基于RT-Thread和主从双机的麦克纳姆轮智能车的设计与实现

封面

> 略过

研究论文使用授权书

> 略过

目录

主要符号表

> 无（本文简单，没用太多符号）

第一章 引言

1.1 智能汽车制作情况

- 因为比赛规定使用的 `CH32V103` 单片机外设资源有限，且为了节省额外编码器和摄像头的开销费用，同时为了锻炼自身代码能力，我们选择了 `双核+舵机` 的方案。

1.2 RT-Thread技术概述

- 利用 `RT-Thread` 系统的多线程、同步性处理智能车运动控制程序，且同时保证交互程序的正常运行。优势很大。

1.3 相关文献综述

- 保留

1.4 本文概括

- 保留

第二章 系统方案概述

2.1 主从机设计

- 因为 `CH32V103` 单片机定时器外设不足，整个设计都是基于为了能满足定时器使用而设计。
- 在结构体上参考逐飞的单核方案，在程序控制上参考逐飞的双核方案。
- 从机主要负责摄像头图像处理，同时采集后驱两个编码器的数值，周期性与主机通讯。
- 主机主要处理传感器的数据计算电机所需运动控制的输出，同时兼顾交互系统的运作。

2.2 从机设计

- 从机的四组定时器分配：左下编码器、右下编码器、摄像头中断、周期定时器。
- 从机时刻接收图像，等待图像接收完毕后在主循环处理图像。同时周期定时器设定触发频率，每次中断服务都会进行一次编码器采集，并把图像关键数据和编码器数据一同发送到主机。
- 从机还有少许交互功能，设有一个拨码盘、一个蜂鸣器和一个显示屏。拨码盘可改变部分上电时初始化的参数，蜂鸣器可作为小车运行时的提示器，屏幕可作为小车静止时的交互器显示各类参数和图像。

2.3 主机设计

- 主机的四组定时器分配：左上编码器、右上编码器、舵机输出、电机输出。
- 主机时刻接收从机数据，等待接收完毕时代表周期定时已到，借此可以达到主从机的时钟同步效果。每次周期定时已到，就会先采集各个传感器的数据，然后融合、处理、计算得出运动控制所需输出给电机。
- 主机具备完善的交互功能，搭配五向按键和屏幕，结合 `Flash存储` 可以实现在线调参等交互功能。

- 主机除了接收从机的图像特征和编码器数据外，还有各个传感器的数据。其中包括电感、陀螺仪和编码器。

第三章 智能车设计

3.1 硬件设计

- 主控电路板完全自主设计，参考沁恒开源的评估板电路，裁剪下最小系统部分。将单片机的主要功能引脚一一引出，和部分功能电路组成。
- 主从机通用电路有稳压、开关、串口、屏幕和编码器；主机特有电路包含电机输出、舵机输出和电感输入；从机特有电路包含蜂鸣器输出和摄像头输入。主从机通讯方式为串口。

3.2 软件设计

- 编程环境是`Win10`，编程软件是`MounRiver Studio`，编程语言为`C语言`，采用嵌入式系统为`RT-Thread`。
- 在程序处理优先级上，算法处理应该最为优先保证，所以采用了多线程设计拜托了以往全都放在中断或主循环内顺序执行的尴尬。
- 比如，从机的图像处理程序，和从机的交互显示程序，是分别在两个线程中运行。而主机的运动控制程序，和主机的交互显示程序，也是两个线程中运行。
- 同时，为了保证`系统线程`与`硬件中断`能达到实时性，还采用信号量来保证，程序运行逻辑和数据的保护。

