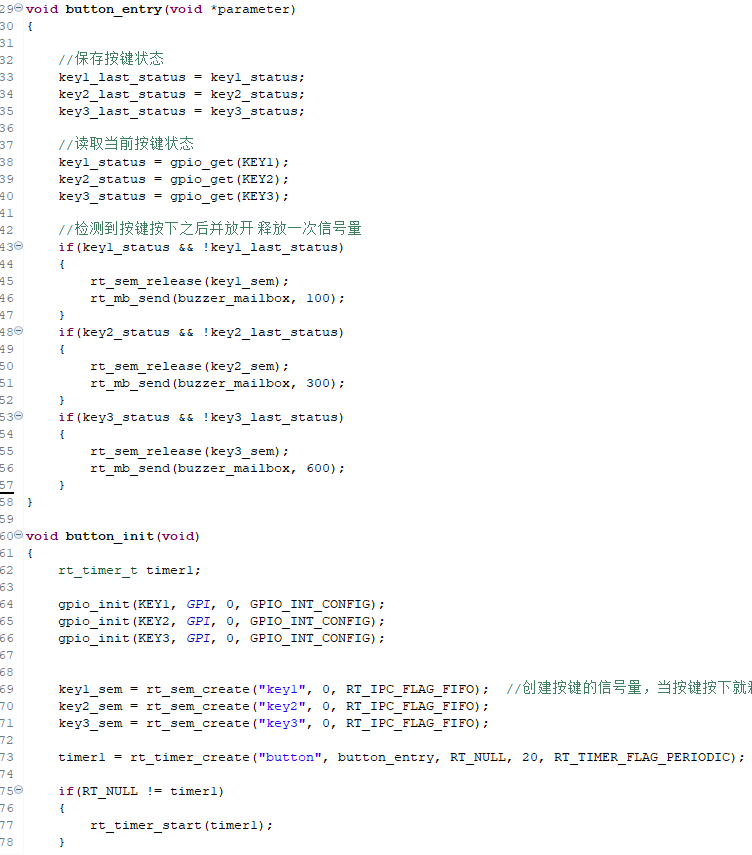
第五章 结论

RT-Thread Nano的移植简明易懂，RT-Thread操作系统本身又在调参，寻迹，处理特殊赛道时发挥了非常重要的作用，尤其是解决了定时器不足的问题；将本该使用的全局变量标志位具体化为信号量、事件等，含义更加清晰，也因此简化了逻辑控制，提高了代码的可读性和封装性。有高效的嵌入式操作系统进行实时的调度，不仅将大脑从复杂的各种考虑中解脱出来，而且使得编译速度大大提高。综上所述，将RT-Thread操作系统应用于智能车的控制中不仅是非常划算的事，更是幸运的事，感谢RT-Thread操作系统让我们见识到了使用操作系统的种种便利！

