

# **第十七届全国大学生 智能汽车竞赛**

## **技术报告**

学 校：华中科技大学

队伍名称：华中科技大学极速越野一  
队

参赛队员：胡远哲

徐轲

李子晗

带队教师：何顶新 李智勇

## 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。

李子晗

胡远哲

徐珂

参赛队员签名：

带队教师签名：

何红伟

李智勇

日期：2022年8月1日

---

## 目录

第一章 引言 .....	5
1.1 智能车模机械设计 .....	5
1.1.1 L 车模的特点 .....	5
1.1.2 L 车模安装情况 .....	6
1.2 电路部分的设计 .....	8
1.2.1 主控芯片 .....	8
1.2.2 电源管理 .....	8
1.2.3 MCU 稳压模块 .....	8
1.2.4 舵机稳压模块 .....	9
1.2.5 外设电源模块 .....	9
1.2.6 人机交互模块 .....	10
1.2.7 传感器 .....	11
1.2.8 驱动电路 .....	11
1.2.9 电路布局 .....	15
1.3 智能车控制算法简介 .....	16
1.3.1 舵机角度的控制 .....	16
1.3.2 导航点的切换 .....	16
1.3.3 无刷电机驱动 .....	17
1.3.4 自动路径产生法 .....	17
1.4 物联网操作系统概述 .....	19
1.5 实时操作系统概述 .....	19
1.6 后续内容的框架安排 .....	19
第二章 RT-Thread 物联网操作系统介绍 .....	20
2.1 RT-Thread 简介 .....	20
2.2 RT-Thread 的版本与架构 .....	20
2.2.1 RT-Thread 标准版 .....	20
2.2.2 RT-Thread Nano 版本 .....	21
2.2.3 RT-Thread Smart 版本 .....	22
2.2.4 RT-Thread Studio .....	22
第三章 RT-Thread 操作系统的具体应用 .....	23
3.1 RT-Thread 的移植 .....	23

3.1.1 移植的硬件环境.....	23
3.1.2 移植的软件环境.....	23
3. 2 初始化的优化.....	28
3. 3 线程分配与管理.....	31
3.3.1 线程.....	31
3.3.2 线程的创建以及各个线程的作用 .....	31
3.3.3 线程的具体设计 .....	32
3. 4 信号量.....	36
3.4.1 信号量简介 .....	36
3.4.2 GPS 处理与信号量.....	37
3.4.3 按钮的信号量 .....	39
3.4.3 无线串口发送允许的信号量 .....	40
3. 5 定时器的使用.....	40
3.5.1 定时器简介 .....	40
3.5.2 定时器应用 .....	41
第四章 RT-Thread 开发中的创新点 .....	46
4. 1 RT-Thread 的软件包的获取与开发.....	46
4. 2 RT-Thread 参考文档的使用与修改.....	48
4. 3 利用 RT-Thread 社区获取技术解答 .....	49
4. 4 与裸机开发的横向比较.....	49
4. 5 RT-Thread 开发的主要优势 .....	50
第五章 总结与展望.....	51
5.1 总结与感想.....	51
5.2 工作的不足与未来的展望.....	51
附录 A.部分程序源代码 .....	52
Main.c .....	53
Route_generating.c (自动生成路径) .....	55
Time_pit_rtt.c(软件定时器).....	70
GPS.c(GPS 数据处理) .....	73
附录 B.参考文献 .....	79

# 第一章 引言

## 1.1 智能车模机械设计

### 1.1.1 L 车模的特点

L 车模长约 42mm，宽约 25mm，主要材料为尼龙加纤材料，强度较高。车模总体尺寸较大，空间比较宽松，右边空间安装电机和舵机，左边存放电池。由于空间较大，甚至可以存放下高容量的 3S 甚至 5S 电池。

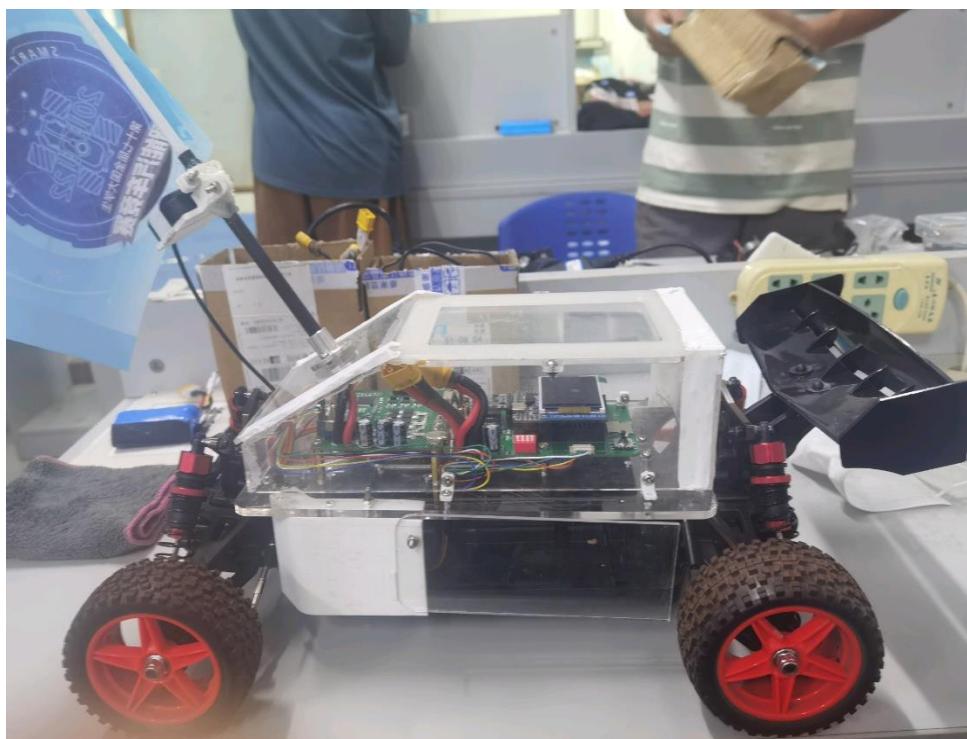


图 1.1.1 带车壳版本的安装方案

转向系统方面，舵机的推荐安装孔位为非对称设计，这点会导致在转向时程序设置舵机左右打角相同而实际打角不同。L 车模很多地方的连接特别是转向系统上的螺钉松紧需要在调车时进行细致把控，不能出现过紧或者过松状况，松紧程度需要不断试错。当舵机打角太大时转向杆会与车模发生干涉，可以通过控制舵机打角或者磨平车模来解决。

车体强度方面，车模在受到强烈碰撞冲击时主要承受载荷的位置为中间的轴骨，而不是底板，也就是说整车结构的强度主要取决于轴骨。轴骨采用了镂空的设计，导致强度很难承受 10m/s 以上速度发生碰撞的刚性冲击。

传动结构方面，L 车模采用一电机带动四驱的设计，将轴骨下的连杆拆除可变为后轮二驱。前后轮都自带差速器方便转向。差速器在轮胎上的螺母拧的太紧的情况下会发生严重打滑，会严重影响车的方向控制。

车辆的轮胎可以通过专门的胶（南大 704）进行密封，使得轮胎更鼓，从而高速起步时能做到更加稳定。

### 1.1.2 L 车模安装情况

由于 L 车模的孔位有限且非常集中, 加上极限越野组对车强度要求很高, 我们组采用的网上定制亚克力板作为底板进行对电路板的安装固定, 固定电路板使用了铜柱加螺丝钉。我们使用了 6.2mm 厚的亚克力板, 但实际用起来还是会出现碎裂的情况。因为亚克力板的硬度大但是难以形变, 所以容易在车祸发生后碎裂。下图是某一次调试时发生事故图, 即使有断电停车装置, 刹车距离过长赛车还是撞到了操场旁边的障碍物。

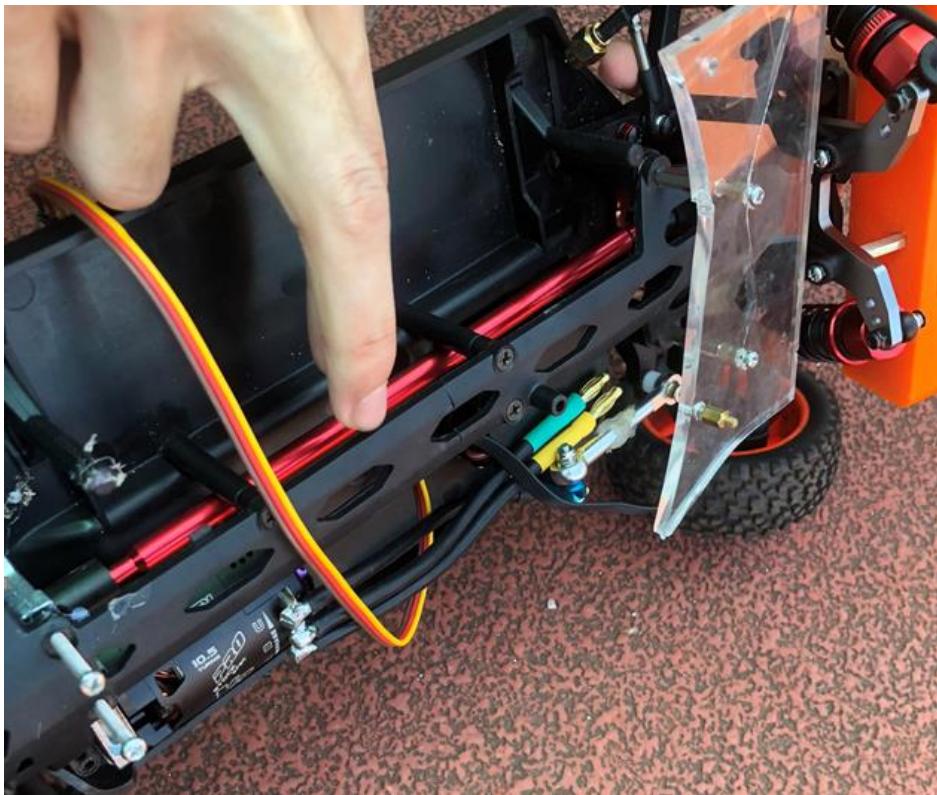


图 1.1.2 某次调试出现的问题

防水车壳是由亚克力板拼接粘合, 鉴于不方便的原因, 非雨天不使用防水车壳。而且防水车壳的强度不理想, 一旦车辆发生侧翻, 车壳难以承受住冲击力而发生损坏。

由于轴骨对车体强度的重要性, 在某次碰撞车模断裂后我们对轴骨进行了简易加强, 在轴骨上绑上了负重铁块, 效果较为明显, 如果有条件还是建议重新设计轴骨。

L 车模默认舵机安装方式为卧式非对称安装, 无法更改, 不多赘述。连杆长度由舵机中值时控制车体保持直线行驶来确定。

前轮倾角采用主销后倾、内倾, 使车轮转向后能及时自动回正和转向轻便。

电机安装有默认位置, 两齿轮的距离需要在不发生打滑的情况下尽量使手推车模时的阻力更小。电机跟电机套间需安装垫片便于电机安装。

车轮调节螺母松紧时在所需调节的一对车轮上贴标签, 舵机中值情况下

---

启动电机观察标签转速至相同。这个操作是为了确保车辆的差速系统能够正常工作，不然转向是容易出现侧偏移。前轮螺母要在不打滑的情况下尽量拧紧，不然会使转向系统较为松散，转向时误差较大，舵机无法回到中值等问题。

GPS 高度没有限高，适中就行，GPS 尽量减少遮挡，保证信号良好。同时 GPS 安装在车辆左边（逆时针路线），减小进内道可能性。比赛时安装的旗帜我们选择装到固定 GPS 的杆子中间的孔中。

## 1.2 电路部分的设计

### 1.2.1 主控芯片

本次比赛我们组主板的主控芯片选择的是灵动微研发的产品 MM32F3277G9P，使用了 ARM Cortex-M3 处理器内核，支持 Thumb-2 与 ARM 指令集。最高工作频率为 120MHz，内置高速存储器（512KB Flash, 128KB SRAM），具有丰富的增强型 I/O 端口和外设，内置了 16 位、32 位通用定时器，16 位基本、高级定时器，12 位的 ADC 与 DAC，以及模拟比较器。

无刷驱动的主控芯片我们采用的是 MM32SPIN27PS，这是一款专门为电机与电源相关应用设计的产品，使用了 Arm Cortex-M0 为内核的 32 位微控制器，最高工作频率可达 96Mhz，内置 128KB Flash、12KB SRAM，具有丰富的 I/O 端口与外设，以及 2 个 12 位 ADC、5 个模拟比较器、4 个运算放大器、1 个 16 位通用定时器、1 个 32 位通用定时器、3 个 16 位基本定时器和 2 个 16 位高级定时器，支持硬件除法器和硬件开方。

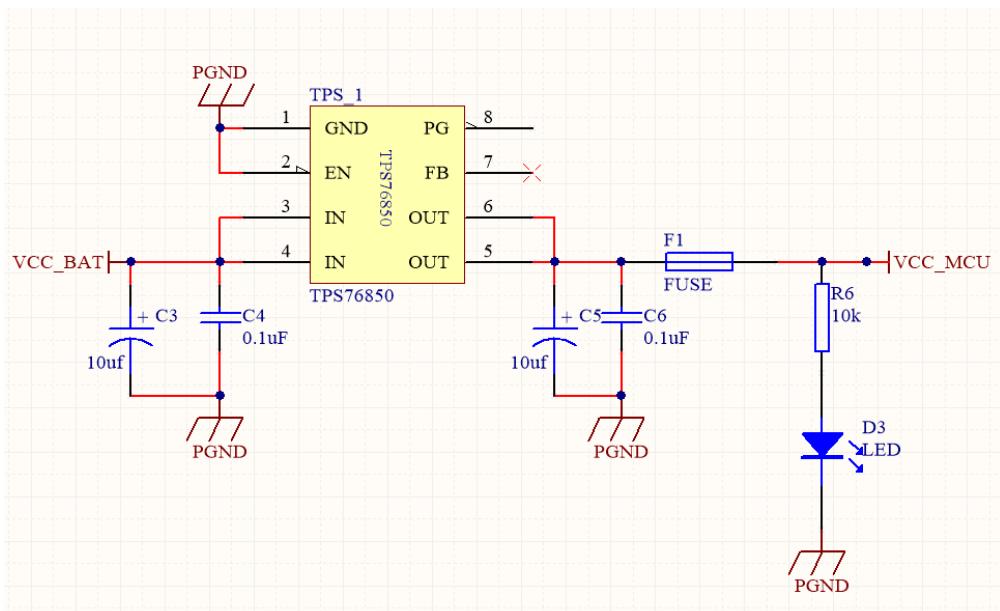
### 1.2.2 电源管理

按照竞赛规则规定以及所用无刷电机的需求，赛车使用 3S 锂电池供电。好的电源管理是将电源有效且稳定的供给各个模块，并使其都能长期稳定高效工作。电源管理的好坏直接关系到小车的表现。

本智能汽车电源管理主要包括：MCU 电源，电机驱动电源，舵机电源，外设模块电源。其中电机驱动电源在驱动电路部分介绍。

### 1.2.3 MCU 稳压模块

MM32F3277 单片机需要 5V 供电，且对纹波有较高要求。故使用线性稳压器 TPS76850 为单片机供电。原理图如下：



---

图 1.2.1 MCU 稳压模块

#### 1.2.4 舵机稳压模块

舵机电源对纹波要求不高，但电流需求较大。根据所用 SD-7 舵机的需求，使用 SPX29302 芯片为舵机提供 6V 供电。原理图如下：

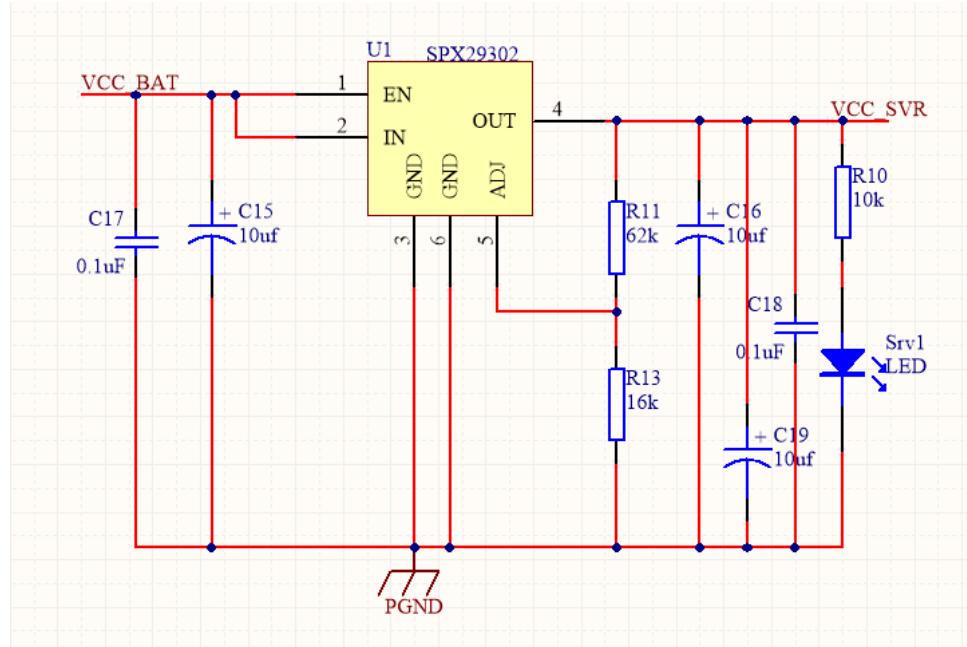


图 1.2.2 舵机稳压模块

#### 1.2.5 外设电源模块

主板的部分外设模块工作在 3.3V 和 5.0V 稳压下，使用 TPS76833, TPS76850 为它们供电。

由于 GPS 模块存在断电重启后坐标偏移的问题，为 GPS 模块供电的 5V 稳压由独立的开关控制，可以在其他模块断电后依旧供电，在正式比赛时，GPS 除更换电池外，全程不断电。原理图如下：

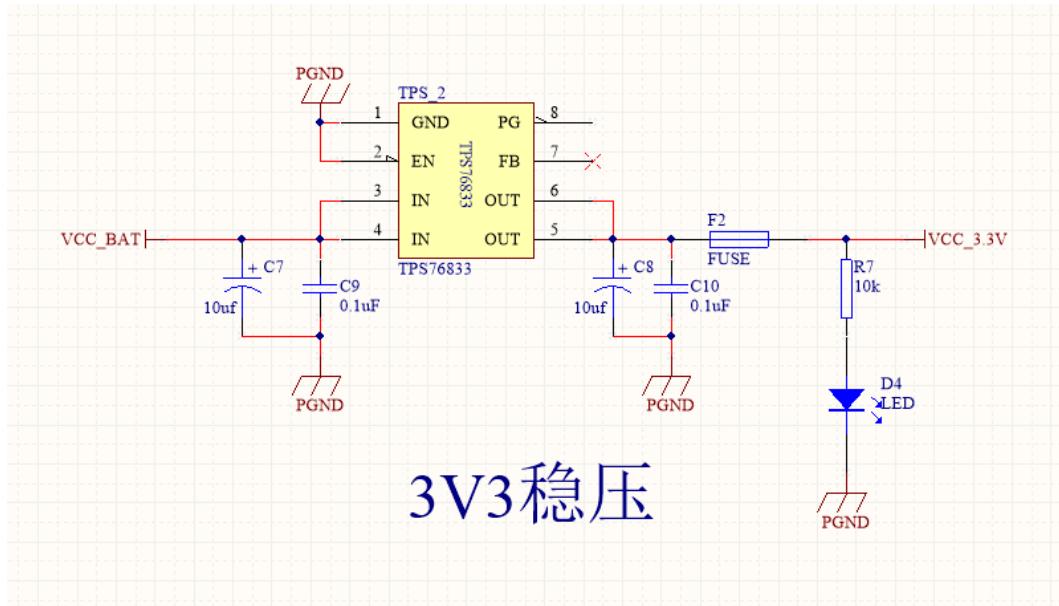


图 1.2.3 3.3V 稳压供电模块

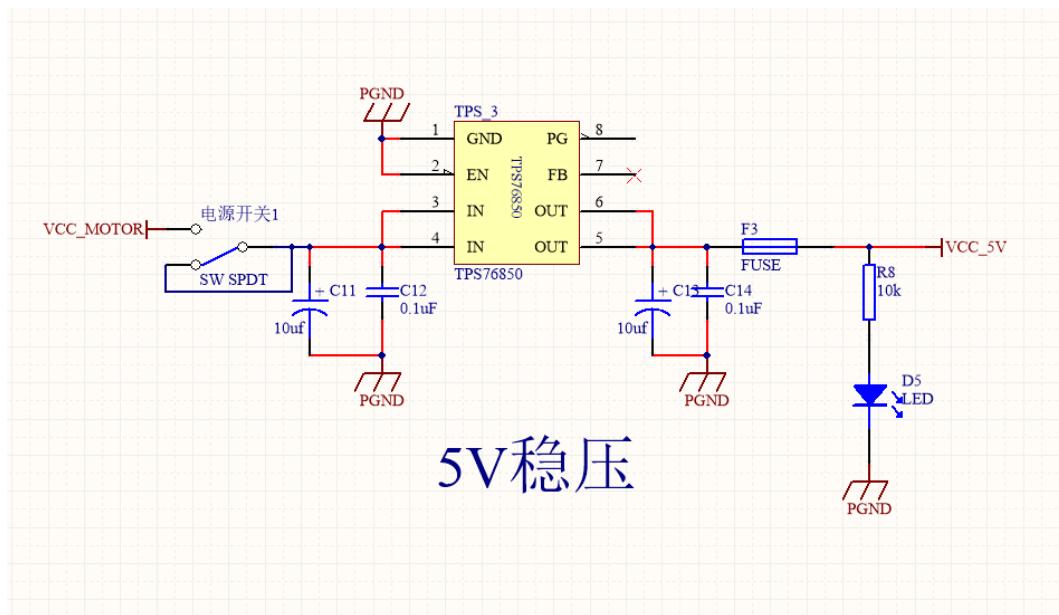


图 1.2.4 5V 稳压模块

### 1.2.6 人机交互模块

本车设置了无线串口模块接口，TFT 显示屏，四路拨码开关，五向按键，TF 卡槽以便于参数的传递与修改，方便了调试。由于极速越野组小车速度较快，跑道较大，常用的 LED 和蜂鸣器无法在行驶过程中起到指示作用，意义不大，改用无线串口模块回传，实时返回小车的运行情况，TF 卡记录下详细数据，便于之后分析。原理图如下：