第四章 RT-Thread系统设计

4.1 线程管理

- 碍于`CH32V103`的20KB易失数据存储区`SRAM`和64KB用户应用程序存储区`CodeFlash`资源。我并没有设置多余的必要线程，为了防止线程运行时资源不足而溢出导致程序卡死。
- 从机有两个线程，主机有三个线程，其中两个是共同的，都是主函数线程`main_thread_entry`和交互系统线程`show_thread_entry`。主机还有第三个线程是用于参数存储`flash_thread_entry`，用于储存主机在运动控制时所用到的参数。
- 早期其实我为多个功能模块都设置了线程，通过FinSH 控制台查看时知道单片机资源其实勉强够用。但是在我实际长时间运行时就发现程序偶尔会进入到线程调度的死循环中。我起初以为是我个人程序的影响，但当我删去大部分逻辑运算只剩下初始化时依旧很容易卡死。最后才不得不删减掉不必要的线程调用，只采用必须的线程在线持续运行。

4.2 时钟管理

- `RT-Thread`系统提供了函数接口`rt_tick_get`供使用者调用，可以方便的获取系统节拍。我用于记录系统线程或部分程序的执行时间，但是有个弊端，因为在配置文件中设置了系统时间单位为毫秒级。如果程序执行的时间差小于毫秒级，就不能准确计算出执行时间。
- 但是实际上软件系统节拍貌似会因为硬件中断而不是完全可信，在测试RT-Thread系统的中断管理时，发现了这个问题。

4.3 线程间同步

- 共享内存块中有许多数据是需要被多个线程使用的，往往一个线程所用到的数据值需要另一个线程先行计算得出并赋值。如果不加以限制就可能出现数据值还未被计算就使用，或是使用时正在被修

改的冲突情况。为了防止数据出错，有必要加上信号量作保护。保证线程间不会冲突，保持同步性。

- 比如从机的图像数据，是由单片机硬件采集，等到采集完成后才释放信号量。然后再由图像处理线程获取信号量，访问图像数据进行运算。防止了图像数据还未被采集完成就重复处理图像，导致计算错误和浪费资源。主机的串口数据读取也是同理。
- 对于信号量，我还有另一种理解用法。因为`RT-Thread`系统有自身的线程调度，所以我在使用单片机硬件中断时会尽量注意不浪费时间，避免影响线程的同步性，我会使用信号量来保证每当中断需要运行程序时，就会释放信号量，交给特定的线程去执行。
- 比如主机的参数存储程序，由较低优先级的交互系统线程触发执行。如果将Flash修改程序直接放在该线程上执行，会导致程序直接卡死，进入无限循环的任务调度。所以我选择另创建一个专门的线程，用于执行参数存储程序。同时使用信号量保证与交互系统线程的触发同步。

4.4 线程间通信

- `RT-Thread`操作系统的邮箱特点是开销低、效率高，被设计用于线程间通信。参考例程中，我也使用邮箱间接控制蜂鸣器和LED的输出或按键的输入。
- 但是我因为执行该功能的线程执行过程太短，导致线程频繁被调用，浪费资源。同样是参考例程，我选择使用延时函数`rt_thread_mdelay`达到延迟线程的运行周期。使低优先的线程释放更多系统资源给高优先级的线程调度。
- 实际操作后却发现，这种实现方式很容易导致程序卡死。再后来，我发现如果延时函数`rt_thread_mdelay`调用频繁，调用时间过段时，就很容易导致程序卡死在线程调度。再后来发现是我的思路错了，如果想让线程让出处理器资源，正确应该调用`rt_thread_yield`。当前线程首先把自己从它所在的就绪优先级线程队列中删除，然后把自己挂到这个优先级队列链表的尾部，然后激活调度器进行线程上下文切换。
- 但是实际上我的线程优先级并不全在一个级别上。所以考虑种种因素，再加上为了简化程序和节省资源利用。我删减了邮箱线程的使用和邮箱的使用，转而将这部分功能和交互系统线程统合一起。