附录A 源程序代码



图A-1 总体结构



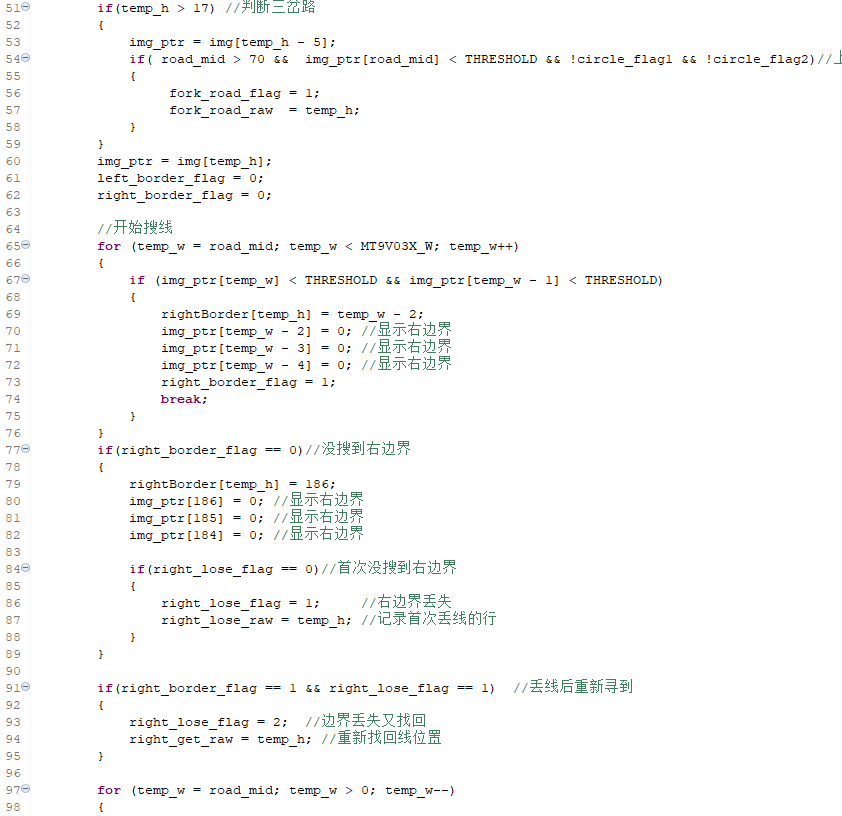
图A-2 Button.c



图A-3 Buzzer.c



图A-4 Camera\_process.c 1



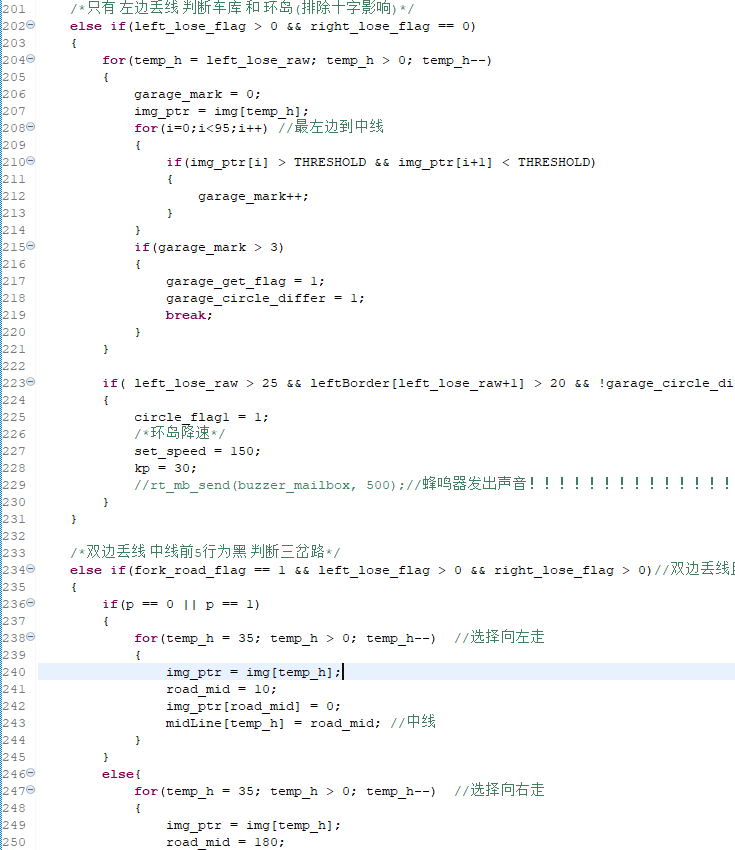
图A-5 Camera\_process.c 2



图A-6 Camera\_process.c 3