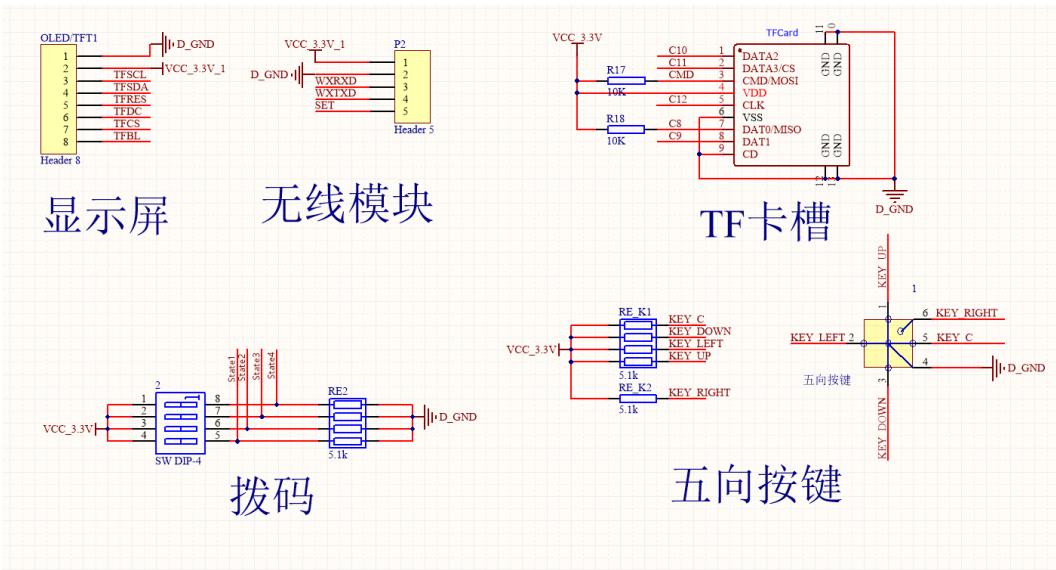


图 1.2.5 人机交互模块原理图

### 1.2.7 传感器

根据规则传感器可以使用 GPS，摄像头，惯性导航，光电管、超声波等。不允许使用 OpenMV。按照赛道情况光电管、超声波意义不大，户外的光线条件以及较高的行驶速度，导致摄像头表现不佳。所以在速度较快后，之后的方案去除了摄像头，只保留了 GPS 和陀螺仪。

GPS 使用双频 GPS 定位模块 TAU1201，定位精度可达一米以内，数据更新速率默认为 1HZ，最大可设置为 10HZ。陀螺仪选用 ICM20602 六轴传感器。原理图如下：

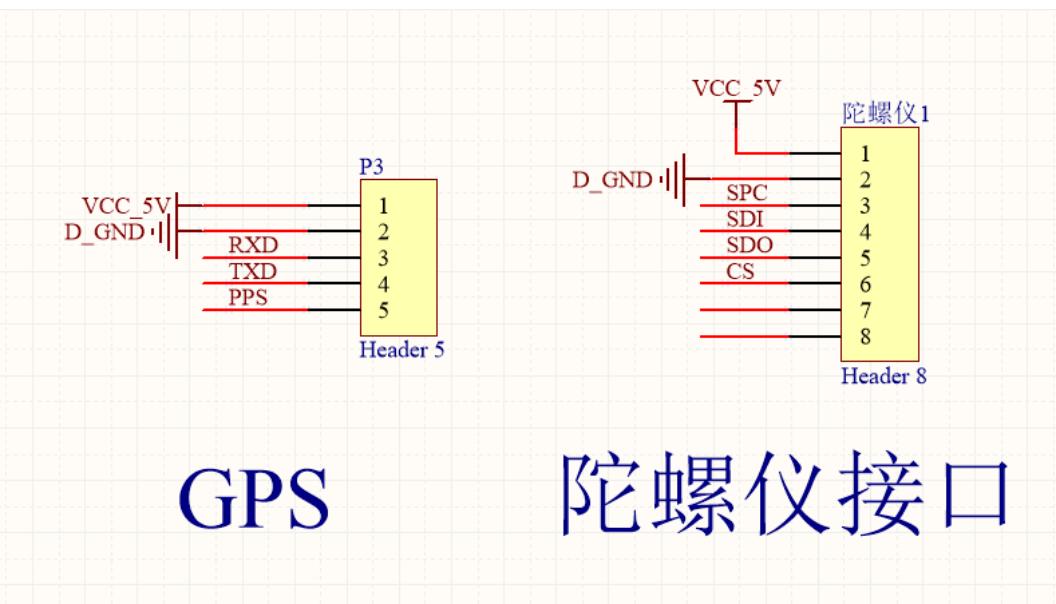


图 1.2.6 传感器部分原理图

### 1.2.8 驱动电路

本次比赛允许极速越野组参赛队伍将车模电机修改成无刷电机，但无刷

电机驱动需要采用基于 MindMotion 的 MCU 的驱动方案。本车基于 MM32SPIN27 设计了无刷驱动方案。

电源管理模块与 MM32F3277 单片机相同，MM32SPIN27 也使用了 TPS76850 芯片供电，但芯片的使能端由 MM32F3277 单片机控制，主控芯片可以随时控制驱动板的工作和停止，保证了主控永远先与驱动上电，同时防止主板与驱动板连接意外断开时出现爆转。另外还设置有电压检测电路以及减小电池电压波动的电容和吸收浪涌功率的双向 TVS。原理图如下：

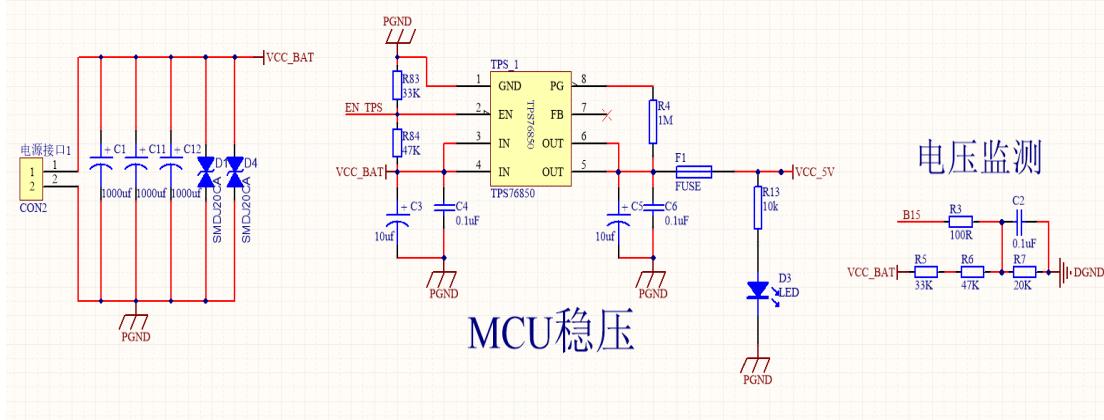


图 1.2.7 驱动板 MCU 稳压

H 桥驱动电路使用了经典的三相 H 桥来驱动无刷电机。MOS 选用 N 沟道 MOS 管 TPH1R403NL，为满足电流需求，使用双 MOS 管并联。二极管 1N4148 使 MOS 管快速放电，可以加快 MOS 管断开。原理图如下：

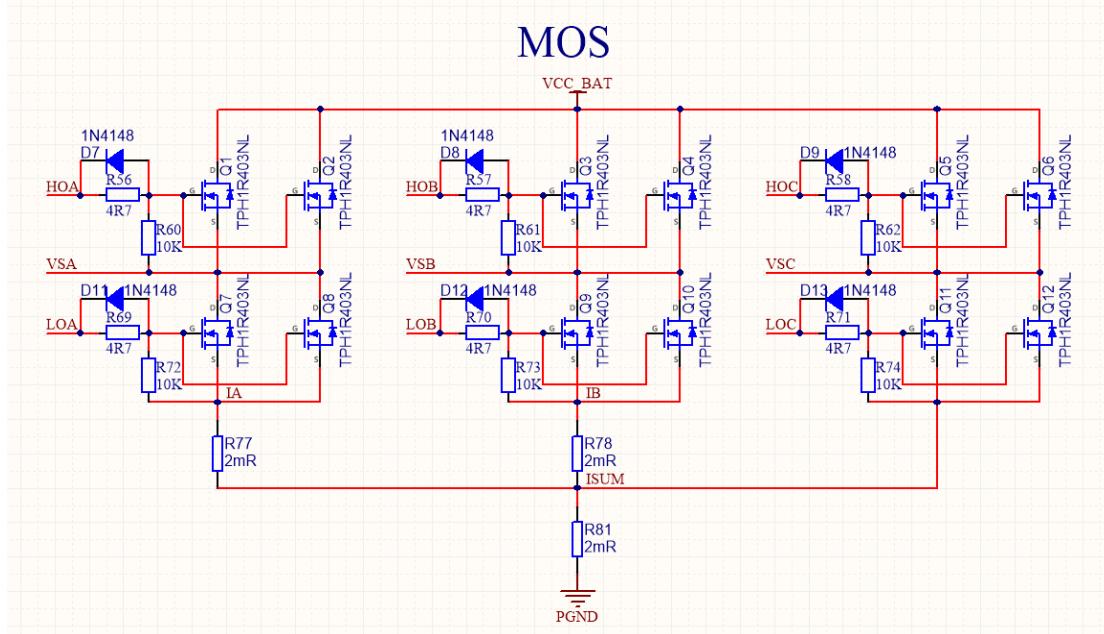


图 1.2.8 H 桥驱动电路原理图

预驱电路使用 EG2181 芯片搭建预驱电路，控制 MOS 的通断，12V 输出保证 MOS 管完全导通，内阻较小。原理图如下：

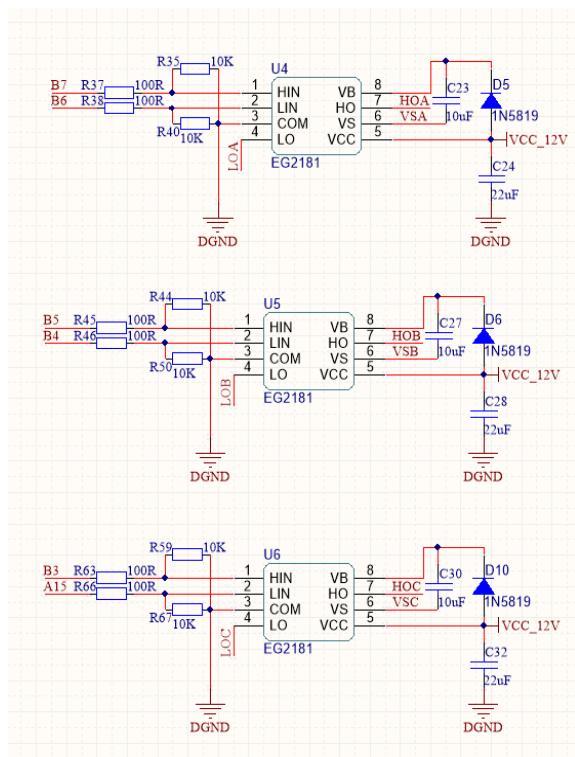


图 1.2.9 预驱动电路原理图

相电流相电压检测使用 2 毫欧电阻采样，再用芯片内部的运放放大后，通过 ADC 引脚检测，利用芯片内部的比较器判断线路电流是否过大，实现堵转保护。原理图如下：

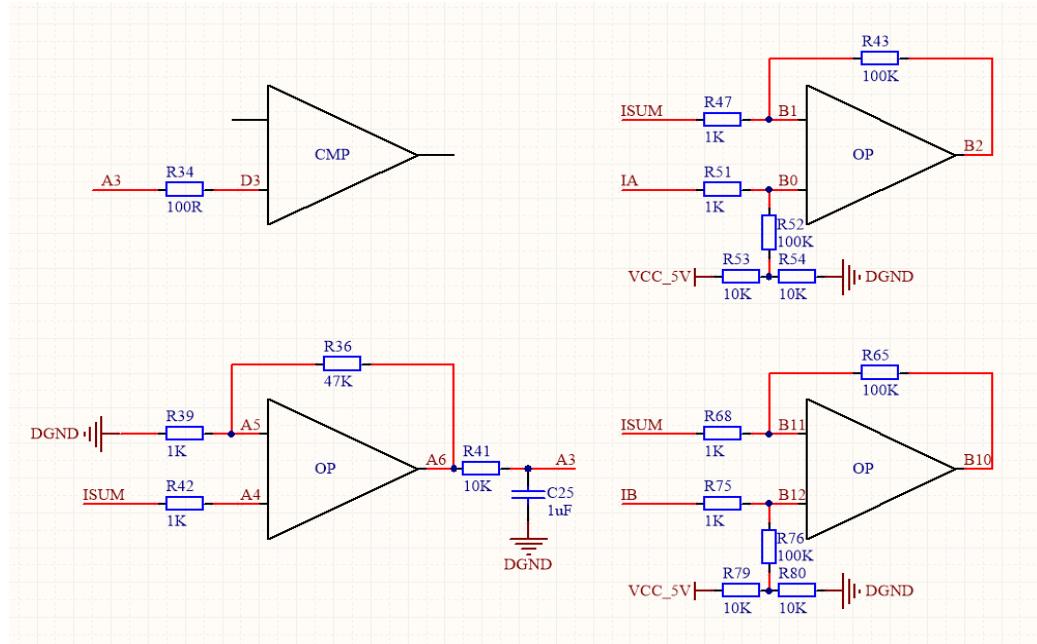


图 1.2.10 堵转保护电路

接口部分，对于主板接口，用于连接驱动板和主板。驱动板接受主板发出的目标转速和方向，同时返回电机的转速和方向。调试接口则用于对驱动

板上的 MM32SPIN27 烧写程序以及在线调试。无刷电机接口用于接入无刷电机的 A、B、C 三相线。无刷电机霍尔接口用于连接无刷电机霍尔线。

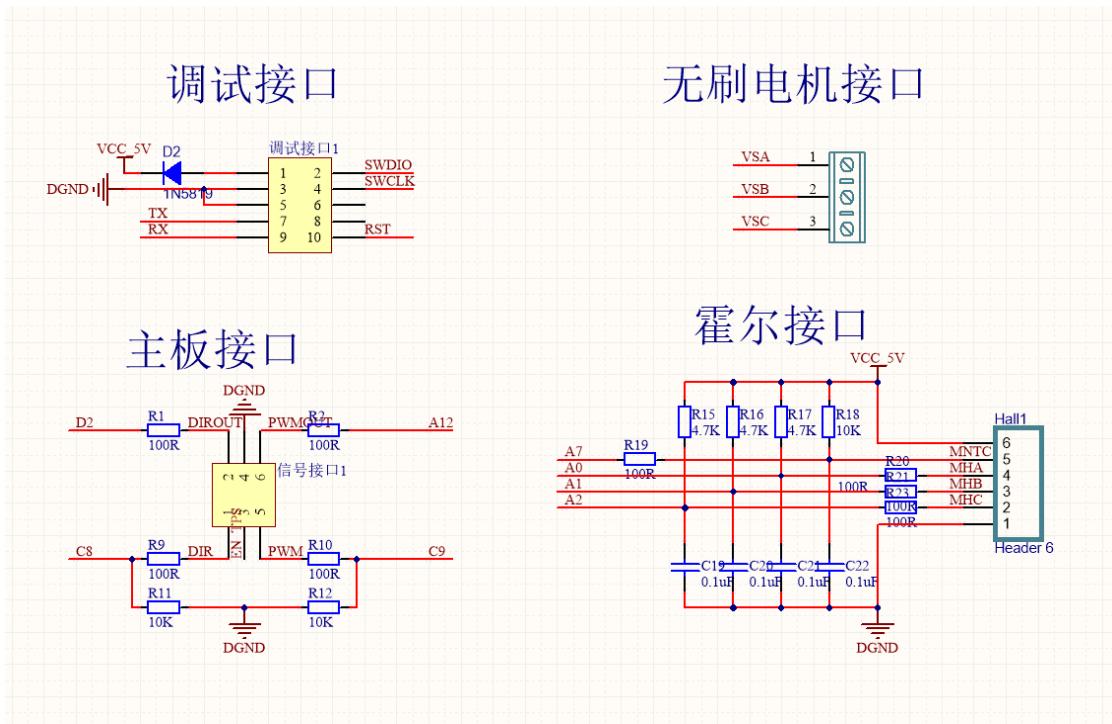


图 1.2.11 接口部分原理图

LED 指示灯用于指示驱动与电机的运行状态，其中使能 LED 灯，打开使能开关时亮起；运行 LED 灯，当无刷电机运行时亮起；错误指示 LED 灯，当无刷电机堵转时此灯亮起。使能开关连接 MCU 的引脚，用于控制电机是否工作。既可通过驱动板上的开关来控制，也可以通过遥控模块远程控制（通过跳线帽切换）。原理图如下：

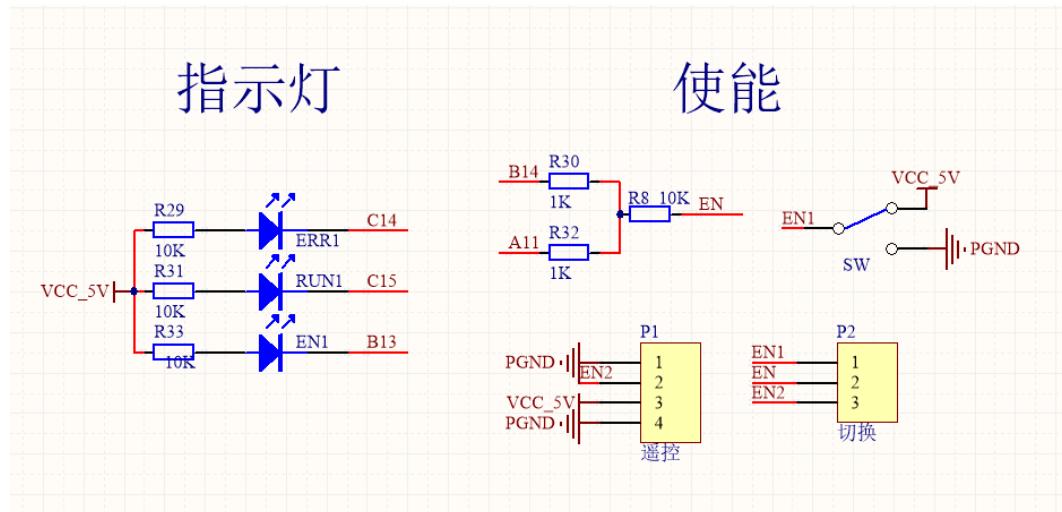


图 1.2.12 驱动部分 LED 模块原理图

### 1.2.9 电路布局

主板上主要有 MM32F3277 单片机接口、电源管理、人机交互模块和传感器接口。采用电源地与数字地分区的原则，通过磁珠连接分别铺铜。

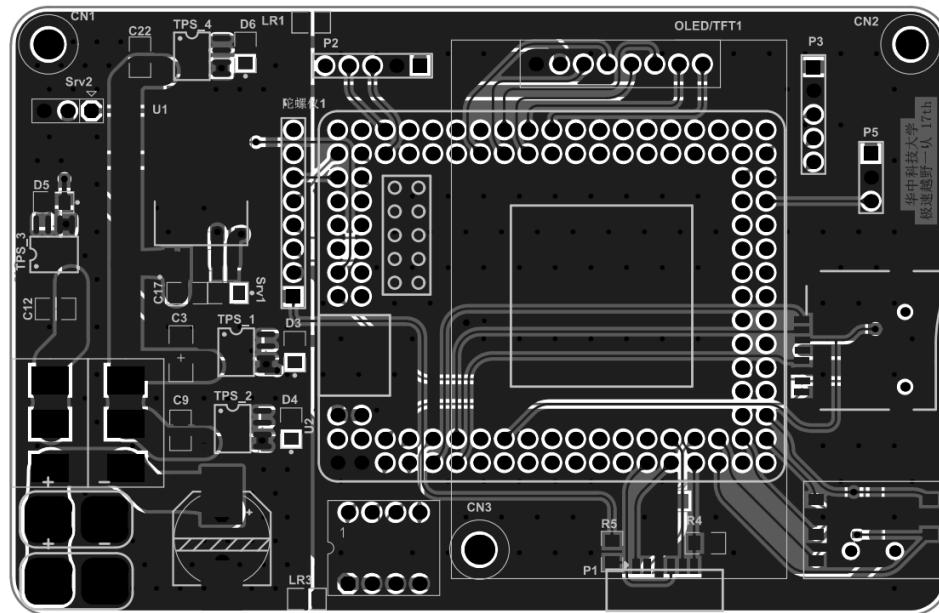


图 1.2.13 主板电路布局

驱动板为了满足电流需求，在大电流走线处开窗上。

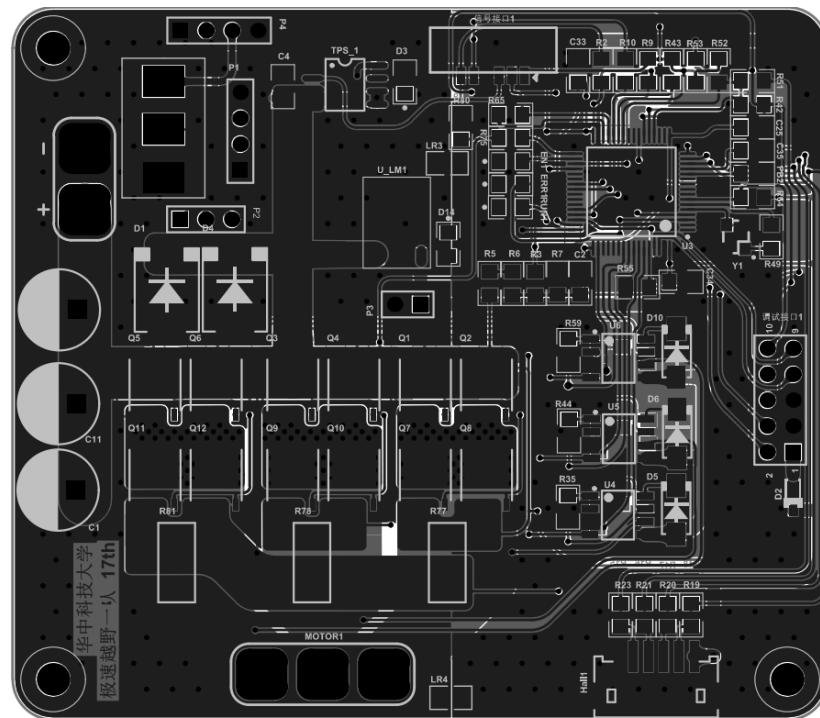


图 1.2.14 驱动板 PCB 布局情况

## 1.3 智能车控制算法简介

### 1.3.1 舵机角度的控制

在控制算法上，舵机角度控制使用的位置式 PD 控制。当某时刻赛车获得当前的 GPS 信息时，结合下一个目标点的坐标，可以计算出车辆的目标航向  $A_0$ ，此时再利用 GPS 返回的航向  $A_1$ ，便可以获得一个赛车与目标点的航向误差。将误差输入到 PD 控制器当中，便可以执行控制算法了。