4.5 中断管理

- `RT-Thread`系统支持软件中断设置，可以节省硬件中断的外设资源。这样可以使本就资源不富裕的单片机可以腾出更多的资源。
- 早期我设计主从机的程序框架时，设定了两种方案：方案一，主从机都使用软件定时器，这样的弊端是数据交流并没有做到完全同步；方案二，从机使用硬件或软件定时器，主机使用串口接收中断。显而易见，方案二会更加严谨。而且我已经采用双核架构，就单片机外设资源已经很充足了，就没必要再节省不必要的资源。
- 方案二需要考虑的就是，选择硬件定时器还是软件定时器。起初两种都试过，我发现个有趣的现象：使用硬件定时器时，函数接口`rt_tick_get`获取系统节拍时得到的结果并不是设定的周期时间，而是缺少了几毫秒。如果使用软件定时器的话就不会出现这个问题，获取到的节拍数就是设定的周期时间。起初我相信软件测试得到的结果，并没有进一步测试。
- 但是很快我就发现在验证主机中断触发时间的间隔时，和我设定的从机中断周期不相同。而且是相差甚远，还会不间断跳动。在咨询沁恒技术人员后，他建议我使用示波器检测引脚电平变化频率，在中断服务中翻转电平。测试发现实则硬件中断的时间才是正确的，软件中断的时间是错误。这也解释了为什么获取时钟节拍数时，硬件中断不正确而软件中断正确了。因为节拍数已经错了，软件中断也是错的。
- 我最后采用从机硬件中断+主机串口中断的方式同步主从机的中断服务函数。虽然节拍数记录是错误的，但是示波器测得是正确的。同时明白到单片机的硬件中断会影响到`RT-Thread`系统的运行节奏。为了减少影响，主机的中断服务函数是放在主循环中获取信号量的方式运行。串口中断中只负责释放信号量。

第五章 结论

5.1 整体过程

- 从机主循环处理图像数据，中断函数采集并将编码器数据和图像数据打包发送。主机串口中断得到数据释放信号量，主循环获取信号量得到串口数据，同时采集编码器、AD值、陀螺仪等传感器的数据，最后融合、计算、输出电机值。这是整个系统的主要运作流程。在这流程间运行的空闲时间中，系统会循环执行优先级较低的交互系统线程。
- 关于参数保存的部分，从机的参数保存仅限于逆透视数组的计算，每次重新下载程序时保存一次即可；主机的参数保存需要实时在线，采用线程+信号量的方式触发执行。

5.2 RT-Thread应用

- 得益于RT-Thread系统的多线程同步性，以往需要中断才能保证运行的程序，只需要改变优先级和时间片就可以很方便的达到需要的效果。节省了单片机资源，最大化利用资源，减少编程难度，同时保证了系统的稳定性。