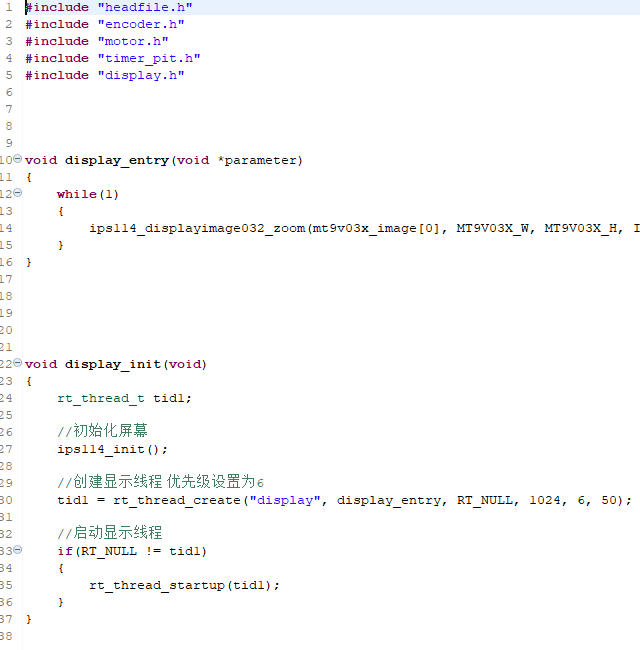
图A-7 Camera\_process.c 4



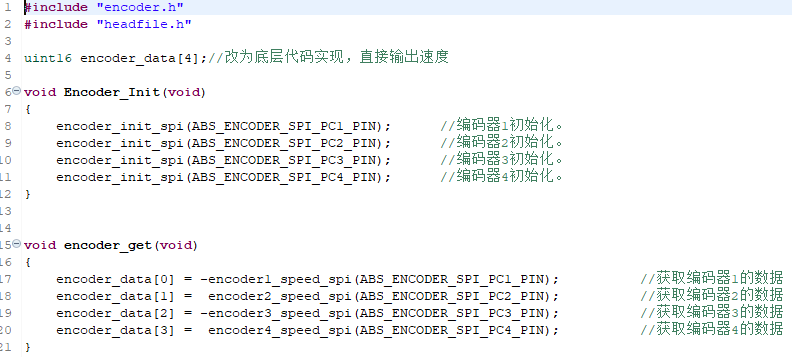
图A-8 Camera\_process.c 5



图A-9 Camera\_process.c 6



图A-10 Display.c



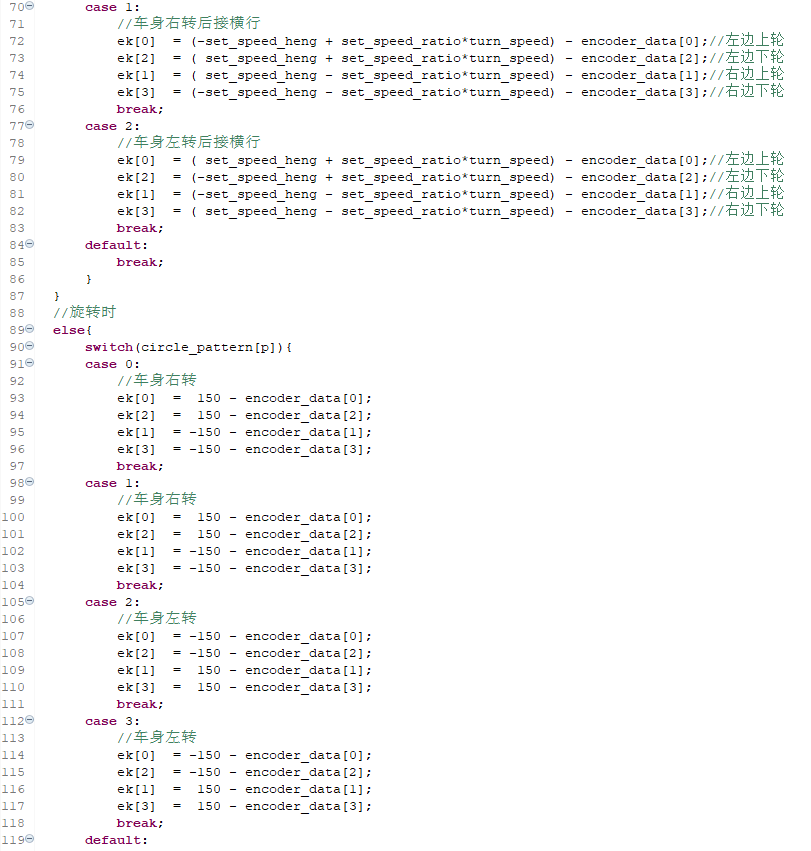
图A-11 Encoder.c