由于 GPS 刷新的频率很低，只有 10Hz，如果仅仅利用 GPS 控制舵机，车辆转向的灵敏程度会大幅度下降。在这 100ms 的控制中，陀螺仪的作用就凸显出来了，陀螺仪每隔 2ms 便可以进行角度累计计算，通过陀螺仪角速度积分，我们可以获得角度的改变量，进而获得一个新的航向误差量用于控制 PD 调节。添加了陀螺仪之后，赛车的角度控制更加的灵活了，使用陀螺仪可以减少角度调节的误差。

### 1.3.2 导航点的切换

导航点的切换主要依靠离下一个点的距离以及到下个点的航向与前后导航点的差值。

如下图所示，当在直道区域与出弯道区域时，如果角度差值大于  $30^\circ$  或者与下一个点的距离小于 1.5m 时，则获得切换信号。在其他区域时，角度差值大于  $45^\circ$  或者与下一个点的距离小于 1.5m 时，则获得切换信号。为了避免在直道区产生过大的振动，所以临界角度会更小。

```
//求目标航向和两点航向的差值绝对值(区分三个航向)
Use_Data.change_angle_delta = delta_heading_angle(Use_Data.angle_next_heading_now, Use_Data.TwoDotsDirec);

if(area_tag == DIRECT_AREA || area_tag == ROUND_OUT_AREA )
{
    if ( Use_Data.change_angle_delta > 30.0    || Use_Data.distan_next_point_now < 1.50)
    {
        change_point = 1;
    }
}
else if ( Use_Data.change_angle_delta > 45.0    || Use_Data.distan_next_point_now < 1.50 )
{
    change_point = 1;
}
```

图 1.3.1 导航点切换代码

### 1.3.3 无刷电机驱动

无刷驱动的方式使用的是六部换相法近似产生交流电压信号驱动有感无刷电机，利用无刷电机内置的霍尔模块，来获取无刷电机转子的位置。然后根据转子的位置，按照一定的方式从 A,B,C 三相输入电压，从而实现无刷电机的控制。

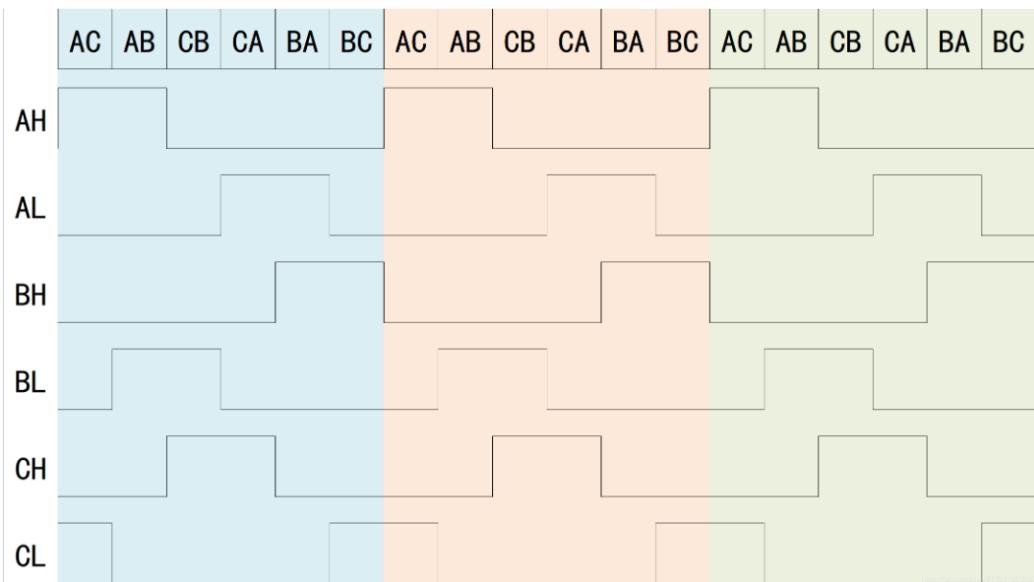


图 1.3.2 无刷驱动三相电压控制图

### 1.3.4 自动路径产生法

采用传统的手推车采集 GPS 数据点容易存在一定的数据误差，比如形状上不规则、单个导航点离路径偏差太大的问题。所以基于此，我们尝试了利用计算机解算，四点生成路径的方法。

在这个方法中，我们把操场看做两个半圆加两个直道组成的赛道。直道部分可以直接按照直道起点和直道终点按照距离等距离划分，这样获得的路径将会是一条标准的直线。弯道部分利用起点和终点，按照每段的距离，近似得出每段的角度，然后按照一定的角度改变量获得半圆形的路径。

通过这种方法生成的路径，不仅在路径上更加的标准规范，减少路径上带来的误差，同时标准化的数据也有利于车辆参数的调节，提升了算法的鲁棒性。

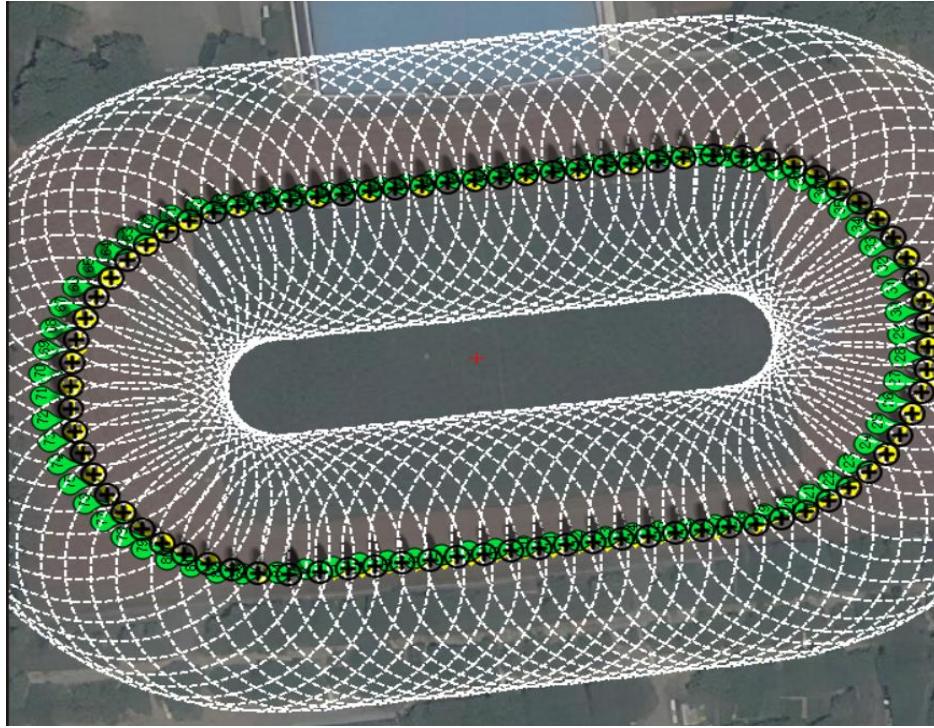


图 1.3.3 自动路径生成法生成的路径

	命令	Delay			Lat	Long	Alt	Frame	删除	向上	向下	坡度	Angle	距离	方 位
1	WAYPOINT	▼	0	0	0	0	30.484454	114.303869	0	Rela...	X	▲	0.0	0.0	1... 103
2	WAYPOINT	▼	0	0	0	0	30.484407	114.303876	0	Rela...	X	▲	0.0	0.0	5.3 173
3	WAYPOINT	▼	0	0	0	0	30.484336	114.303882	0	Rela...	X	▲	0.0	0.0	5.3 174
4	WAYPOINT	▼	0	0	0	0	30.484313	114.303889	0	Rela...	X	▲	0.0	0.0	5.3 173
5	WAYPOINT	▼	0	0	0	0	30.484266	114.303896	0	Rela...	X	▲	0.0	0.0	5.3 173
6	WAYPOINT	▼	0	0	0	0	30.484219	114.303902	0	Rela...	X	▲	0.0	0.0	5.3 174
7	WAYPOINT	▼	0	0	0	0	30.484173	114.303909	0	Rela...	X	▲	0.0	0.0	5.2 173
8	WAYPOINT	▼	0	0	0	0	30.484126	114.303916	0	Rela...	X	▲	0.0	0.0	5.3 173
9	WAYPOINT	▼	0	0	0	0	30.484079	114.303922	0	Rela...	X	▲	0.0	0.0	5.3 174
10	WAYPOINT	▼	0	0	0	0	30.484032	114.303929	0	Rela...	X	▲	0.0	0.0	5.3 173
11	WAYPOINT	▼	0	0	0	0	30.483985	114.303936	0	Rela...	X	▲	0.0	0.0	5.3 173
12	WAYPOINT	▼	0	0	0	0	30.483938	114.303942	0	Rela...	X	▲	0.0	0.0	5.3 174
13	WAYPOINT	▼	0	0	0	0	30.483891	114.303949	0	Rela...	X	▲	0.0	0.0	5.3 173
14	WAYPOINT	▼	0	0	0	0	30.483845	114.303956	0	Rela...	X	▲	0.0	0.0	5.2 173
15	WAYPOINT	▼	0	0	0	0	30.483798	114.303962	0	Rela...	X	▲	0.0	0.0	5.3 174
16	WAYPOINT	▼	0	0	0	0	30.483751	114.303969	0	Rela...	X	▲	0.0	0.0	5.3 173
17	WAYPOINT	▼	0	0	0	0	30.483704	114.303976	0	Rela...	X	▲	0.0	0.0	5.3 173
18	WAYPOINT	▼	0	0	0	0	30.48366	114.303983	0	Rela...	X	▲	0.0	0.0	4.9 172
19	WAYPOINT	▼	0	0	0	0	30.483617	114.303999	0	Rela...	X	▲	0.0	0.0	5.0 162
20	WAYPOINT	▼	0	0	0	0	30.483576	114.304042	0	Rela...	X	▲	0.0	0.0	5.0 156
21	WAYPOINT	▼	0	0	0	0	30.483541	114.304045	0	Rela...	X	▲	0.0	0.0	4.6 148
22	WAYPOINT	▼	0	0	0	0	30.483506	114.304076	0	Rela...	X	▲	0.0	0.0	4.9 143
23	WAYPOINT	▼	0	0	0	0	30.483472	114.304116	0	Rela...	X	▲	0.0	0.0	5.4 135
24	WAYPOINT	▼	0	0	0	0	30.483445	114.304158	0	Rela...	X	▲	0.0	0.0	5.0 127
25	WAYPOINT	▼	0	0	0	0	30.483423	114.304203	0	Rela...	X	▲	0.0	0.0	5.0 120

图 1.3.4 部分经纬度数据

通过专业的无人机软件 mission planner 观察，可得产生的路径在基本符合操场跑道的形状，而且点与点之间的距离都能够控制的很好。人工采集数据点因为要采集数十个点，并且在炎热的天气中抱着车辆采集数据难免会产生一定的错误。

---

## 1.4 物联网操作系统概述

物联网的定义是通过一些信息传感设备，按约定的协议，把任何物品与互联网相连接，进行信息交换和通信，以实现对物品的一些功能一种网络。21世纪互联网的发展，越来越多的物联网系统被应用到物流供应管理、灾害预警管理、环境监测、军事、医疗等领域，物联网系统也在随着时代不断地发展。物联网系统本身是为了更好地服务物联网，使得其中的层次结合更加紧密，数据共享更加通畅，从而提升物联网的生产效率。

物联网操作系统具备一些常见的功能，例如：设备资源管理功能、提供统一的编程接口、培育物联网生态环境、降低物联网应用开发成本、物联网统一管理等功能。一般来说，一个物联网操作系统包括内核、通信支持、外围组件、集成开发环境等内容。

## 1.5 实时操作系统概述

实时操作系统能够根据外界的事件与数据，能够接受并且以足够快的速度进行处理并给出响应，调度一切可以利用的资源完成实时任务，并控制所有实时任务协调一致运行。实时操作系统有硬实时和软实时之分，硬实时要求在规定的时间内必须完成操作，这是在操作系统设计时保证的；软实时则只要按照任务的优先级，尽可能快地完成操作即可。通常，实时操作系统会是二者的结合，实时操作系统的核心在于保障系统的实时性。

## 1.6 后续内容的框架安排

后续部分会介绍：

- RT-Thread 操作系统的内容
- 以及在本次智能车竞赛上面的应用，包括 RT-Thread 操作系统的移植、初始化的优化、线程分配与管理、信号量、软件定时器与 Env 的工具使用方法
- 还会介绍参与 RT-Thread 社区建设以及代码贡献的方法；
- 介绍 RT-Thread 与裸机开发的对比以及凸显的优势；
- 针对使用的总结以及心得体会
- 部分程序源代码

## 第二章 RT-Thread 物联网操作系统介绍

### 2.1 RT-Thread 简介

RT-Thread 又称为 Real Time Thread，是成千上万个 RTOS 中的一种。第一个内核版本是于 2006 年初发布的 0.1 版本。因为 RTOS 中的任务更类似于通用操作系统中的线程，并且这个系统支持基于优先级的抢占式任务调度算法，调度器的时间复杂度是  $O(1)$ ，所以把它命名为 RT-Thread，即实时线程。

经过 16 年的发展，RT-Thread 被广泛应用于智能家居、智慧城市、安防、工控、穿戴等众多行业领域，累计装机量超过 8 亿台，GitHub 的 Star 数量超过 5k，嵌入式开源社区活跃度行业第一，是国人自主开发、国内最成熟稳定和装机量最大的开源 RTOS。

RT-Thread 支持较多的硬件平台，还拥有及其丰富的组件和软件包（包括文件系统、网络、IoT、AI、传感器等等），提供了便捷的开发环境和 IDE 工具，以及有众多技术文档、参考设计和活跃的开发者社区。相比于其他操作系统，RT-Thread 的主打特性是“小而美的物联网操作系统”。

“小”体现在 RT-Thread 的体积小，最小资源占用 1.2KB RAM 和 2.5KB flash。RT-Thread 可伸缩、易裁剪的特性，可以避免用户占用过多的资源。并且能够最大限度的降低系统功耗，针对不同的应用场景，采用自动功耗控制策略；同时该操作系统简单易用，架构清晰、调试方便。

“美”不单止代码质量和代码风格，还有 RT-Thread 的使用和开发体验：跨芯片平台支持所有的主流微控制器，解决设备碎片化问题；使用了实时操作系统内核；组件上十分的丰富，支持设备虚拟文件系统、设备管理器框架、低功耗管理框架、协议栈、图形库等一系列组件与功能；调试上方便，内置 Shell 调试工具便于实时监测内核信息；支持主流的编译工具，例如：Keil, IAR, GCC 等开发环境。

针对物联网的场景，RT-Thread 提供了众多的组件与软件包，比如 AT 组件、Wifi、蓝牙、AI 等等，以及一些安全层面的优化，这适合 RT-Thread 在物联网领域能够被广泛地使用。

### 2.2 RT-Thread 的版本与架构

#### 2.2.1 RT-Thread 标准版

也被称之为 RT-Thread 的全功能版本，主要由软件包、组件与服务层、硬实时内核组成。同时也具备低功耗、安全、通信协议支持的能力，功能完整。RT-Thread 与其他很多 RTOS 如 FreeRTOS、uC/OS 的主要区别之一是，它不仅仅是一个实时内核，还具备丰富的中间层组件，如下图所示。

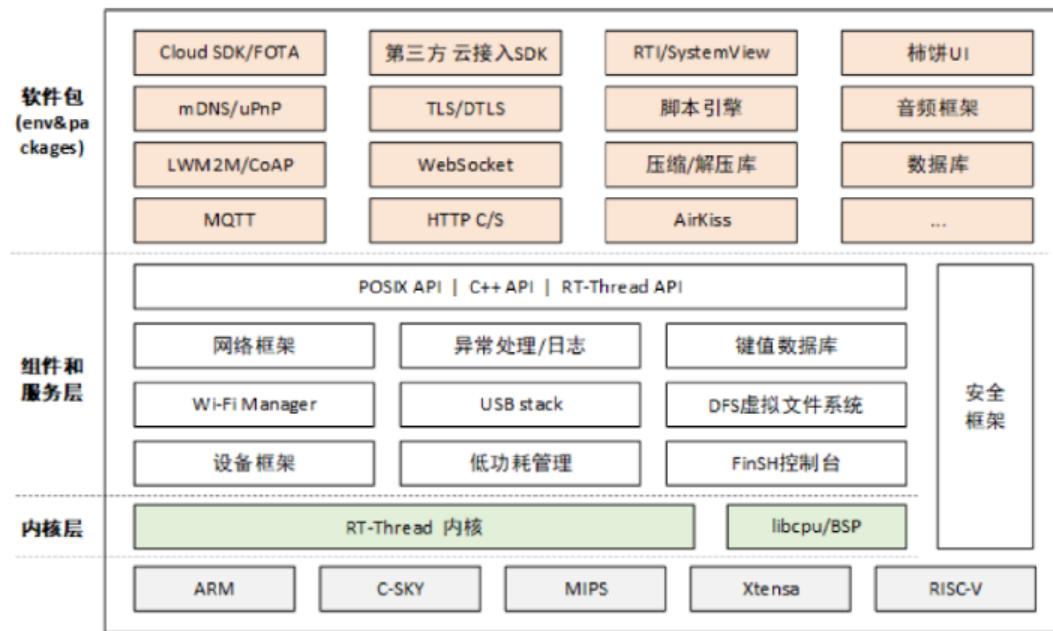


图 2.2.1 RT-Thread 中间层

在中间层，主要包括三个部分：

内核层是该操作系统的核心部分，包括内核系统中对象的实现，比如多线程及其调度、信号量、定时器等；libcpu/BSP（芯片移植相关文件 / 板级支持包）与硬件密切相关，由外设驱动和 CPU 移植构成。

组件与服务层：组件是基于 RT-Thread 内核之上的上层软件，例如虚拟文件系统、FinSH 命令行界面等。采用模块化设计，做到组件内部高内聚，组件之间低耦合。

软件包：RT-Thread 软件包运行于 RT-Thread 物联网操作系统平台上，面向不同应用领域的通用软件组件，由描述信息、源代码或库文件组成。RT-Thread 已经支持的软件包数量已经达到 400+，比如：Openmv, WebClient, mongoose、WebTerminal 等等。

## 2. 2. 2 RT-Thread Nano 版本

这是一个极简版本的硬实时内核，由 C 语言开发，采用面向对象编程的思维，代码风格良好，是一款可裁剪、抢占实时多任务的 RTOS。内存资源占用极小，功能包括任务处理、软件定时器、信号量、邮箱和实时调度等相对完整的实时操作系统特性。适用于大量使用的 32 位 ARM 入门级 MCU 的场合。



图

### 2.2.2 RT-Thread Nano 的软件框图

Nano 版本对于主流的 CPU 架构都支持, 比如 ARM 的 Cortex M0/ M3/ M4/ M7 等, 本次参赛所使用的芯片 MM32F3277G9P 便使用了 Cortex M3 架构。在功能上支持线程管理、时钟管理、中断管理、内存管理等功能。

### 2.2.3 RT-Thread Smart 版本

RT-Thread Smart (简称 rt-smart) 是基于 RT-Thread 操作系统衍生的新分支, 面向带 MMU, 中高端应用的芯片, 例如 ARM Cortex-A 系列芯片, MIPS 芯片, 带 MMU 的 RISC-V 芯片等。rt-smart 在 RT-Thread 操作系统的基础上启用独立、完整的进程方式, 同时以混合微内核模式执行。