参考文献

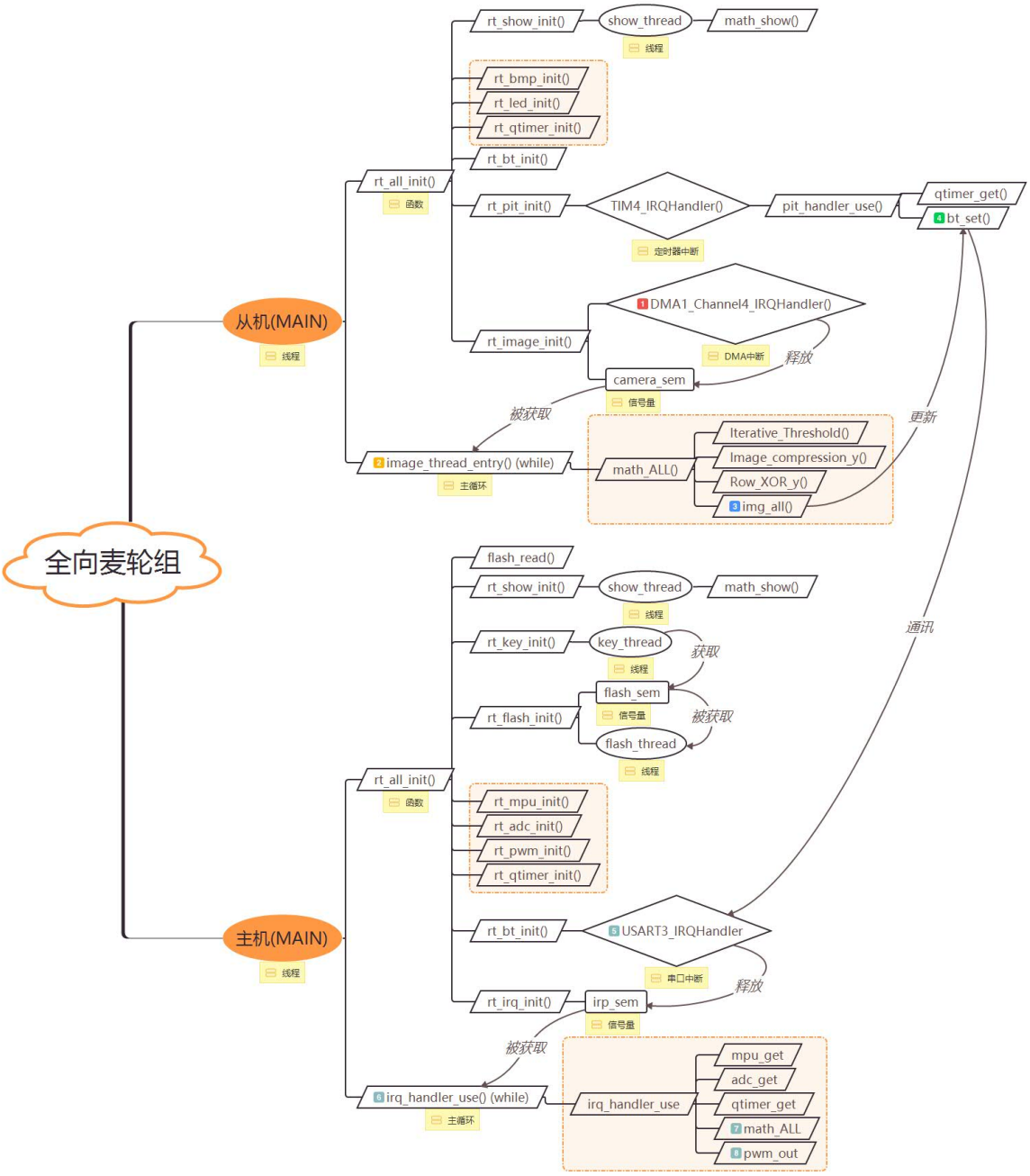
- 保留

附录

程序源码

- 保留

系统框架


全向麦轮组
从机(MAIN)
线程
rt_all_init()
函数
rt_show_init()
show_thread
线程
math_show()
rt_bmp_init()
rt_led_init()
rt_qtimer_init()
rt_bt_init()
rt_pit_init()
TIM4_IRQHandler()
定时器中断
pit_handler_use()
qtimer_get()
4 bt_set()
rt_image_init()
1 DMA1_Channel4_IRQHandler()
DMA中断
释放
camera_sem
信号量
被获取
2 image_thread_entry() (while)
主循环
math_ALL()
Iterative_Threshold()
Image_compression_y()
Row_XOR_y()
3 img_all()
更新
主机(MAIN)
线程
rt_all_init()
函数
flash_read()
rt_show_init()
show_thread
线程
math_show()
rt_key_init()
key_thread
线程
获取
flash_sem
信号量
rt_flash_init()
被获取
flash_thread
线程
rt_mpu_init()
rt_adc_init()
rt_pwm_init()
rt_qtimer_init()
rt_bt_init()
5 USART3_IRQHandler
串口中断
释放
通讯
rt_irq_init()
irp_sem
信号量
被获取
9 irq_handler_use() (while)
主循环
irq_handler_use
mpu_get
adc_get
qtimer_get
7 math_ALL
8 pwm_out