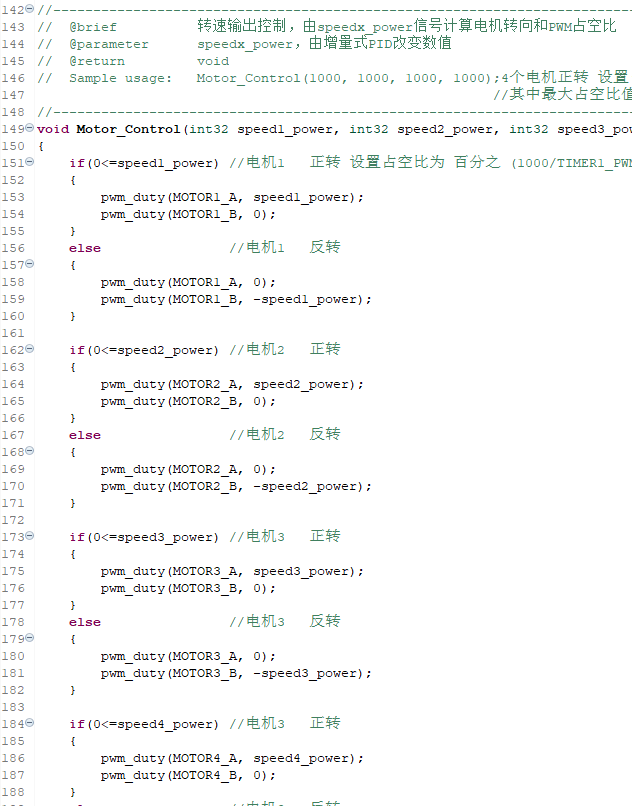
图A-12 fork.c



图A-13 Motor.c 1



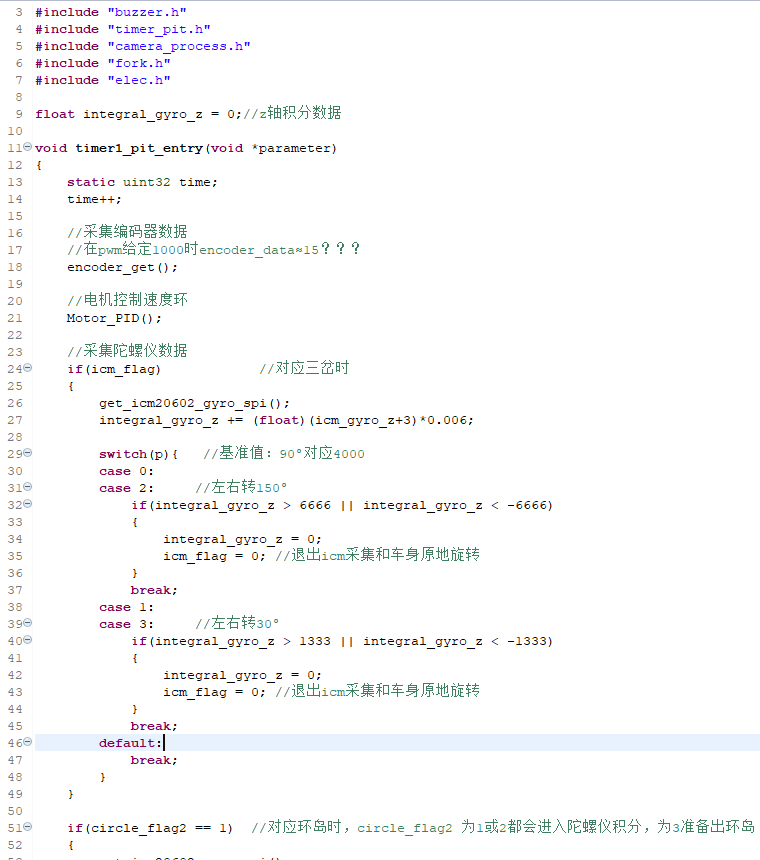
图A-14 Motor.c 2



图A-15 Motor.c 3



图A-16 Servo.c



图A-17 Timer\_pit.c 1



图A-18 Timer\_pit.c 2



图A-19 Main.c