### 2.2.4 RT-Thread Studio

RT-Thread Studio 是一款专业的 RT-Thread 开发工具, 通过简单易用的图形化配置系统以及丰富的软件包和组件资源, 让物联网开发变得简单和高效。功能主要包括工程创建和管理, 代码编辑, SDK 管理, RT-Thread 配置, 构建配置, 调试配置, 程序下载和调试等功能, 结合图形化配置系统以及软件包和组件资源, 减少重复工作, 提高开发效率。总而言之这是一款功能强大的物联网系统开发工具。

在本次比赛任务中原本想借助 RT-Thread Studio 作为开发工具, 但是 RT-Thread Studio 在创建工程时暂时还不支持自己所需要的使用的芯片型号 (MM32F3277G9P) 的资源包 (创建工程时选择芯片型号, 灵动系列的芯片暂时只支持 MM32L373PF), 所以最后还是不得不选择 KEIL5 作为开发工具。所以在这里也是希望官方能够针对 RT-Thread Studio 进行一定的更新, 希望在不久远的将来能够支持更多芯片的开发。

## 第三章 RT-Thread 操作系统的具体应用

### 3.1 RT-Thread 的移植

#### 3.1.1 移植的硬件环境

本次比赛我们组所采用的的主控芯片为 MM32F3277G9P 与 MM32SPIN27，由灵动微电子研发生产。详细介绍请参照前文中电路设计的主控芯片介绍单元。因为 MM32SPIN27 芯片控制的无刷驱动板直到比赛前一周不到的时间内才稳定投入使用，由于较大的备赛压力，来不及对无刷驱动环境进行操作系统移植，所以 RT-Thread 操作系统的应用主要集中在主板主控芯片上。后续如果有时间有机会，我们小组将会尝试在无刷驱动上使用 RT-Thread 操作系统。

#### 3.1.2 移植的软件环境

本次采用的 RT-Thread 的源代码版本是最新的 V4.10 版本，移植到逐飞 MM32F3277 开源库进行编程的。这两个开源库，我们可以在 gitee 上下载，而且逐飞也已经推出了适配 RT-Thread 的开源库。虽然逐飞科技已经在码云发布了适配 RT-Thread 的开源库，但是我们还是进行了一个学习与尝试。根据我们的功能需求，在移植的过程中主要使用了 bsp、components、include、libcpu、src 等这几个文件夹，具体的移植操作会在后文中给出。

#### 3.1.3 具体的移植过程

首先，我们能够在 RT-Thread 的官网与码云下载对应的源码与开源库，在官网我们能够进入下载页：

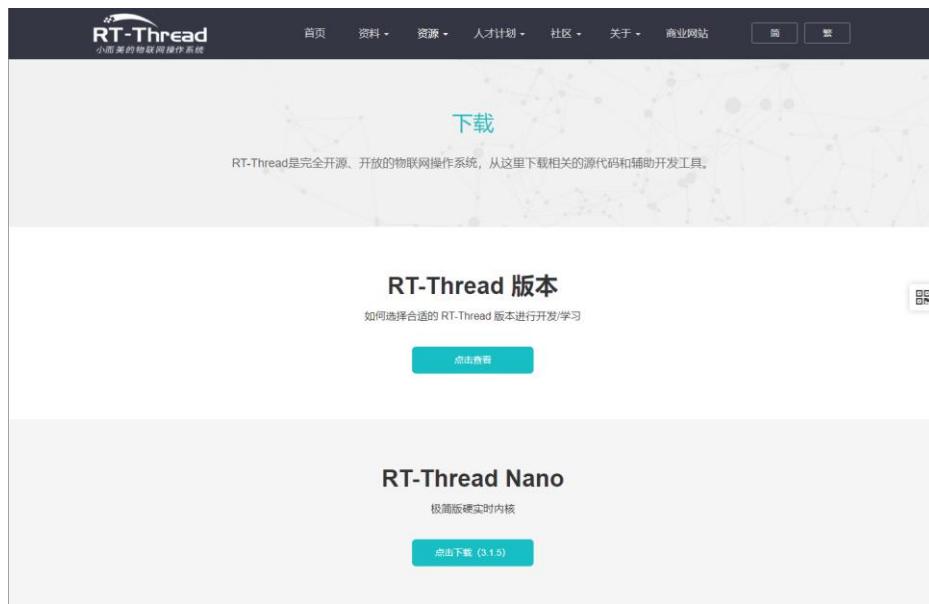


图 3.1.1 RT-Thread 官网首页

通过 gitee 链接，我们既能够下载逐飞开源库与 RT-Thread 最新版的源码，也能够阅读对应的版本说明文件。这里不推荐使用百度网盘下载，因为官网在百度网盘中给出的 Rt-Thread 的源码版本较为老旧，而且该版本中的 bsp 缺

乏 mm32f3277 系列的 rtconfig.h 文件。

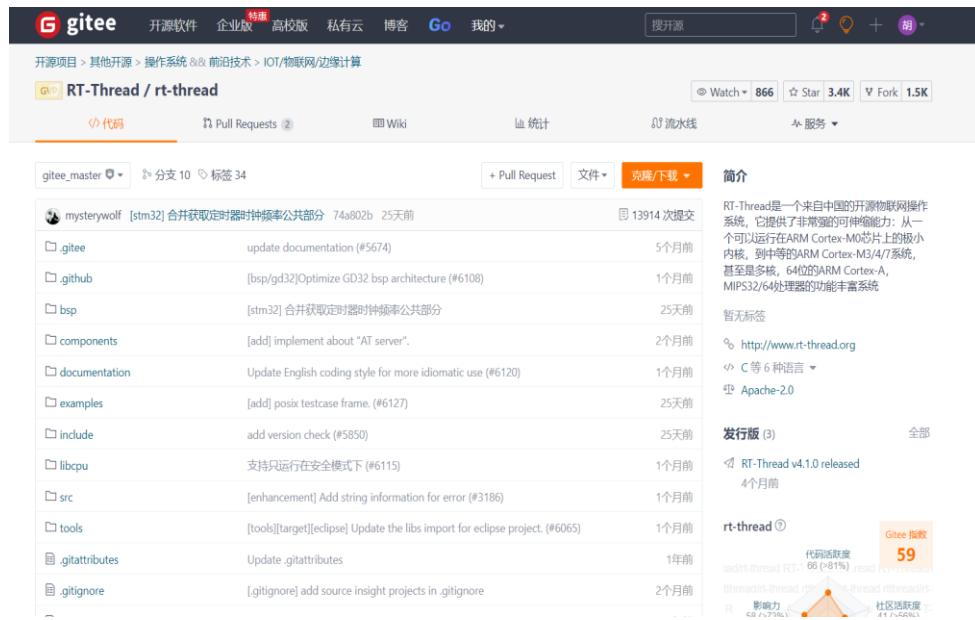


图 3.1.2 RT-Thread 码云下载页

在准备好开源库与原码之后，就可以把源码添加到逐飞的开源库当中了。打开逐飞开源库的工程文件，这里我们所使用的软件开发系统是 KEIL5，打开 MDK 文件，首先我们在资源管理器的侧边栏添加操作系统的原码文件夹：

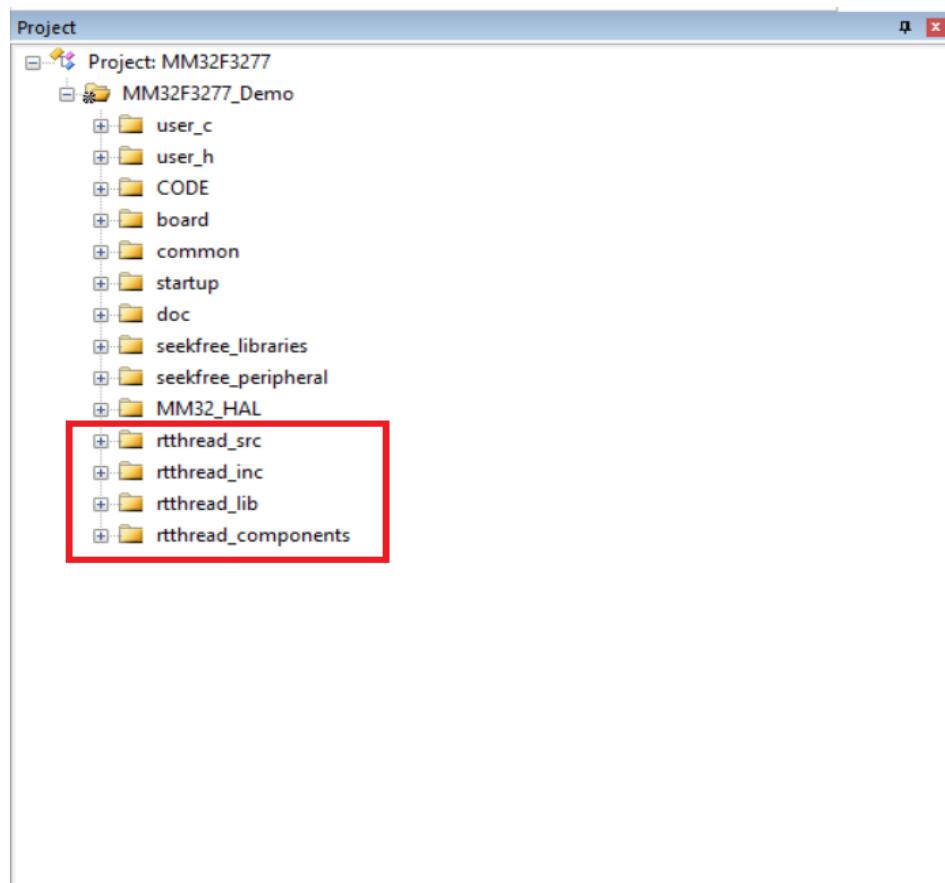


图 3.1.3 新建文件夹

这四个文件夹分别是 rtthread\_src、rtthread\_inc、rtthread\_lib、rtthread\_components。第一个文件夹 rtthread\_src 用于存放内核源码文件，对应的也是下载下来的 RT-Thread 原码文件中的 src 文件夹；第二个文件夹 rtthread\_inc 用于保存头文件，也就是源码文件中的 include 文件夹（这两个文件夹的东西全部保留）；第三个文件夹是 rtthread\_lib 用于保存处理器接口文件，这里针对我们芯片的种类，我们保留了 context\_rvds.S 与 cpuport.c 这两个文件；第四个文件夹 rtthread\_components 用于保存组件内容，在这里我们只保存了 finsh 调试组件用于调试。

在 user\_h 这个文件中，需要根据自己的芯片型号，添加对应的 rtconfig.h 文件。我们使用了 mm32f327x 这个文件夹，并把对应的头文件加入了 user\_h 中。



图 3.1.4 添加头文件

现在，需要把刚下添加进来的文件的头文件路径给添加进来，如下图所示：

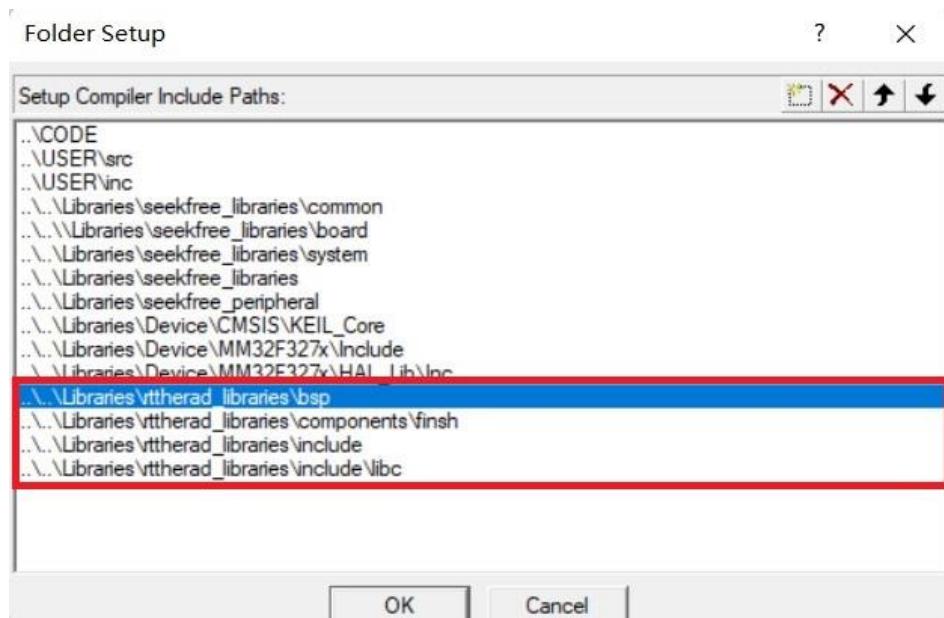


图 3.1.5 添加头文件的路径

接下来，根据自己的芯片内核，添加汇编的头文件路径，如下图所示：

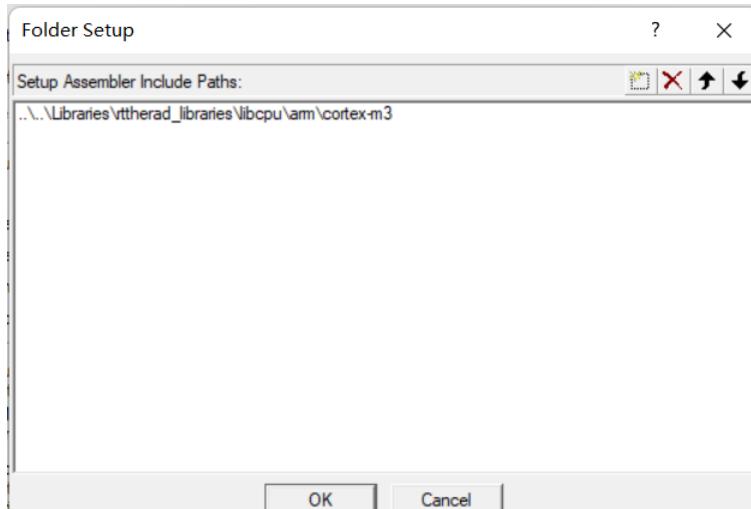


图 3.1.6 添加汇编的头文件路径

完成了文件配置之后，我们需要对原来开源库的一些代码做出修改，需要修改的是 board.c 文件。

对于 void board\_init (bool debug\_enable)这个函数，应当修改为：

```

99 // @brief 核心板初始化
100 // @param debug_enable 是否开启默认 debug 输出 默认 UART1
101 // @return void
102 // Sample usage:      board_init(TRUE);
103 //
104 void board_init (bool debug_enable)
105 {
106     NVIC_SetPriorityGrouping(4);
107     // 设置中断优先级不分组
108     if(debug_enable)
109     {
110         uart_init(DEBUG_UART, DEBUG_UART_BAUD, DEBUG_UART_TX, DEBUG_UART_RX);
111         // 默认初始化 UART1 用以支持 printf 输出
112     }
113     uart_rx_irq(DEBUG_UART, 1);
114 }
115
116

```

图 3.1.7 board\_init 函数修改

接下来，由于 DEBUG 选择的是 UART1 这个串口，则需要配置 void UART1\_IRQHandler(void)这个函数，内容修改如下图所示。这个可以用于 finsh 中的模块进行调试，finsh 组件之所以能接受串口的数据，是因为在串口中断中发送了邮件，finsh 线程接收到了邮件。

```

117 void UART1_IRQHandler(void)
118 {
119     uint8 dat;
120     rt_interrupt_enter();
121     if(UART1->ISR & UART_ISR_TX_INTF)    // 串口发送缓冲空中断
122     {
123         UART1->ICR |= UART_ICR_TXICLR;    // 清除中断标志位
124     }
125     if(UART1->ISR & UART_ISR_RX_INTF)    // 串口接收缓冲中断
126     {
127         uart_getchar(DEBUG_UART, &dat);
128         rt_mb_send(uart_mb, dat);          // 发送邮件
129         UART1->ICR |= UART_ICR_RXICLR;    // 清除中断标志位
130     }
131     rt_interrupt_leave();
132 }

```

图 3.1.8 UART1 中断函数修改

---

紧接着，添加操作台的输出与输入函数内容，如下图所示：

```
76 void rt_hw_console_output(const char *str)
77 {
78     while(RT_NULL != *str)
79     {
80         if('\n' == *str)
81         {
82             uart_putchar(DEBUG_UART, '\r');
83         }
84         uart_putchar(DEBUG_UART, *str++);
85     }
86 }
87
88
89
90 char rt_hw_console_getchar(void)
91 {
92     uint32 dat;
93     //等待邮件
94     rt_mb_recv(uart_mb, &dat, RT_WAITING_FOREVER);
95     //uart_getchar(DEBUG_UART, &dat);
96     return (char)dat;
97 }
98
```

图 3.1.9 操作台函数的添加

在 board.c 中，也同样需要时钟的配置，既需要配置系统时钟，也需要对操作系统的时钟节拍进行配置。在这个 c 文件中，需要添加一个系统时钟中断函数，也就是 void SysTick\_Handler(void)，在这个函数中调用 rt\_tick\_increase() 函数便可，如图所示：

```
66 void SysTick_Handler(void)
67 {
68     rt_interrupt_enter();
69
70     rt_tick_increase();
71
72
73     rt_interrupt_leave();
74 }
75
```

图 3.1.10 系统时钟中断

初始化部分的内容也需要添加，rt\_hw\_board\_init 初始化函数先配置了系统时钟滴答，然后也包含了原来的主板初始化函数 board\_init();然后针对使用的组件与是否使用堆来给予初始化，最后创建了 UART 邮箱用于通信，内容如下图所示：

```

25   extern uint32_t SystemCoreClock;
26
27   rt_mailbox_t uart_mb;
28
29   static uint32_t systick_config(rt_uint32_t ticks)
30 {
31     if ((ticks - 1) > 0xFFFFFFF)
32     {
33       return 1;
34     }
35
36     SysTick->LOAD = ticks - 1;
37     nvic_init(SysTick_IRQn, 7, ENABLE);
38     SysTick->VAL = 0;
39     SysTick->CTRL = 0x07;
40
41     return 0;
42   }
43
44   void rt_hw_board_init()
45 {
46   systick_config(SystemCoreClock / RT_TICK_PER_SECOND);
47
48   board_init();
49   /* Call components board initial (use INIT_BOARD_EXPORT()) */
50 #ifdef RT_USING_COMPONENTS_INIT
51   rt_components_board_init();
52#endif
53
54 #if defined(RT_USING_HEAP)
55   rt_system_heap_init((void *)HEAP_BEGIN, (void *)HEAP_END);
56#endif
57
58   uart_mb = rt_mb_create("uart_mb", 10, RT_IPC_FLAG_FIFO);
59 }

```

图 3.1.11 RT-Thread 初始化函数

接下来配置的是底层接口的滴答定时器，打开 zf\_systick.c 文件。找到 void systick\_delay\_ms (uint32 time)这个函数，注释掉之前的内容，我们把里面的代码修改为只保留“rt\_thread\_mdelay(time);”这一行。然后再把不开放给用户使用的一些函数比如：void systick\_timing(uint32 time)、void systick\_start(void) 给注释掉。同时也添加了软件延时函数 void systick\_delay (uint32 time)，但这个延时函数并不推荐使用，容易造成 CPU 资源的浪费。

最后，在中断函数内部要加上 rt\_interrupt\_leave();这个函数，表示中断结束。

### 3.2 初始化的优化

通过查阅 RT-Thread 官方给出的文档，了解到了 RT-Thread 具备自动初始化机制。自动初始化机制指的是初始化函数只需要函数定义处通过宏定义的方式进行申明，这样就会在系统启动过程中被执行。

在设计智能车的程序部分时，需要很多外设进行初始化，这样的初始化少则几个，多则十几个，通过自动初始化设计，可以简化代码设计，提升了程序的可读性。如图，这是裸机开发时的初始化部分代码。

```

37 int main (void)
38 {
39
40     my_gps_init();
41     icm20602_init_spi();
42     lcd_init();
43     motor_init();
44     seekfree_wireless_init();
45     FlashReadParams();
46     duoji = pwm_mid_angle;
47     button_init();
48     servo_motor_init();
49     //初始化四个关键点
50     generating_route_init();
51     //g_send = 2;
52     lcd_clear(BLACK);
53     delay();
54     lcd_clear(GRAY);
55     delay();
56     lcd_clear(WHITE);
57     //定时器中断
58     tim_interrupt_init(TIM_5, 1000, 1);
59     //每个点的间隔(直道路)
60     distance_each_point_direct = 5.0;
61     //每个点的间隔(弯道路)
62     distance_each_point_round = 5.0;
63
64     for (uint8 i = 0; i < 8; i++)
65         lcd_drawpoint(x_cursor + i, y_cursor, BLACK);
66
67     x_cursor = 96;
68     y_cursor = 32;

```

图 3.2.1 裸机开发的初始化

在修改之前，需要了解 RT-Thread 的启动流程，启动流程图如下所示。

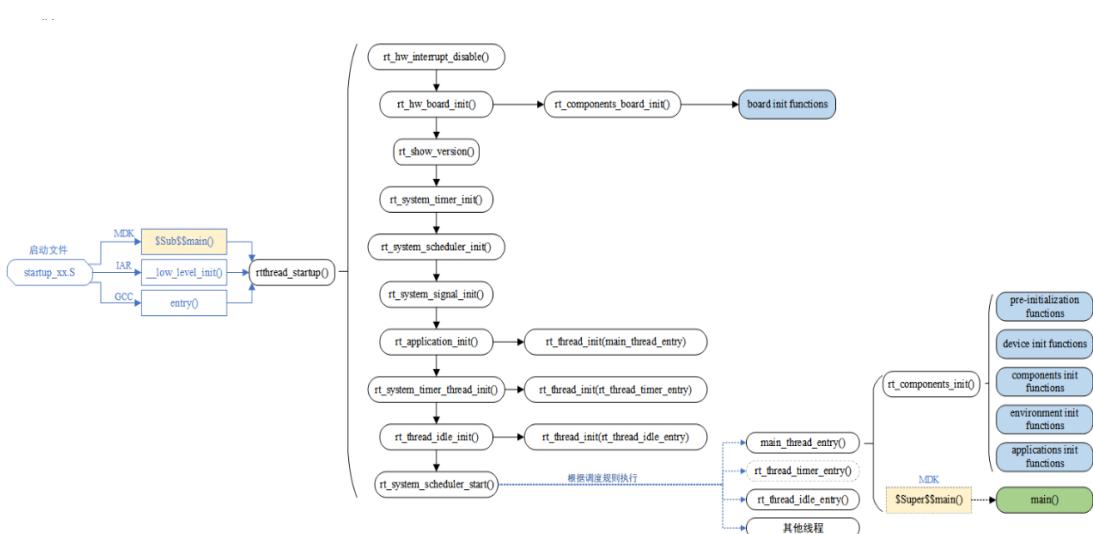


图 3.2.2 启动流程图

从图中可以看出，先是经过了 startup\_xx.S 这个启动文件，然后进入 RT-Thread 的启动函数 rtthread\_startup()，最后进入用户入口函数 main()。因为使用了 MDK 的扩展功能\$Sub\$\$ 和 \$Super\$\$，使得在 main()线程执行前

提前执行一些功能，这样用户便可以不必理会之前的初始化操作。

启动代码可以大致分成四个主要功能：(1) 硬件初始化，(2) 系统内核对象初始化，(3) 创建 main 线程，并对其中的各类模块依次进行初始化 (4) 初始化定时器线程、空闲线程，启用调度器。

RT-Thread 的自启动机制使用 RTI 符号段，把函数指针放到符号段当中，系统启动时会遍历函数表，并调用函数，达到了初始化的目的。实现自初始化的有 6 个宏接口，详细的描述如下表所示：

顺序	宏接口	描述
1	INIT_BOARD_EXPORT(fn)	早期初始化，先于调度器启动
2	INIT_PREV_EXPORT(fn)	用于纯软件初始化
3	INIT_DEVICE_EXPORT(fn)	外设驱动初始化
4	INIT_COMPONENT_EXPORT(fn)	组件初始化
5	INIT_ENV_EXPORT(fn)	系统环境初始化，例如挂载 FAFTS 文件系统
6	INIT_APP_EXPORT(fn)	应用初始化，比如 GUI 应用

表 3.2.3 宏接口描述表

经过修改之后，初始化的代码如下图所示，可见代码更加的清晰了。

```

38 //initialization adaptation
39 int device_init(void)
40 {
41     my_gps_init();
42     icm20602_init_spi();
43     lcd_init();
44     motor_init();
45     seekfree_wireless_init();
46     button_init();
47     servo_motor_init();
48 }
49
50 int app_init(void)
51 {
52     FlashReadParams();
53     duoji = pwm_mid_angle;
54     //初始化四个关键点
55     generating_route_init();
56     lcd_clear(BLACK);
57     delay();
58     lcd_clear(GRAY);
59     delay();
60     lcd_clear(WHITE);
61     //定时器中断
62     tim_interrupt_init(TIM_5, 1000, 1);
63     for (uint8 i = 0; i < 8; i++)
64         lcd_drawpoint(x_cursor + i, y_cursor, BLACK);
65     x_cursor = 96;
66     y_cursor = 32;
67 }
68
69 INIT_DEVICE_EXPORT(device_init);
70 INIT_APP_EXPORT(app_init);

```

---

图 3.2.4 修改过后的初始化部分

```
72 int main (void)
73 {
74     while (1)
75     {
76         page_spin();
77
78         //拨码器2号损坏
79         sum[0] = gpio_get(SWITCH_1);
80         sum[1] = gpio_get(SWITCH_2);
81         sum[2] = gpio_get(SWITCH_3);
82         sum[3] = gpio_get(SWITCH_4);
83
84         if (write_flash_points == 1)
85         {
86             generating_route();
87             write_flash_points = 5;
88         }
89
90         if (send_order >= 1 && newgpsgot == false)
91         {
92             send_spin_vofa();
93             send_order = 0.0;
94         }
95         else
96         {
97             ;
98         }
99     }
100 }
```

图 3.2.5 修改过后的 main 线程

### 3.3 线程分配与管理

#### 3.3.1 线程

完成极速越野车模的运行控制，可以分为几个小任务来解决，比如：GPS 信号的采集、GPS 信号的解析、控制量的输出等。在 RT-Thread 中与这些小任务对应的程序实体便是线程。线程不仅仅描述任务运行环境，也描述了任务的优先级，重要的任务可以设置相对较高的优先级，反之不那么重要的任务可以设置低优先级。

在线程中的程序不能够陷入死循环操作，否则比它低优先级的线程都不能得到执行。在刚开始编程的时候，忽略了这个要点，导致了 GPS 解算功能不能够正常执行。所以线程中必须要有让出 CPU 使用权的操作，一般可以通过主动挂起或者调用延时实现。

#### 3.3.2 线程的创建以及各个线程的作用

根据比赛的任务需要，我们创建了若干个线程去执行相应的任务，下面是各个线程的任务简介。

##### (1)main 线程

这个线程会根据利用 UART 串口通信从 GPS 获取的报文，执行相应的数据解算与数据处理功能。

##### (2)gps\_deal 线程

这个线程放置了最核心的任务，也就是根据此时的解算完成的 GPS 相关的信息，控制舵机的角度与输入给无刷电机驱动板的 PWM 信号进而控制小车

速度。

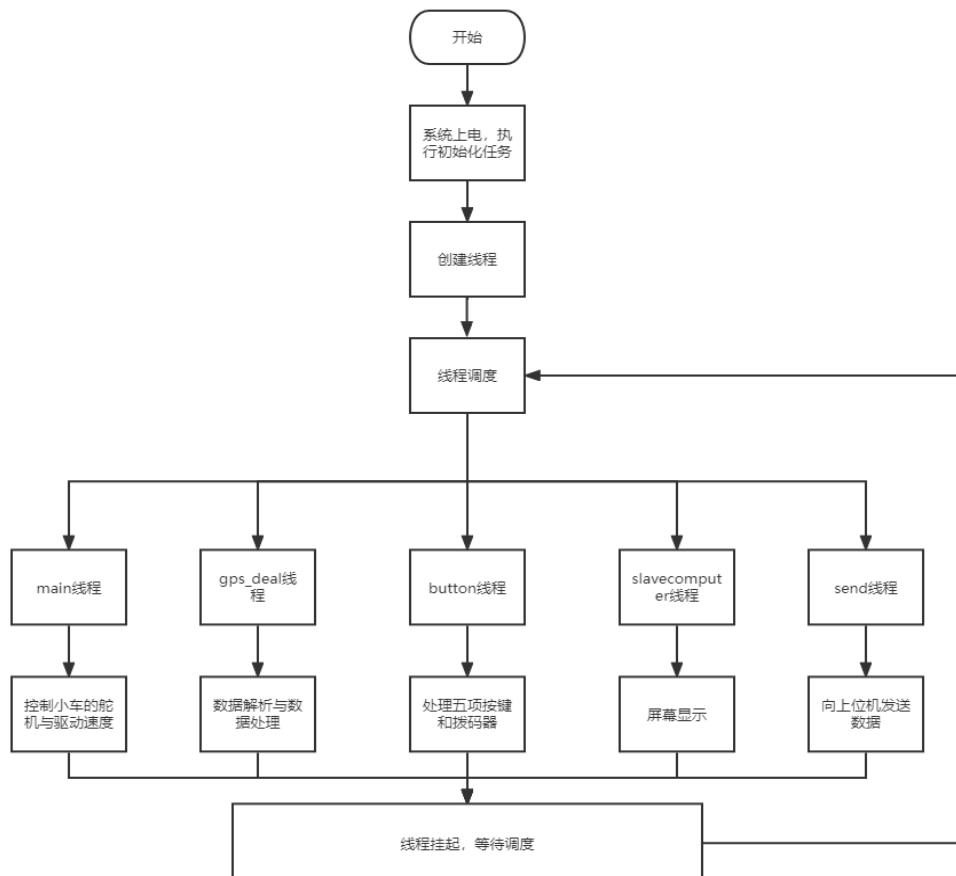
### (3) slavecomputer、button 线程

这两个线程通过 TFT180 显示屏，进行人机交互的线程，用于参数调整以及显示当前的 GPS 信息，同时也有按键扫描与拨码器扫描的功能。

### (4) send 线程

这个是负责利用无线串口发送数据的线程，当小车在运行的过程中，我们可以通过无线串口把数据发送到上位机，从而更好的了解小车运行情况，便于参数调整。

根据分配的线程，小车大致的执行流程如图所示。



### 3.3.3 线程的具体设计

现在根据每个线程的功能需求来安排线程创建的内容，线程创建的编写可以通过查阅官网的内核实验手册学习。线程创建使用的是 `rt_thread_create` 这一函数，创建动态线程，利用了系统的动态内存管理，使用起来比较的方便。

首先是 main 线程，main 线程中的任务安排的是 GPS 信息的获取与加工。

---

通过 UART2 这个串口，可以接收到 GPS 发送过来的报文。每次有新报文送达时，newgpsgot 这个 bool 型变量会转变为 true，从而可以着手从新报文中获得经纬度、航向、对低速度等一系列重要的数据。

对 GPS 信号进行具体的处理的是 gps\_deal 线程，该线程的优先级设计为 17，因为这是主要的功能部分，而且在路径规划时会调用一个较大的结构体数组，再加上 MM32F3277G9P 的 RAM 对于比赛任务较为充足，所以栈空间安排了 10KB。

如果车辆的 gps\_mode == 1，第一次进入 gps\_deal\_entry() 函数时首先会根据目前的经纬度坐标信息自动生成赛车的路径。路径的生成方式有两种，第一种是直接生成法，就是按照已经保存的 GPS 坐标信息从第 0 个开始到最后一个点依次规划路径，第二种则是会根据当前小车所处的位置，搜寻已保存的点中最近的几个点，按照逆时针航向再结合角度自动产生一圈的路径。在路径规划前，引入了赛道平移变量，发车手可以根据赛车运行时在跑道上所处的位置，对操场划分 8 个区域内的坐标点进行整体平移，以免赛车的路径过于切内或者过于靠外而出界犯规甚至与障碍物发生碰撞。

```
void gps_deal_entry()
{
    if (Save_Data.isUsefull == true && gps_mode == 1)
    {
        if (passed_sections == 0 && Save_Data.f_ground_speed < 1.8)
        {
            //舵机固定中值，防止车辆起步产生偏移
            pwm_duty_update(TIM_8, SERVO_PIN, pwm_mid_angle);
            //与第一个点的距离与航向计算，显示在tft屏幕上，为发车提供参考
            error_length_s_p = GPSDistance(Save_Data.f_lati, Save_Data.f_longi, Points_Use[1].lat_keypoint, Points_Use[1].lon_keypoint);
            error_direction = GPSBearingAngle(Save_Data.f_lati, Save_Data.f_longi, Points_Use[1].lat_keypoint, Points_Use[1].lon_keypoint);
        }

        if (analysis_finished == false && guiding_mode > 0)
        {
            //赛道位置微操
            micro_operating();
            //gps自动规划
            gps_auto_plannation();
            //赛道位置微操恢复
            micro_operating_recover();
            //路径规划结束
            analysis_finished = true;
        }
        else if (analysis_finished == false && guiding_mode <= 0)
        {
            //直接导航模式
            for (int i = 0; i < ALL_POINTS; i++)
            {
                Points_Use[i] = Points_Stored[i];
                analysis_finished = true;
            }
        }
    }
}
```

图 3.3.1 路径规划的函数调用代码

在完成了路径规划任务之后，便开始着手获取赛车下一个点和前一个点的位置信息与类型信息，便于速度决策与导航点切换决策。速度决策就是根据点的类型设定不同的速度，在速度决策上，我们把点分为四类：起跑加速区、直道区、弯道区、直道弯道切换区。利用前一个点与下一个导航点之间的航向，则可以作为导航点切换的依据之一，另一个依据便是当前所在点与下一个导航点的距离。

```

if (passed_sections <= run_sections && dot_read == false && analysis_finished == true)
{
    area_tag = Points_Use[passed_sections + 1].type_point;
    //速度规划
    if (passed_sections <= 1)
    {
        g_speed_set = speed_max_region[3];
        setspeed_to_pwm();
    }
    else if (area_tag == DIRECT_AREA || area_tag == ROUND_OUT_AREA)
    {
        g_speed_set = speed_max_region[0];
        setspeed_to_pwm();
    }
    else if (area_tag == ROUND_AREA)
    {
        g_speed_set = speed_max_region[1];
        setspeed_to_pwm();
    }
    else if (area_tag == ROUND_IN_AREA)
    {
        g_speed_set = speed_max_region[2];
        setspeed_to_pwm();
    }
    //获取上个已通过的点的信息
    Use_Data.lat_last = Points_Use[passed_sections].lat_keypoint;
    Use_Data.lon_last = Points_Use[passed_sections].lon_keypoint;
    //获取下一个目标点的信息
    Use_Data.lat_next = Points_Use[passed_sections + 1].lat_keypoint;
    Use_Data.lon_next = Points_Use[passed_sections + 1].lon_keypoint;
    //两点间的航向
    Use_Data.TwoDotsDirec = GPSBearingAngle(Use_Data.lat_last, Use_Data.lon_last,
                                             Use_Data.lat_next, Use_Data.lon_next);
    //因为刚切换完点，之前的陀螺仪累计值清零
    Data_Gyro.angle_sum_once = 0;
    //表明已经读取过信息，避免代码重复执行
    dot_read = true;
}

```

图 3.3.2 速度规划与坐标点切换

接下来的是当前点与下一个点的航向与距离计算，航向计算与距离计算可以利用已经编写完成的公式。当速度大于一定值时，就会进行导航点切换与航向误差的计算，当速度较低时，GPS 解算出的航向值误差较大，不应当用于舵机打角控制。导航点的切换有两个依据，第一个是距离判据，当目前点与下一个导航点的距离低于一个阈值便触发导航点切换，第二个则是角度，当前点与下一目标点的航向与之前提到的两点航向所成的角度大于一定值时，便会触发换点。因为当车速提高时，导航点的切换如果只是依靠距离则有可能造成切换失败使得车辆转向往回走，造成比赛任务失败。

```

//计算与下一个点的距离
Use_Data.distan_next_point_now = GPSDistance(Save_Data.f_lati, Save_Data.f_longi,
                                              Use_Data.lat_next, Use_Data.lon_next);
//计算与下一导航点的航向
Use_Data.angle_next_heading_now = GPSBearingAngle(Save_Data.f_lati, Save_Data.f_longi,
                                                   Use_Data.lat_next, Use_Data.lon_next);
//当速度大于一定值时,
if (Save_Data.f_ground_speed > 3.15)
{
    //第一次发车, 控制量复位
    if (start_car == 0)
    {
        start_car = 1;
        Data_Gyro.angle_sum_all = 0.0;
        Data_Gyro.angle_sum_area = 0.0;
        error_icm_last = 0.0;
    }
    //计算航向误差, 进行角度控制
    GpsAngleError();
    //导航点切换
    GpsDotSelection();
    //陀螺仪角度累计归零
    Data_Gyro.angle_sum_once = 0.0;
    error_icm_last = 0.0;
}
//如果导航点已经跑完, 便可以自动停车了
if (passed_sections >= run_sections && passed_sections > 0)
{
    g_speed_set = 0.0;
    setspeed_to_pwm();
}

```

图 3.3.3 计算距离、航向、误差

当 `gps_mode == -2` 时, 便会执行车辆复位的流程, 便于不断电发车, 节约比赛时间。

`Slavecomputer` 线程负责屏幕显示, 优先级设置为 31, 该任务的优先级为最低。首先屏幕的显示需要消耗大量的时间, 这样设置可以避免对其他任务产生的影响。单独设置为一个低优先级的线程, 和裸机开发直接放在 `main()` 函数的 `while(1)` 循环相比, 能更好的根据优先级安排好任务。

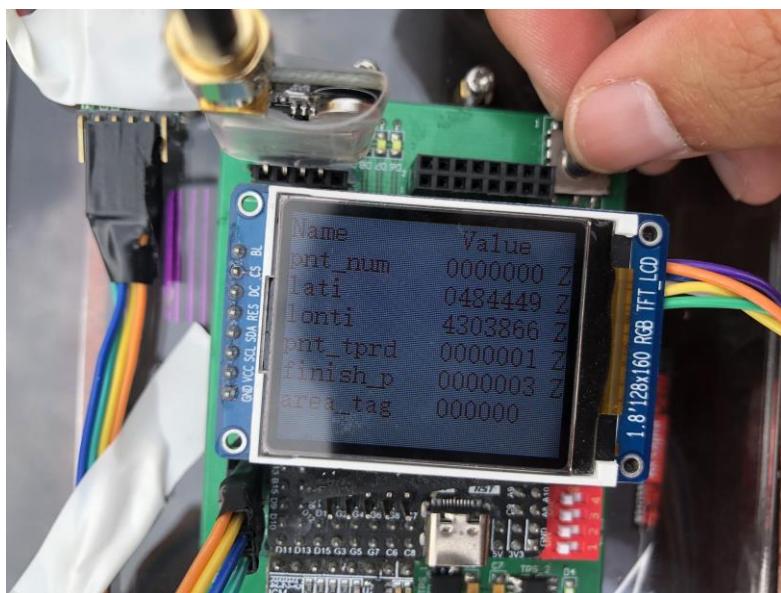


图 3.3.4 比赛前利用 `slavecomputer` 修改参数

RT-Thread 的线程概念就完美解决了这个问题。把不同的任务放在不同的线程，根据任务需求分配好相应的优先级。高优先级任务挂起的时候，可以很好的利用处理器的空闲资源来处理低优先级任务，低优先级任务不会影响高优先级任务的执行。

与之配合的是 button 线程，这是利用定时器创建的周期性按键扫描的线程。定时器设置的入口函数是 button\_control()，定时的周期为 15ms，过长的周期会使得按键的相应延迟非常明显，时间过短会导致按键不能消除抖动。该函数读取了五向按键和拨码器的数值后，根据一定的按键组合，配合屏幕显示，可执行参数修改、FLASH 写入等操作。

Send 线程的任务便是通过无线串口进行数据发送，这个线程的优先级设置为 21，优先级不高一般对主要任务影响不大。

### 3.4 信号量

#### 3.4.1 信号量简介

信号量是一种轻型的用于解决线程间同步问题的内核对象，每个信号量对象都有一个信号量值与线程等待队列，信号量的值对应了信号量对象的实例数目与资源数目。当信号量实例数目为 0 时，再申请该信号量的线程就会被挂起在该信号量的等待队列上，等待可用的信号量实例（资源）。

信号量的管理需要通过一个结构体 struct rt\_semaphore 来实现，内部包含一个结构体继承自 ipc\_object 类、一个信号量的值，其最大值为 65535。

这次比赛信号量的管理操作使用了三个操作：创建信号量、获取信号量、释放信号量。

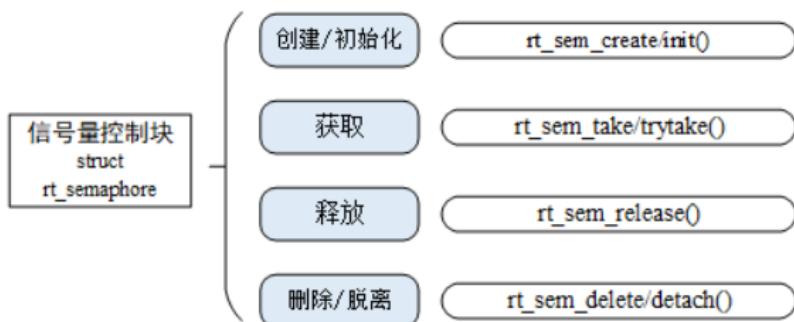


图 3.4.1 信号量控制结构

信号量的创建使用的函数为：

```
rt_sem_t rt_sem_create(const char* name,  
                       rt_uint32_t value, rt_uint8_t flag);
```

其中输入的参数 name 表示信号量名称，value 表示信号量的初始值，flag 表示信号量标志，可以取得 RT\_IPC\_FLAG\_FIFO 或 RT\_IPC\_FLAG\_PRIO 这两个值，这个表示了信号量不可用时多个线程等待的排队方式。

---

获取信号量可以使用这个函数：

```
rt_err_t rt_sem_take(rt_sem_t sem, rt_int32_t time);
```

其中第一个参数表示信号量对象的句柄，time 表示指定的等待时间。调用该函数时，信号量的值如果等于 0，那么说明当前信号量资源不可用。申请信号量的线程会在指定的 time 下执行相应的操作，当 time 大于等于 0 时，会按照 time 对应的时间进行等待，当时间为负数时，线程会永远在信号量上进行挂起，直到信号量在其他的线程或者中断中被释放。信号量获取的函数应当优先在线程中使用，如果在中断中使用，等待的时间会导致中断中的程序无法被执行。

信号量的释放使用的函数为：

```
rt_err_t rt_sem_release(rt_sem_t sem);
```

释放信号量可以唤醒在该信号量上的线程，使用一次信号量的值会增加 1。

### 3.4.2 GPS 处理与信号量

GPS 的信息提取是放在 main 线程当中，GPS 的信息处理是放在 gps\_deal 线程，GPS 的报文（字符串）的接收是通过 UART2 串口实现。GPS 的信息获取、解析、处理与舵机角度控制之间的时间延迟要尽可能的短暂，否则会造成控制上的延迟使得误差不能够及时修正，同时在新的 GPS 信号来临之前，对旧数据的解析没有意义而且会对资源造成浪费。

针对 GPS 的解析，我们使用了一个 newgpsgot 这一信号量，每当新的 GPS 报文送达时，便会立刻释放信号，及时进入 GPS 解析程序。

信号量的生成要优先于 UART2 中断的开启，否则释放一个空的信号量会导致程序崩溃。

```
//创建newgpsgot这一信号量
newgpsgot = rt_sem_create("gps_get", 0, RT_IPC_FLAG_FIFO);

uart_rx_irq(GPS_TAU1201_UART, 1);
```

图 3.4.2 创建 newgpsgot 信号量，位于 my\_gps\_init() 函数

```
if (UART2->ISR & UART_ISR_RX_INTF) // {
    UART2->ICR |= UART_ICR_RXICLR; //

    uart_getchar(UART_2, &buf);

    if (buf == '\r') {
        ReadBuff[num] = '\0';
        strcpy(Save_Data.GPS_Buffer, ReadBuff);
        Save_Data.isGetData = 1;
        num = 0;
        //信号释放
        rt_sem_release(newgpsgot);
    }
}
```

图 3.4.3 newgpsgot 信号量的释放，位于 UART2 接收中断

```

66 int main (void)
67 {
68     //创建信号
69     gps_calculation = rt_sem_create("gps_calculation",0,RT_IPC_FLAG_FIFO);
70     while (1)
71     {
72         //等待新的GPS数据送达
73         rt_sem_take(newgpsgot,RT_WAITING_FOREVER);
74         //解析GPS数据
75         parseGpsBuffer();
76         //获取初始的GPS数据
77         Get_InitData(&Save_Data.longi_Z, &Save_Data.longi_X, &Save_Data.lati_Z, &Save_Data.lati_X);
78         //修改GPS数据格式, 把ddmm.mm格式化成整数+小数格式, 便于计算
79         Trans_Data_Z_plus_X();
80
81         //解析完毕, 释放信号
82         rt_sem_release(gps_calculation);
83         rt_thread_mdelay(10);
84     }
85 }

```

图 3.4.4 newgpsgot 信号量获取, 位于 main 线程

同时, 针对 GPS 的数据的误差解算, 我们也设置了一个信号 gps\_calculation。每当 GPS 数据解析完毕之后, 便会释放信号便于 gps\_deal 线程继续执行 GPS 误差量计算的命令。

```

66 int main (void)
67 {
68     //创建信号
69     gps_calculation = rt_sem_create("gps_calculation",0,RT_IPC_FLAG_FIFO);
70     while (1)
71     {
72         //等待新的GPS数据送达
73         rt_sem_take(newgpsgot,RT_WAITING_FOREVER);
74         //解析GPS数据
75         parseGpsBuffer();
76         //获取初始的GPS数据
77         Get_InitData(&Save_Data.longi_Z, &Save_Data.longi_X, &Save_Data.lati_Z, &Save_Data.lati_X);
78         //修改GPS数据格式, 把ddmm.mm格式化成整数+小数格式, 便于计算
79         Trans_Data_Z_plus_X();
80
81         //解析完毕, 释放信号
82         rt_sem_release(gps_calculation);
83         rt_thread_mdelay(10);
84     }
85 }

```

图 3.4.5 创建信号、释放信号

```

591 void gps_deal_entry()
592 {
593     while(1)
594     {
595         //等待开始计算的信号送达
596         rt_sem_take(gps_calculation,RT_WAITING_FOREVER);
597
598         if (Save_Data.isUsefull == true && gps_mode == 1)
599         {
600
601             if (passed_sections == 0 && Save_Data.f_ground_speed < 1.8 && 0)
602             {
603                 //舵机固定中值, 防止车辆起步产生偏移
604                 pwm_duty_update(TIM_8,SERVO_PIN, pwm_mid_angle);
605                 //与第一个点的距离与航向计算, 显示在tft屏幕上, 为发车提供参考
606                 error_lenth_s_p = GESDistance(Save_Data.f_lati, Save_Data.f_longi, Points_U
607                 error_direction = GPSBearingAngle(Save_Data.f_lati, Save_Data.f_longi, Poin
608             }
609
610 }

```

图 3.4.6 gps\_calculation 信号获取

通过这两个信号, 保障了 GPS 数据的获取解析计算的过程, 避免了重复计算的过程, 提高了 GPS 数据解析的效率, 使得系统的实时性得到了更有力

---

的保障。同时也避免了 GPS 数据字符串的填充与解析的同时发生，虽然 GPS 的数据发送频率为 10Hz，使得单片机有充足的时间完成上一次的解算工作。

### 3.4.3 按钮的信号量

智能车下位机修改参数时，需要用到按键系统，为了便于调试且进一步学习信号量的知识点，于是在原有代码的基础上利用 RT-Thread 的信号量设计了一个按键人机交互系统。

首先，我们可以在按钮的初始化部分创建按键的信号量，当按键被按下时就会释放这个信号量。

```
141 void button_init()
142 {
143
144     gpio_init(KEY_UP, GPIO, GPIO_HIGH, GPIO_PULL_UP); // 初始化为GPIO浮空输入 默认上拉高电平
145     gpio_init(KEY_CENTER, GPIO, GPIO_HIGH, GPIO_PULL_UP);
146     gpio_init(KEY_LEFT, GPIO, GPIO_HIGH, GPIO_PULL_UP);
147     gpio_init(KEY_DOWN, GPIO, GPIO_HIGH, GPIO_PULL_UP);
148     gpio_init(KEY_RIGHT, GPIO, GPIO_HIGH, GPIO_PULL_UP);
149
150     //靠近五项按键的为1
151     //从车头方向看过去，右侧是关闭
152     gpio_init(SWITCH_1, GPIO, GPIO_HIGH, GPIO_PULL_UP); // 初始化为GPIO浮空输入 默认上拉高电平
153     gpio_init(SWITCH_2, GPIO, GPIO_HIGH, GPIO_PULL_UP);
154     gpio_init(SWITCH_3, GPIO, GPIO_HIGH, GPIO_PULL_UP); // 初始化为GPIO浮空输入 默认上拉高电平
155     gpio_init(SWITCH_4, GPIO, GPIO_HIGH, GPIO_PULL_UP);
156
157     keyup_sem = rt_sem_create("keyup", 0, RT_IPC_FLAG_FIFO); //创建按键的信号量
158     keycenter_sem = rt_sem_create("keycenter", 0, RT_IPC_FLAG_FIFO);
159     keyleft_sem = rt_sem_create("keyleft", 0, RT_IPC_FLAG_FIFO); //创建按键的信号量
160     keydown_sem = rt_sem_create("keydown", 0, RT_IPC_FLAG_FIFO);
161     keyright_sem = rt_sem_create("keyright", 0, RT_IPC_FLAG_FIFO); //创建按键的信号量
162 }
163
```

图 3.4.1 在按键初始化时创建信号量

```
//若存在按键更新
if (last_key_num != key_num && key_num != 0)
{
    if (key_num == CENTER)
    {
        rt_sem_release(keycenter_sem);
    }
    //up
    else if (key_num == UP)
    {
        rt_sem_release(keyup_sem);
    }
    //left
    else if (key_num == LEFT)
    {
        rt_sem_release(keyleft_sem);
    }
    //down
    else if (key_num == DOWN)
    {
        rt_sem_release(keydown_sem);
    }
    //right
    else if (key_num == RIGHT)
    {
        rt_sem_release(keyright_sem);
    }
}
```

图 3.4.2 根据按键信息释放对应的信号

```

18 void move()
19 {
20     uint8 i = x_cursor;
21     //LEFT
22     if (keyleft_sem->value >= 1)
23     {
24         rt_sem_take(keyleft_sem, 0);
25         //画图
26         for (i = 0; i < 8; i++)
27             lcd_drawpoint(x_cursor + i, y_cursor, WHITE);
28         //防止越界
29         if (x_cursor - 8 < 0)
30             x_cursor = 152;
31         else
32             x_cursor = x_cursor - 8;
33
34         //画图
35         for (i = 0; i < 8; i++)
36             lcd_drawpoint(x_cursor + i, y_cursor, BLACK);
37     }
38     //UP
39     else if (keyup_sem->value >= 1)
40     {
41         rt_sem_take(keyup_sem, 0);
42         for (i = 0; i < 8; i++)
43             lcd_drawpoint(x_cursor + i, y_cursor, WHITE);
44
45         if (y_cursor - 16 < 32)
46         {
47

```

图 3.4.3 根据信号量的值来确定对应的按键操作

释放信号量之后，可以根据信号量的值以及对应的信号量来执行对应的操作。

### 3.4.3 无线串口发送允许的信号量

因为操场跑道相比于室内跑道，范围更加的开阔。赛车所安装的无线串口因为要优先保障通信距离，所以无线串口波特率有所下调。针对这个改变，设置一信号量位于软件定时器中断当中，以便无线模块每隔足够长的时间发送数据包给上位机。

## 3.5 定时器的使用

### 3.5.1 定时器简介

定时器的作用是从指定的时刻开始，经过一定的时间触发一个时间。定时器有硬件和软件之分：

(1) 硬件定时器一般是由外部晶振给芯片提供时钟上的输入，芯片向软件模块提供一组配置寄存器，接受控制输入后，到达设定的时间值后芯片中断控制器会产生时钟中断。硬件定时器的精度很高，可以达到纳秒级别，由中断方式触发。

(2) 软件定时器是由操作系统提供的一类系统接口，使用软件定时器可以使得系统避免受到因为硬件定时器的不足产生的限制。但是时钟的定时数值必须是时钟节拍的整数倍，如果时钟节拍是 1ms 时，定时的时间只能是 1ms 的整数倍，不能定时 1.5ms 这样的时间。相比之下软件定时器的精度可能会低于硬件定时器。

下图是定时器的管理控制块，在管理层次上与信号量有点类似，都包括创建/初始化、启动、停止/控制、删除/脱离这四个操作。

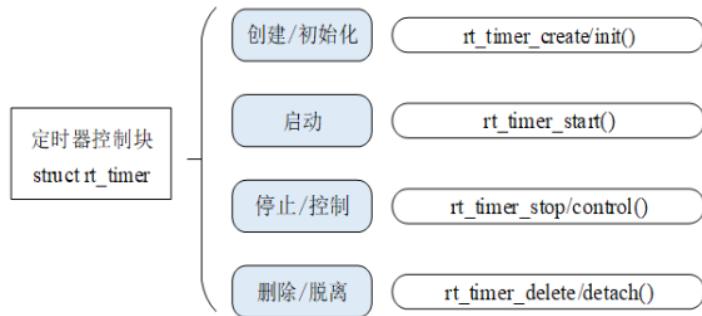


图 3.5.1 定时器控制块结构图

### 3.5.2 定时器应用

(1)利用软件定时器对按键调参进行控制：

```

//创建软件定时器
timerl = rt_timer_create("button_scan", Button_Control, RT_NULL, 15, RT_TIMER_FLAG_PERIODIC);

if(RT_NULL != timerl)
{
    rt_timer_start(timerl);
}

```

图 3.5.2 软件定时器创建

按键检测的定时器时间为 15ms，这个时间是结合五项按键的外设以及本人操作习惯得出来的最合适的时间。如果低于 15ms，则会导致按键过灵敏，难以精确控制参数加减，如果高于 15ms 太多，按键就会出现失灵，翻页的速度过慢。

接下来编写 Button\_Control 这一入口函数，该函数运行时每隔 15ms 会检测一遍按键的状态。

```

229 void Button_Control()
230 {
231     //迭代，按键消抖
232     last_key_num = key_num;
233     //按键值获取
234     key_num = Button_get();
235     //若存在按键更新
236     if (last_key_num != key_num && key_num != 0)
237     {
238         button_semrelease();
239     }
240     //按照五项按键中间键按下次数选择调试模式
241     button_mode_selection();
242     //如果button_mode是偶数，则是光标移动模式，否则为数值修改模式
243 }
244

```

图 3.5.3 按键控制函数

(2) 定时器控制的其它外设

首先创建一个 1ms 的定时器

```
//创建软件定时器
timer1 = rt_timer_create("time_pit_lms", time_pit_lms, RT_NULL, 1, RT_TIMER_FLAG_PERIODIC);
```

图 3.5.4 定时器创建

然后根据外设的时间要求，利用  $\text{total\_time \% t1 == 0}$  类似于这样的判断条件，每隔  $t1$  执行一次操作。放在该定时器中的外设主要有陀螺仪、舵机、无线串口。陀螺仪设定的时间是 2ms，每 2ms 根据当前的角速度进行角度积分。舵机的 PWM 频率为 50Hz，所以设定的是每 20ms 对舵机角度进行一次控制。

### 3.6 ENV 工具的使用

这是 RT-Thread 推出的辅助开发工具，针对基于 RT-Thread 操作系统的项目工程，提供编译构建环境、图形化系统配置及软件包管理功能。

这次我们使用了 ENV 工具中的 menuconfig 图形化配置界面，使用了 scons 工具生成工程，并生成了相应的 rtconfig.h 文件用于工程配置。

ENV 工具在 RT-Thread 的官网便可以获取下载，安装完成之后，便可以尝试打开 ENV 控制台了。打开 ENV 控制台推荐使用鼠标右键菜单，这种方法有利于路径的选择，添加 ENV 至右侧菜单的方法已经包括在了下载的文件当中了（添加 Env 至右键菜单.png），这里不再赘述。

我们使用了 Env 工具结合 scons 相关命令编译了 RT-Thread 与使用 menuconfig 配置了 BSP，操作的简要步骤介绍如下：

首先，需要进入 bsp 的根目录区域，比如说：rt-thread-gitee\_master\bsp\ mm32f327x。注意路径中不要出现中文，否则无法完成配置操作。

然后选择鼠标右键，如下图所示



图 3.6.1 利用鼠标右键打开 Env

接着，在控制台输入 menuconfig 命令，并回车确认，就会跳转到如下图所示界面。

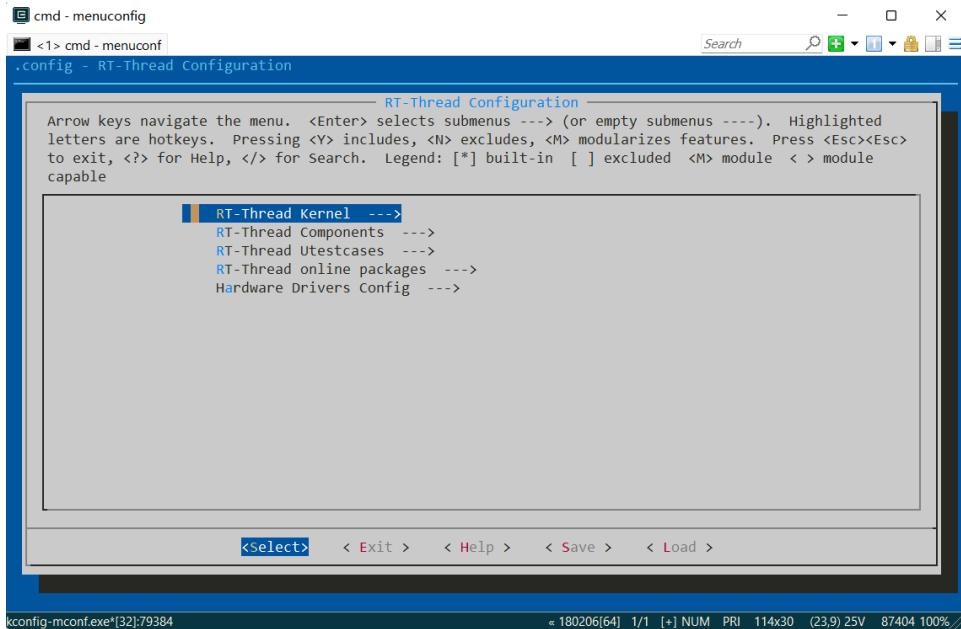


图 3.6.2 menuconfig 界面

进入 menuconfig 之后，便可以按照自己的实际需求对 RT-Thread 工程进行配置了。这次的配置修改了 command shell 处的内容，因为硬件板子的原因，串口 IO 容易受到干扰，使用 shell 组件时，有一定概率要连接串口其他线程才能够运行，否则只能运行 main 线程。再加上默认使用 UART1 串口进行通信，赛车在运行时无法与烧写器连接。基于这两个原因，同时也为了安全起见，在比赛时的烧录的程序中还是选择关闭了 Shell 组件。

关闭 Shell 组件需要进入 RT-Thread Components 这个选项，选定后按下回车，到达如下图所示的界面。

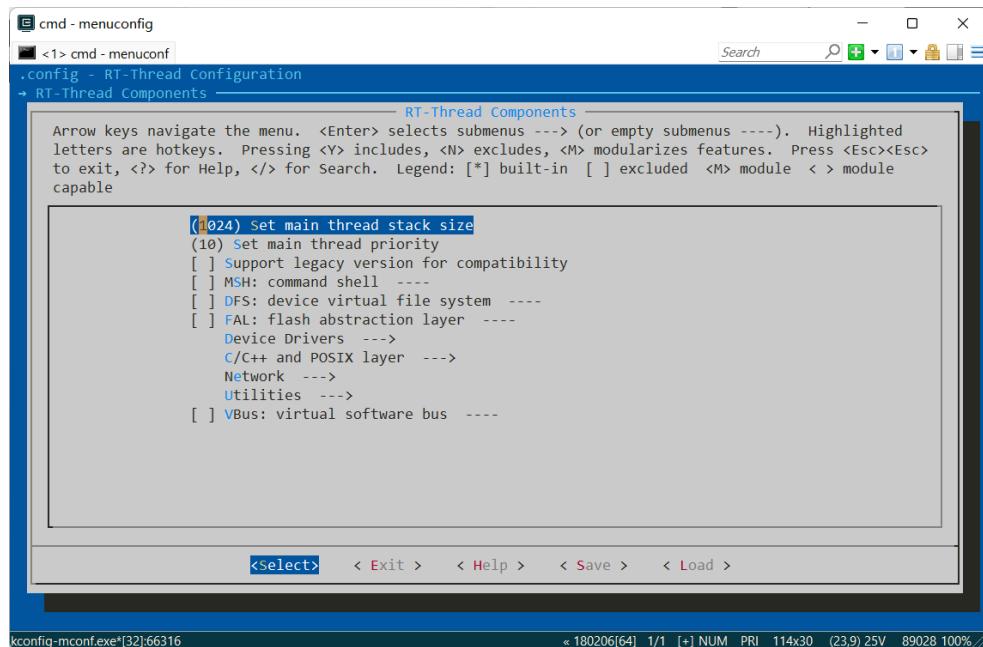


图 3.6.3 关闭 shell 组件

选定其中的 MSH:command shell 选项, 按下 N 键进行关, 并保存好配置。根据提示, 保存为一个对应的 config 文件。

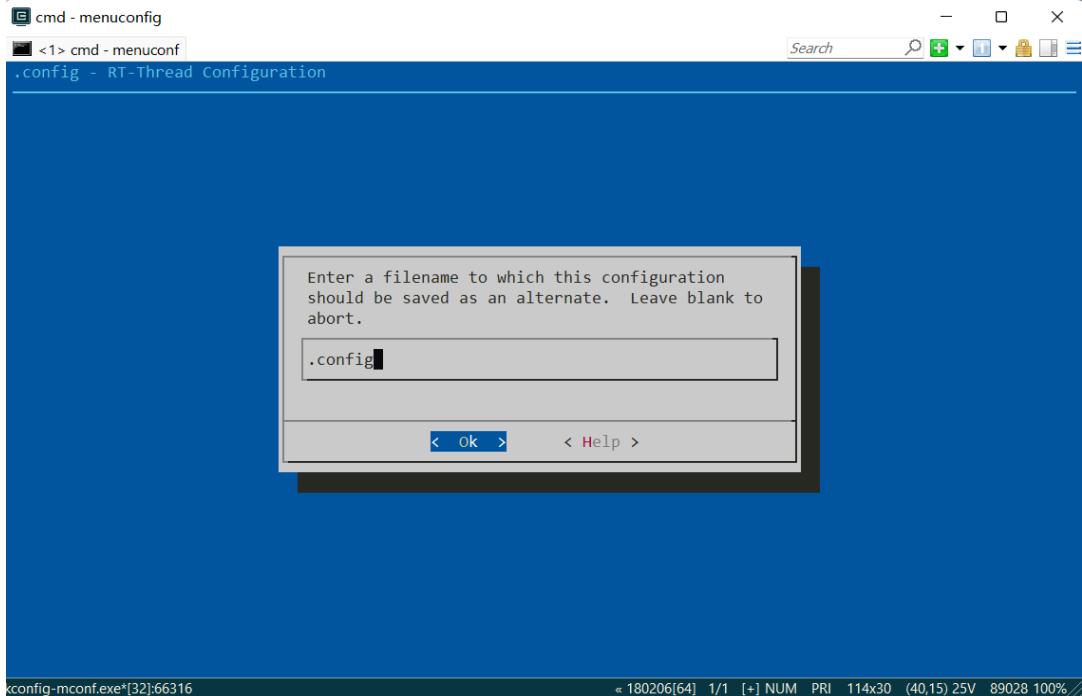


图 3.6.4 生成配置保存.config 文件

从 menuconfig 退出之后, 回到之前的控制台窗口。根据使用的开发软件, 输入对应的命令, 以作者为例, 使用的是 MDK5 文件, 则输入 scons --target=mdk5 这一命令重新生成对应的工程。

```
cmd
<1> cmd
CC build\Libraries\MM32F327X\HAL_lib\src\hal_comp.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_crc.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_crs.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_dac.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_dbg.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_dma.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_eth.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_exti.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_flash.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_fsmc.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_gpio.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_i2c.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_iwdg.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_misc.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_pwr.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_rcc.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_rtc.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_sdio.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_spi.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_tim.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_uart.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_uid.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_ver.o
CC build\Libraries\MM32F327X\HAL_lib\src\hal_wwdg.o
AS build\Libraries\MM32F327X\Source\KEIL_StartAsm\startup_mm32f327x_keil.o
CC build\Libraries\MM32F327X\Source\system_mm32f327x.o
LINK rtthread.elf
fromelf --bin rtthread.elf --output rtthread.bin
fromelf -z rtthread.elf
scons: done building targets.

lenovo@LAPTOP-5PI54N1P E:\RTT\rt-thread-gitee_master\rt-thread-gitee_master\bsp\mm32f327x>
cmd.exe*[64]:86200
```
180206[64] 1/1 [+]
NUM PRI 114x30 (40,15) 25V 89028 100% //
```

图 3.6.5 生成完毕

---

完成生成之后，使用新的工程时便获得了保存下来的配置，或者利用重新生成的 rtconfig.h 这一文件替换原有的 rtconfig.h 文件也可以。

Env 工具中的 packages 文件中包含了其他软件包的索引，通过这些索引文件，我们就可以找到对应的软件包的 github 下载地址。

## 第四章 RT-Thread 开发中的创新点

### 4.1 RT-Thread 的软件包的获取与开发

RT-Thread 物联网操作平台上有面向不同的领域的软件包，软件包由描述信息、源代码或者库文件组成。在刚刚开始接手这个比赛项目前，我们就参考了 RT-Thread 中 gps\_rmc 软件包中的代码，编写了自己用足用于解析 GPS 模块的\$XXRMC 类型数据的函数。

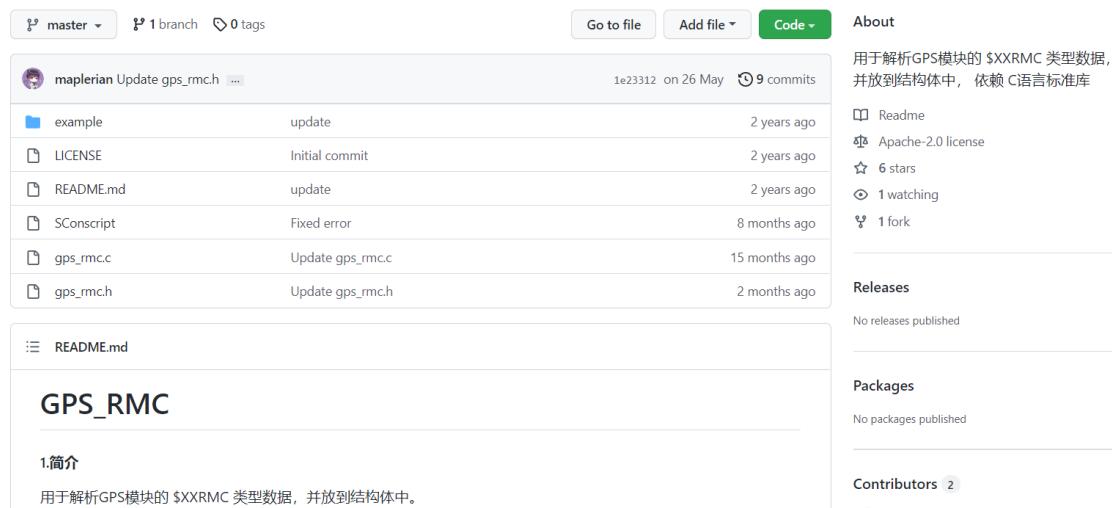


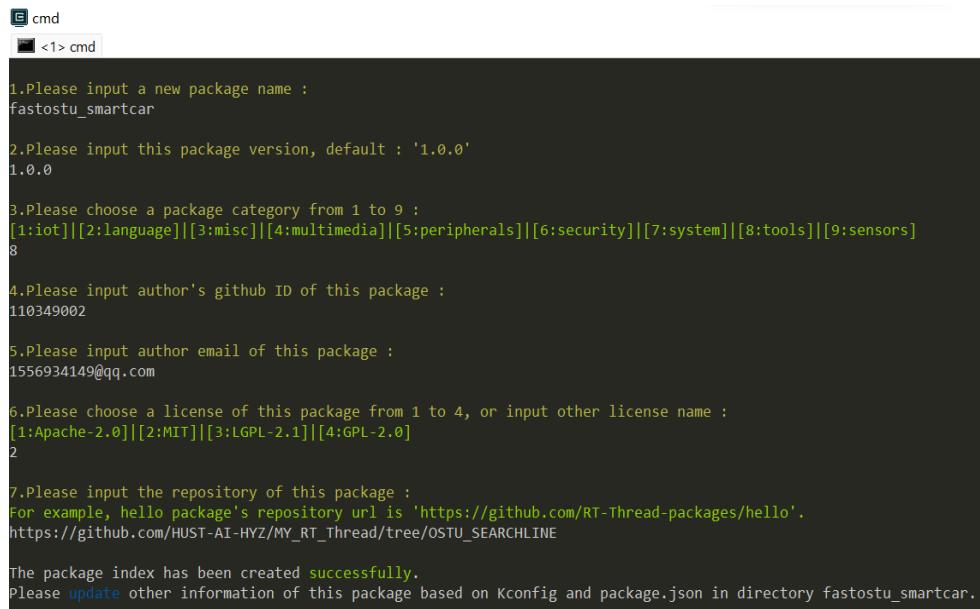
图 4.1.1 GPS\_RMC 软件包的 github 链接

有时候使用自带的软件包进行程序开发，可以避免重复造轮子的操作，节约开发人员的时间，使其能够把更多的精力用在一些有创意的创新性工作上。在撰写论文的这几天中阅读官网的技术文档时，我们看到了代码贡献栏中的软件包开发指南的内容。当再次仔细浏览了当时使用过的 GPS\_RMC 软件包中的内容时，发现其实创建一个简单的软件包也并没有想象中的那么困难。本着学习交流为社区做贡献的目的，我尝试了开发了一个简单的软件包。

首先，要确定自己的软件包中的内容不应当与已有的软件包内容重叠。我在软件包下载中心按照“OTSU”、“IMAGE”等关键词进行搜索时，结果发现未找到匹配的软件包。然后，我确定了软件包中的内容，准备在我的第一个软件包当中添加有关智能车图像处理搜线算法的处理代码，这是我在车队培训阶段结业时撰写的一个小项目。智能车处理蓝白黑赛道一直是一个非常重要的任务，也是一些室内组别比赛的关键。本着实用性与通用性的原则，我决定把我当时使用的程序整理成软件包上传，借这个机会与智能车选手交流提升自身的能力。

---

软件包中应当包含两大主要内容：软件包代码与说明文档、软件包索引。软件包代码与说明文档可以参照官网给出的 hello 示例编写内容。软件包的索引创建可以使用 Env 工具中的索引生成向导，使用命令 pkgs --wizard 便会进入创建流程。



```
cmd
<1> cmd

1. Please input a new package name :
faststu_smartcar

2. Please input this package version, default : '1.0.0'
1.0.0

3. Please choose a package category from 1 to 9 :
[1:iot][2:language][3:misc][4:multimedia][5:peripherals][6:security][7:system][8:tools][9:sensors]
8

4. Please input author's github ID of this package :
110349002

5. Please input author email of this package :
1556934149@qq.com

6. Please choose a license of this package from 1 to 4, or input other license name :
[1:Apache-2.0][2:MIT][3:LGPL-2.1][4:GPL-2.0]
2

7. Please input the repository of this package :
For example, hello package's repository url is 'https://github.com/RT-Thread-packages/hello'.
https://github.com/HUST-AI-HYZ/MY_RT_Thread/tree/OSTU_SEARCHLINE

The package index has been created successfully.
Please update other information of this package based on Kconfig and package.json in directory faststu_smartcar.
```

图 4.1.2 填写软件包索引信息

我们需要填写的信息有 1-7 类，在填写完成后，会自动产生对应的文件。软件包索引用于后续提交，便属于自己软件包的推广。

然后，按照官方文档上的说明，修改其中的 package.json 这个文件。是软件包的描述信息文件，包括许可证信息，软件包名称，软件包描述，作者等信息，以及必须的 package 代码下载链接。根据需求，软件包的 Kconfig 文件也可以进行一定的修改。

接下来，上传软件包，软件包我上传到了自己的 git 账号中。

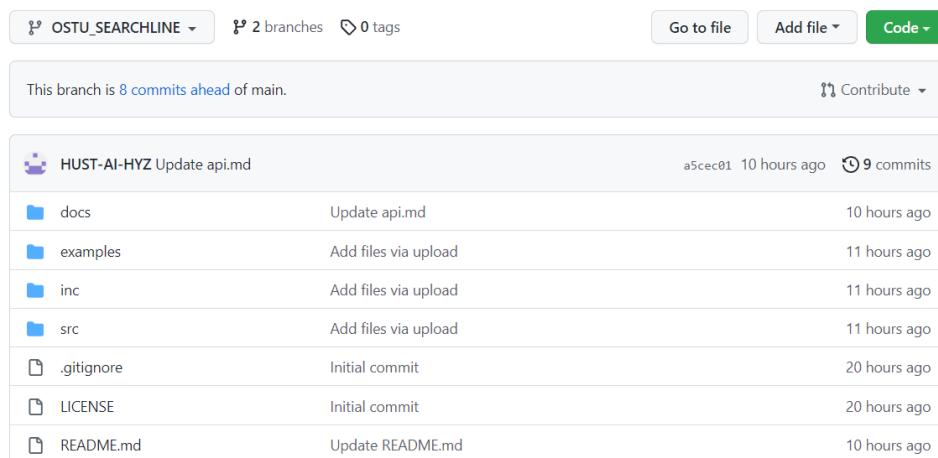


图 4.1.3 上传在 git 的软件包

后续阶段就是测试自己的软件包，等测试通过之后会稍作一定的修改与优化，最后向软件包索引提交 PR，并进行审核。

## 4.2 RT-Thread 参考文档的使用与修改

为了方便开发人员对 RT-Thread 的学习，RT-Thread 官网处提供了各个版本的指导文件。官网针对标准版本或者是 Nano 版本有各自的说明文档，学习编程时可以参考文档中的范例程序理解原理与使用方法。

The screenshot shows the RT-Thread documentation website with the 'Kernel Foundation' page selected. The page contains a diagram illustrating the RT-Thread kernel architecture. The diagram is divided into several sections: 'Kernel Library' (containing '内核库' and 'kservice.h/c'), 'Object Management' (containing '对象管理: object.c'), 'Real-time Scheduler' (containing '实时调度器: scheduler.c'), 'Thread Management' (containing '线程管理: thread.c'), 'Inter-Thread Communication' (containing '线程间通信: ipc.c'), 'Clock Management' (containing '时钟管理: timer.c'), 'Memory Management' (containing '内存管理: mem.c/memheap.c...'), and 'Device Management' (containing '设备管理: device.c'). Below this is the 'CPU Architecture Support' section (containing '芯片移植: rccpu' and 'BSP (Board Support Package) 板级支持包: bsp'), and at the bottom is the 'Hardware' section (containing '硬件: CPU/RAM/Flash/UART/ETH/AC等'). A note below the diagram states: '内核库是为了保证内核能够独立运行的一套小型的类似 C 库的函数实现子集。这部分根据编译器的不同自带 C 库的情况也会有些不同。当使用 GNU GCC 编译器时，会携带更多的标准 C 库实现。' A tip box says: '提示: C 库: 也叫 C 运行库 (C Runtime Library)，它提供了类似 "strcpy"、"memcpy" 等函数，有些也会包括 "printf"、"scanf" 函数的实现。RT-Thread Kernel Service Library 仅提供内核用到的一小部分 C 库函数实现，为了避免与标准 C 库重名，在这些函数前都会添加上 rt\_ 前缀。'

图 4.2.1 RT-Thread 参考文档内核基础页

我们注意到，文档的编辑权限是开放的，参与修改只需要提交 PR，关于 PR 的提交有专门的文章介绍，通过后台审核完成后会并入仓库。

修改的第二种方法可以在 gitee 页面中添加 issue 作为建议进行修改，因为这一次修改需要更换图片，添加图片与修改图片需要提交两次 PR，所以我们尝试在 issue 处提出了我们的建议，希望对参考文档中过时的说明图片进行更换。

The screenshot shows a Gitee issue page with the title '更新软件包开发指南中的图片' (Update the software package development guide's picture). The issue details are as follows:

- 状态: 待办的 (Pending)
- 议题 ID: #15JUYW
- 创建者: 胡圆旗
- 创建时间: 2022-08-01 11:53
- 操作: 编辑 | 删除

The main content area displays the RT-Thread documentation center with the 'Software Package Development Guide' section selected. It shows a comparison between the old screenshot and the new one. The new screenshot shows a more recent version of the 'env' interface for generating package indexes.

软件开发包指南中使用索引生成向导的env界面图片版本较老，请求更新，例如更新为：

图 4.2.2 提出 issue 建议图片更换

因为现在还是在对 RT-Thread 深入学习的阶段，一些技术层面的文档问题凭借我们的能力比较难发现，但是在使用过程中发现的更新需求还是提出了一点建议，希望通过自己的绵薄之力为 RT-Thread 社区做出一些贡献。

### 4.3 利用 RT-Thread 社区获取技术解答

RT-Thread 嵌入式开源社区拥有丰富的资源，我们组曾在 Finsh 组件上遇到的线程问题其他的开发者早就有提问。在结合了他们给出的建议之后，最终选择了在比赛时关闭 Finsh 组件的方式。一些常见的问题基本上都已经有开发者先提出过了，阅读这样的提问与文章可以减少开发过程中遇到的困难。

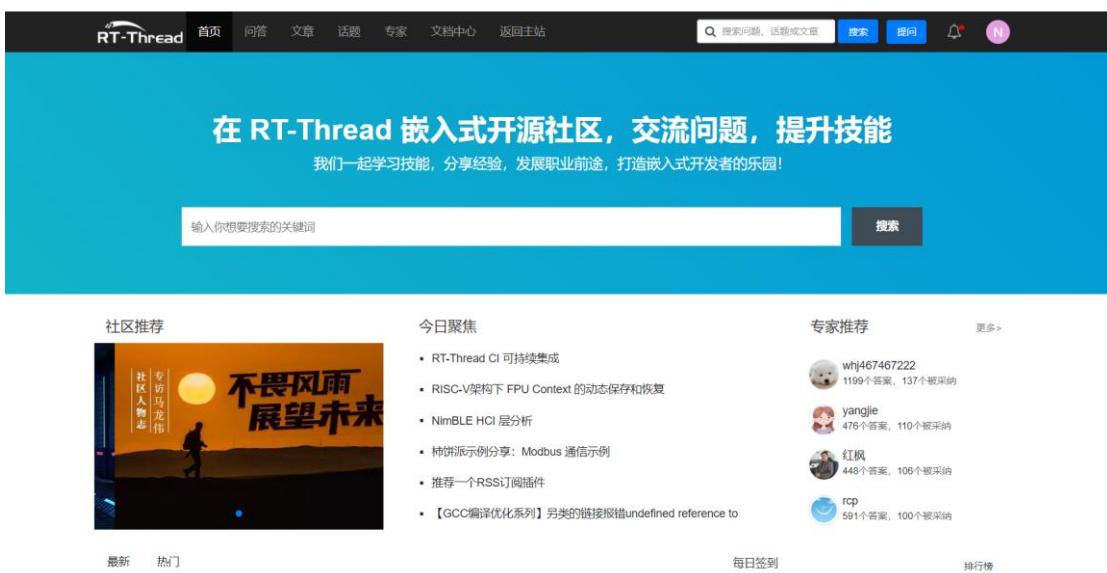


图 4.3.1 RT-Thread 社区界面

### 4.4 与裸机开发的横向比较

裸机开发与使用 RT-Thread 相比，有一些很明显的特征与不足。

(1) 裸机开发的难以避免在主程序中放一个 `while(1)` 循环，然后通过定时器中断调度其他的外设。在 `while(1)` 循环中难以避免会引入一些延时函数 `delay()`，比如流水灯显示以及按键消抖。但是传统的裸机开发使用 `delay()` 函数明显使得 CPU 的利用率下降，也使得一些任务的执行周期变长。在 `while(1)` 中如果安排了一些等待执行的代码段，在等待期间就会一直占据 CPU 的控制权造成空转。但是 RT-Thread 操作系统可以解决这两个问题，首先 `delay()` 函数能够释放 CPU 的控制权，而且通过进程挂起也可以对 CPU 控制权进行释放。使用 RT-Thread 可以很好地解决裸机开发中使用大型 `while(1)` 循环带来的问题。而且放在 `while(1)` 循环中的一些代码，比如屏幕显示函数，

这个函数对比赛时的车模的运动控制毫无帮助，甚至会抢占 CPU 资源，但是通过线程优先级的合理分配，可以使得 CPU 处理完其他重要任务时有充分的资源再调动这些函数。

(2) 裸机开发的实时性保障较差。裸机开发时使用定时器中断来保障外设处理的实时性，但是当定时器中断的外设处理代码增多任务处理更加复杂时，传统的定时查询方式实时性也就难以得到保障。如果引入 RT-Thread 的多线程控制与线程通信的思想，对于任务完成的安排也会更加的明朗，可以有效的保障系统的实时性。

(3) 线程间的通信会以全局变量为主，全局变量在嵌入式开发中应当少用慎用，可以避免一些不必要的弊端。但是在传统的编程模式下各个任务之间的通信很难避免不适用全局变量。使用 RT-Thread，利用信号量、互斥量的方式实现线程间的通信，在一定程度上减少了全局变量的使用。

## 4.5 RT-Thread 开发的主要优势

本次设计时采用的 RT-Thread 的功能还是比较直观，操作系统很多强大的功能很多有利的优势并没有显现出来。如果比赛的任务更加复杂，应用场景要求更高，RT-Thread 相比于裸机直接开发的优越性会更加的明显。

### (1) 提升程序执行效率

就像在前文中谈到的那样，RT-Thread 通过多线程管理方式，相比于裸机开发对 CPU 的利用程度更高。而且通过线程的优先级分配，可以使得高优先级任务更难被低优先级任务干扰

### (2) 提升代码的可读性，程序结构更加清晰

在裸机开发中，很多时候代码如果堆在 while(1) 大循环中，使得程序的可读性与逻辑性较差。但是如果使用线程控制的思想，不同的任务安排在不同的线程，就可以提升代码的运行效率，也更有利于模块化编程。

### (3) 系统的实时性将会得到有力的保障

通过多线程编程中优先级的安排，使得任务执行能够统筹规划，保障了高优先级的任务能够及时完成。

---

## 第五章 总结与展望

### 5.1 总结与感想

极速越野组是一个对实时性要求很高的系统，稍有控制上的不慎就会导致车辆损毁的惨剧。本次比赛我们小组以灵动微 MM32 系列芯片作为平台，结合 RT-Thread 操作系统，对其工作的机制以及系统架构进行了学习与应用。

这次比赛对极速越野赛车的软件与硬件系统进行了详细的设计：

- 采用 GPS，陀螺仪进行信息采集
- 使用无刷驱动板进行车辆速度控制
- 使用舵机与 PD 控制法控制车辆转向
- 使用液晶显示屏、拨码器、五向按钮作为人机交互手段
- 使用无线串口进行赛车与 PC 机之间的通信。

同时也开发了小车的控制算法、参考开源项目设计了无刷驱动的程序、利用 MATLAB、MISSION PLANNER 等软件设计了路径生成算法。针对 RT-Thread 的应用，仔细学习了相关的技术手册以及内核实验报告，对 RT-Thread 操作系统基本的架构以及内核有了初步的了解，尝试将 RT-Thread 操作系统应用在赛车设计上并成功完成了比赛。

通过参加这次比赛，我们不仅对平时课上所学的 C 语言知识进行了进一步的加深应用与学习，同时第一次站在嵌入式软件开发者的角度学习了 RT-Thread 的内核应用，也尝试了利用自己的能力尝试为 RT-Thread 社区做出贡献。虽然 RT-Thread 的学习并不是一帆风顺的，就连基础的 RT-Thread 操作系统的移植就耗费了不少的精力，中途偶尔也想过放弃学习，转回之前的裸机开发模式。好在，心中对嵌入式开发对物联网系统的热爱以及那股不服输的劲头让我们走到了最后，让我们能站在一个作者的角度总结自己的学习。当我们第一次把各个线程划分正确且合理时，当我们成功发现并消灭一个又一个的断言错误、硬件中断错误以及逻辑漏洞时……那种进步的喜悦让人难以忘怀，让我们意识到了坚持所带来的进步。

在这里还是真诚的感激 RT-Thread 的技术传播者们，他们录制了通俗易懂的教学视频、精心编写了详细的技术文档、提交了功能齐全的软件包以及解答了那些在社区提出的问题……正是因为他们的开源精神与不懈努力，使得一个个技术新人在成长的道路上能够走得更加顺利更加踏实。我们希望通过学习通过项目实战的经验，不断的成长，实现由技术的“索取者”向技术的“贡献者”和技术的“创造者”的进步与飞跃。比如：可以在平台发布一些更有使用价值的软件包、在技术论坛积极有效地解决他人的提问、参加 RT-Thread 组织的开发者能力认证（RAC）、作为参赛的选手在学校的车队宣传 RT-Thread 的使用或者为使用该操作系统的学弟学妹提供指导等等。

### 5.2 工作的不足与未来的展望

(1) 在备赛的节奏把控上存在一定的欠缺，低估了无刷驱动板的制作难

度。按照当时的进度安排，当有刷驱动程序达到速度上限时，便开始更换无刷驱动进行下一步的调试。这个策略不能说是错误的，因为如果无刷驱动迟迟未解决，控制算法的调试将会难以进行。查阅我们小组的开发日志，可知我们组的赛车早在 5 月 14 日便基本达到了有刷驱动的速度峰值。但是直到 6 月 29 日我们小组才将 MM32SPIN360C 芯片控制的无刷电机驱动安装在赛车上，中间也有考试月影响了开发进度，但是这宝贵的一个月的备赛时间内工作效率并不高。

但是第一版无刷驱动在 7 月 10 日损坏以后，稳定的 MM32SPIN27C 的驱动板过了将近一周的时间才投入使用。中间因为遇到了一些技术上的麻烦而且 360C 的驱动方案没有准备零件导致备赛效率下降。

(2) 机械结构上的调整与问题发现不够及时，实验室遗留下来的老 L 车模转向结构存在问题，当车辆速度提升时，容易出现侧偏的现象。硬件问题没有及时排除进而导致了软件开发上的方向也出现过偏差。

(3) 调试方法不够合理以及早期调试场地选择不够恰当。刚更换无刷驱动板时，因为车速的提升，一些隐藏的问题也相继暴露出来了。但是调试时速度提升没有讲究循序渐进，明明 9m/s 的车速还没有稳定完赛，下一次却直接尝试 12m/s 以上的车速。所以也出了一两次严重的调试事故，好在后期调试时对速度提升的严格把控以及紧急刹车模块的投入使用使得比赛前车辆基本没有大的碰撞事故发生。

最开始调试时，因为算法不成熟容易产生安全事故，我们组选择的是一个三面有高楼的空场地调试车辆，迫于较差的 GPS 信号，一开始的算法设计上增加了不少没有必要的处理任务。等到达操场这种空旷的大场地之后，才发现之前走了不少的弯路浪费了不少的时间。

好在这些问题，我们在比赛前都尽力解决了，但是中间的时间浪费却难以弥补回来了，当我们组知道如何更好的调试更好的提速时，区赛基本上要开始了，所以希望以后类似的错误不要再犯。

## 附录 A.部分程序源代码

---

## Main.c

```
1. #include "headfile.h"
2. #include "VOFA.h"
3. #include "math.h"
4. #include "button.h"
5. #include "parameter.h"
6. #include "common.h"
7. #include "flash.h"
8. #include "gps.h"
9. #include "isr.h"
10. #include "motor.h"
11. rt_sem_t camera_sem;
12.
13. //initialization adaptation
14. int device_init(void)
15. {
16.     //GPS
17.     my_gps_init();
18.     //陀螺仪
19.     icm20602_init_spi();
20.     //显示屏
21.     lcd_init();
22.     //电机初始化
23.     motor_init();
24.     //无线串口
25.     seekfree_wireless_init();
26.     //舵机
27.     servo_motor_init();
28.     //按钮
29.     button_init();
30. }
31.
```

```
32. int app_init(void)
33. {
34.     //Flash 读操作
35.     FlashReadParams();
36.     //舵机占空比归中
37.     duoji = pwm_mid_angle;
38.     //初始化四个关键点
39.     generating_route_init();
40.     //下位机线程初始化
41.     slavecomputer_init();
42.     //按钮扫描线程初始化
43.     button_scan_init();
44.     //上位机发送初始化
45.     send_init();
46.     //GPS 处理线程初始化
47.     gps_deal_init();
48. }
49.
50. INIT_DEVICE_EXPORT(device_init);
51. INIT_APP_EXPORT(app_init);
52.
53. int main(void)
54. {
55.     //创建信号
56.     gps_calculation = rt_sem_create("gps_calculation", 0, RT_IPC_F
LAG_FIFO);
57.     while (1)
58.     {
59.         //等待新的 GPS 数据送达
60.         rt_sem_take(newgpsgot, RT_WAITING_FOREVER);
61.         //解析 GPS 数据
62.         parseGpsBuffer();
```

```
63.         //获取初始的 GPS 数据
64.         Get_InitData(&Save_Data.longi_Z, &Save_Data.longi_X, &Save
   _Data.lati_Z, &Save_Data.lati_X);
65.         //修改 GPS 数据格式，把 ddmm.mm 格式化成整数+小数格式，便于计算
66.         Trans_Data_Z_plus_X();
67.
68.         //解析完毕，释放信号
69.         rt_sem_release(gps_calculation);
70.         rt_thread_mdelay(10);
71.     }
72. }
```

## Route\_generating.c (自动生成路径)

```
1. #include "parameter.h"
2. #include "gps.h"
3. #include <stdio.h>
4. #include <math.h>
5. #include <stdlib.h>
6. #include "route_generating.h"
7. #include "button.h"
8. //用于保存产生的路径
9. typedef struct RPoints
10. {
11.     //纬度
12.     double rps_latitude;
13.     double rps_longitude;
14.     //点类型
15.     int rps_type;
16. } RPoints;
17. //暂时用 double 型保存，别的类型可以后续转换
18.
```

```
19. #define PI 3.14159265357
20. //操场
21. FourPoints kps_plygrnd[4];
22. RPoints rps_plygrnd[150];
23.
24. int generate_round_road(float step, FourPoints strt, FourPoints end,
   d, int *delt_dots);
25. int generate_direct_road(float step, FourPoints strt, FourPoints end,
   int *delt_dots);
26. int write_in(int delt_dots, double *dis, double *direction);
27.
28. void generating_route_init()
29. {
30.
31.     kps_plygrnd[0].kps_latitude = 30.4844538;
32.     kps_plygrnd[0].kps_longitude = 114.303869;
33.     kps_plygrnd[0].kps_type = 0;
34.     //点 1
35.     kps_plygrnd[1].kps_latitude = 30.483704;
36.     kps_plygrnd[1].kps_longitude = 114.3039757;
37.     kps_plygrnd[1].kps_type = 1;
38.
39.     //点 2
40.     kps_plygrnd[2].kps_latitude = 30.4837835;
41.     kps_plygrnd[2].kps_longitude = 114.3048075;
42.     kps_plygrnd[2].kps_type = 2;
43.
44.     //点 3
45.     kps_plygrnd[3].kps_latitude = 30.4845494;
46.     kps_plygrnd[3].kps_longitude = 114.3047000;
47.     kps_plygrnd[3].kps_type = 3;
48.
```

```
49.         //每个点的间隔(直道路)
50.         distance_each_point_direct = 5.0;
51.         //每个点的间隔 (弯道路)
52.         distance_each_point_round = 5.0;
53.
54.         if (write_flash_points == 1)
55.         {
56.             generating_route();
57.             write_flash_points = 5;
58.         }
59.     }
60.
61. void generating_route()
62. {
63.     //初始化数据
64.     //起点 0
65.
66.     //临时数据 (中点)
67.     double temp_mid_point_lati = 0.0;
68.     double temp_mid_point_longi = 0.0;
69.
70.     //处理的点数
71.     int delt_dots = 0;
72.     //校验用数组
73.     double exam_distance[200] = {0.0};
74.     double exam_direction[200] = {0.0};
75.
76.     //处理第一段直道
77.     rps_plygrnd[0].rps_latitude = kps_plygrnd[0].kps_latitude;
78.     rps_plygrnd[0].rps_longitude = kps_plygrnd[0].kps_longitude;
e;
```

```
79.         rps_plygrnd[0].rps_type = 0;
80.         generate_direct_road(distance_each_point_direct, kps_plygrnd[0], kps_plygrnd[1], &delt_dots);
81.         generate_round_road(distance_each_point_round, kps_plygrnd[1], kps_plygrnd[2], &delt_dots);
82.         generate_direct_road(distance_each_point_direct, kps_plygrnd[2], kps_plygrnd[3], &delt_dots);
83.         generate_round_road(distance_each_point_round, kps_plygrnd[3], kps_plygrnd[0], &delt_dots);
84.
85.         //校验组
86.         for (size_t i = 0; i < delt_dots + 1; i++)
87.         {
88.             exam_distance[i] = GPSDistance(rps_plygrnd[i].rps_latitude, rps_plygrnd[i].rps_longitude, rps_plygrnd[i + 1].rps_latitude, rps_plygrnd[i + 1].rps_longitude);
89.             exam_direction[i] = GPSBearingAngle(rps_plygrnd[i].rps_latitude, rps_plygrnd[i].rps_longitude, rps_plygrnd[i + 1].rps_latitude, rps_plygrnd[i + 1].rps_longitude);
90.         }
91.
92.         write_in(delt_dots, exam_distance, exam_direction);
93.         //分配区域
94.         round_in_out();
95.     }
96.
97. //角度解算要注意 360 这个条件
98.
99. int generate_round_road(float step, FourPoints strt, FourPoints end, int *delt_dots)
100. {
```

```
101.         double mid_lati = (strt.kps_latitude + end.kps_latitude  
102.             ) / 2;  
103.         double mid_longi = (strt.kps_longitude + end.kps_longitude) / 2;  
104.         double radius = GPSDistance(mid_lati, mid_longi, strt.k  
105.             ps_latitude, strt.kps_longitude);  
106.         //计算角度改变量,sina=a,注意弧度值和角度值转换  
107.         double delta_sita = (step / radius) * 180 / PI;  
108.         //计算次数  
109.         int times = (int)(180 / delta_sita);  
110.         //重新算角度  
111.         delta_sita = 180 * 1.0 / times;  
112.         double vector_lati = 0.0;  
113.         double vector_longi = 0.0;  
114.         //计算中点与右侧点的向量  
115.         vector_lati = strt.kps_latitude - mid_lati;  
116.         vector_longi = strt.kps_longitude - mid_longi;  
117.  
118.         //有没有过半  
119.         int passbyhalf = 0;  
120.         //与之垂直的向量  
121.         double verti_vector_lati = vector_longi;  
122.         double verti_vector_longi = -vector_lati;  
123.         //方向向量  
124.         //角度比率,这次使用搜索法  
125.         double a_ratio = 0.001;  
126.         double dis_ratio = 0.001;  
127.         double error_ratio = 0;  
128.         double direction_vector_lati = vector_lati + verti_vect  
or_lati * a_ratio;
```

```
129.         double direction_vector_longi = vector_longi + verti_ve  
                      ctor_longi * a_ratio;  
130.         //圆直径基准角  
131.         double basic_angle = 0.0;  
132.         //从右到左  
133.         basic_angle = GPSBearingAngle(mid_lati, mid_longi, strt  
                      .kps_latitude, strt.kps_longitude);  
134.  
135.         //退出方式  
136.         bool out_way = true;  
137.  
138.         int i = 1;  
139.  
140.         //划分后的距离(调试用)  
141.         double angle_sum = 0;  
142.         double gene_dis1[80] = {0.0};  
143.         double gene_dis2[80] = {0.0};  
144.         double gene_dis3[80] = {0.0};  
145.         double di_angle[80] = {0.0};  
146.  
147.         //水平向量方向  
148.  
149.         while (i < times + 1)  
150.         {  
151.             //第 i 个点与圆心的方位角  
152.             double dir_angle = GPSBearingAngle(mid_lati, mi  
                           d_longi, mid_lati + direction_vector_lati, mid_longi + direction_v  
                           vector_longi);  
153.             //改变量(过 360 那个坎)  
154.             double change_angle = delta_heading_angle(basic  
                           _angle, dir_angle);  
155.
```

```
156.             while (change_angle < angle_sum + delta_sita)
157.             {
158.                 if (passbyhalf == 0)
159.                 {
160.                     direction_vector_lati = vector_
161.                         lati + verti_vector_lati * a_ratio;
162.                     direction_vector_longi = vector_
163.                         _longi + verti_vector_longi * a_ratio;
164.                     a_ratio = a_ratio + 0.001;
165.                 }
166.                 else
167.                 {
168.                     direction_vector_lati = -vector_
169.                         _lati + verti_vector_lati * a_ratio;
170.                     direction_vector_longi = -vecto_
171.                         r_longi + verti_vector_longi * a_ratio;
172.                     a_ratio = a_ratio - 0.001;
173.                 }
174.             }
175.
176.             //第 i 个点与圆心的方位角
177.             dir_angle = GPSBearingAngle(mid_lati, m_
178.                 id_longi, mid_lati + direction_vector_lati, mid_longi + direction_
```

```

179.         while (dis_tomidpnt < radius)
180.     {
181.         dis_ratio = dis_ratio + 0.001;
182.         dis_tomidpnt = GPSDistance(mid_lati, mi
183.                                     d_longi, mid_lati + direction_vector_lati * dis_ratio, mid_longi +
184.                                     direction_vector_longi * dis_ratio);
185.     }
186.     //第一次分析
187.     rps_plygrnd[*delt_dots + i].rps_latitude = mid_
188.                               lati + direction_vector_lati * dis_ratio;
189.     rps_plygrnd[*delt_dots + i].rps_longitude = mid_
190.                               _longi + direction_vector_longi * dis_ratio;
191.     rps_plygrnd[*delt_dots + i].rps_type = 1;
192.     gene_dis1[i] = GPSDistance(rps_plygrnd[*delt_d
193.                               ts + i].rps_latitude, rps_plygrnd[*delt_dots + i].rps_longitude, r
194.                               ps_plygrnd[*delt_dots + i - 1].rps_latitude, rps_plygrnd[*delt_dot
195.                               s + i - 1].rps_longitude);
196.     //如果距离出现问题
197.     if (gene_dis1[i] < step * 0.90 || gene_dis1[i]
198.         > step * 1.10)
199.     {
200.         error_ratio = step / gene_dis1[i];
201.     }
202.     //再来一次
203.     rps_plygrnd[*delt_dots + i].rps_latitud
204.     e = rps_plygrnd[*delt_dots + i - 1].rps_latitude + (rps_plygrnd[*d
205.                               elt_dots + i].rps_latitude - rps_plygrnd[*delt_dots + i - 1].rps_l
206.                               atitude) * error_ratio;
207.     rps_plygrnd[*delt_dots + i].rps_longitu
208.     de = rps_plygrnd[*delt_dots + i - 1].rps_longitude + (rps_plygrnd[
```

```
*delt_dots + i].rps_longitude - rps_plygrnd[*delt_dots + i - 1].rp  
s_longitude) * error_ratio;  
199.                                rps_plygrnd[*delt_dots + i].rps_type =  
    1;  
200.                                }  
201.        else  
202.        {  
203.        }  
204.        di_angle[i] = GPSBearingAngle(mid_lati, mid_lon  
gi, rps_plygrnd[*delt_dots + i].rps_latitude, rps_plygrnd[*delt_do  
ts + i].rps_longitude);  
205.        angle_sum = delta_heading_angle(basic_angle, di  
_angle[i]);  
206.  
207.        gene_dis2[i] = GPSDistance(rps_plygrnd[*delt_do  
ts + i].rps_latitude, rps_plygrnd[*delt_dots + i].rps_longitude, r  
ps_plygrnd[*delt_dots + i - 1].rps_latitude, rps_plygrnd[*delt_dot  
s + i - 1].rps_longitude);  
208.        gene_dis3[i] = GPSDistance(rps_plygrnd[*delt_do  
ts + i].rps_latitude, rps_plygrnd[*delt_dots + i].rps_longitude, m  
id_lati, mid_longi);  
209.        dis_ratio = 0.001;  
210.  
211.        if (delta_heading_angle(basic_angle, di_angle[i  
]) > 75)  
212.        {  
213.            passbyhalf = 1;  
214.        }  
215.  
216.        if (delta_heading_angle(basic_angle, di_angle[i  
]) > 170)  
217.        {
```

```
218.                     out_way = false;
219.                     break;
220.                 }
221.
222.             i++;
223.         }
224.
225.         if (out_way == true)
226.         {
227.             i--;
228.         }
229.
230.         *delt_dots = *delt_dots + i;
231.         double dis_l = GPSDistance(end.kps_latitude, end.kps_lo-
ngitude, rps_plygrnd[*delt_dots].rps_latitude, rps_plygrnd[*delt_d-
ots].rps_longitude);
232.
233.         if (dis_l < distance_each_point_round)
234.         {
235.             rps_plygrnd[*delt_dots].rps_latitude = end.kps_
latitude;
236.             rps_plygrnd[*delt_dots].rps_longitude = end.kps_
longitude;
237.             rps_plygrnd[*delt_dots].rps_type = 1;
238.         }
239.     else
240.     {
241.         rps_plygrnd[*delt_dots].rps_latitude = (end.kps_
latitude + rps_plygrnd[*delt_dots - 1].rps_latitude) * 0.5;
242.         rps_plygrnd[*delt_dots].rps_longitude = (end.kp-
s_longitude + rps_plygrnd[*delt_dots - 1].rps_longitude) * 0.5;
243.         rps_plygrnd[*delt_dots].rps_type = 1;
```

```
244.             *delt_dots = *delt_dots + 1;
245.
246.             rps_plyrnd[*delt_dots].rps_latitude = end.kps_
247.                 latitude;
248.             rps_plyrnd[*delt_dots].rps_longitude = end.kps_
249.                 _longitude;
250.
251.             return 1;
252.         }
253.
254.     int generate_direct_road(float step, FourPoints strt, FourPoint
255.                               s end, int *delt_dots)
256.     {
257.         //按照逆时针行进方向
258.         if (!((strt.kps_type == 0 && end.kps_type == 1) || (strt
259.               .kps_type == 2 && end.kps_type == 3)))
260.         {
261.             //表示运行错误
262.             return 0;
263.         }
264.     }
265.
266.     //计算两点间距离, 进行划分
267.     double dis = 0.0;
268.     dis = GPSDistance(strt.kps_latitude, strt.kps_longitude
269.                       , end.kps_latitude, end.kps_longitude);
270.     //计算区域数量, 计算划分情况
271.     double minstep = dis / (step * 1.05);
```

```
271.         double maxstep = dis / (step * 0.95);
272.         int stepnumber = 0;
273.         //求差值
274.         //如果比较小
275.         if (maxstep - minstep >= 1.0)
276.         {
277.             stepnumber = (int)((maxstep + minstep) / 2);
278.         }
279.         else
280.         {
281.             stepnumber = (int)((maxstep) / 2);
282.         }
283.         //等分划分
284.         int i = 1;
285.         //划分后的距离(调试用)
286.         double gene_dis1[50] = {0.0};
287.         double gene_dis2[50] = {0.0};
288.
289.         //每一小段
290.         double lati_step = (end.kps_latitude - strt.kps_latitude) / stepnumber;
291.         double longi_step = (end.kps_longitude - strt.kps_longitude) / stepnumber;
292.         while (i < stepnumber)
293.         {
294.             rps_plygrnd[*delt_dots + i].rps_latitude = rps_plygrnd[*delt_dots + i - 1].rps_latitude + lati_step;
295.             rps_plygrnd[*delt_dots + i].rps_longitude = rps_plygrnd[*delt_dots + i - 1].rps_longitude + longi_step;
296.             rps_plygrnd[*delt_dots + i].rps_type = 0;
297.
```

```
298.             gene_dis1[i] = GPSDistance(rps_plygrnd[*delt_dot
ts + i].rps_latitude, rps_plygrnd[*delt_dots + i].rps_longitude, e
nd.kps_latitude, end.kps_longitude);
299.             gene_dis2[i] = GPSDistance(rps_plygrnd[*delt_dot
ts + i].rps_latitude, rps_plygrnd[*delt_dots + i].rps_longitude, r
ps_plygrnd[*delt_dots + i - 1].rps_latitude, rps_plygrnd[*delt_dot
s + i - 1].rps_longitude);
300.             i++;
301.         }
302.         //最后一个直接连接上
303.         rps_plygrnd[*delt_dots + i].rps_latitude = end.kps_latit
tude;
304.         rps_plygrnd[*delt_dots + i].rps_longitude = end.kps_lon
gitude;
305.         rps_plygrnd[*delt_dots + i].rps_type = 0;
306.
307.         *delt_dots = *delt_dots + i;
308.         return 1;
309.     }
310.
311.     int write_in(int delt_dots, double *dis, double *direction)
312.     {
313.         finished_gpspoints = delt_dots;
314.
315.         for (int i = 0; i < delt_dots; i++)
316.         {
317.             int16 hund = 0, ten = 0;
318.             //double 部分
319.             Points_Stored[i].lon_keypoint = rps_plygrnd[i].
rps_longitude;
320.             Points_Stored[i].lat_keypoint = rps_plygrnd[i].
rps_latitude;
```

```

321.
322.          //整数小数分离
323.          ten = (int)(rps_plygrnd[i].rps_latitude / 10);

324.          Points_Stored[i].lat_kypnt_z = ten * 10 + (int)
            (rps_plygrnd[i].rps_latitude / 1 - (ten * 10));
325.          Points_Stored[i].lat_kypnt_f = rps_plygrnd[i].r
            ps_latitude - Points_Stored[i].lat_kypnt_z;
326.          //整数小数分离
327.          hund = (int)(rps_plygrnd[i].rps_longitude / 100
            );
328.          ten = (int)(rps_plygrnd[i].rps_longitude / 10)
            - hund * 10;
329.          Points_Stored[i].lon_kypnt_z = hund * 100 + ten
            * 10 + (int)(rps_plygrnd[i].rps_longitude - (hund * 100 + ten * 1
            0));
330.          Points_Stored[i].lon_kypnt_f = rps_plygrnd[i].r
            ps_longitude - Points_Stored[i].lon_kypnt_z;
331.          //点类型
332.          Points_Stored[i].type_point = rps_plygrnd[i].rp
            s_type;
333.      }
334.  }
335.  //区域划分
336. void round_in_out()
337. {
338.     int last_type_tag = Points_Stored[0].type_point;
339.     //确定
340.     for (int i = 0; i < finished_gpspoints; i++)
341.     {
342.         //如果出现了 nan, 直接退出
343.         if (isnan(Points_Stored[i].lat_keypoint))

```

```
344.         {
345.             break;
346.         }
347.         //判断要不要切换
348.         if (last_type_tag != Points_Stored[i].type_poin
t)
349.         {
350.             if (Points_Stored[i].type_point == 1 &&
last_type_tag == 0)
351.             {
352.                 Points_Stored[i].type_point = R
OUND_IN_AREA;
353.                 Points_Stored[i - 1].type_point
= ROUND_IN_AREA;
354.                 Points_Stored[i - 2].type_point
= ROUND_IN_AREA;
355.                 last_type_tag = 1;
356.             }
357.             else
358.             {
359.                 Points_Stored[i].type_point = R
OUND_OUT_AREA;
360.                 Points_Stored[i + 1].type_point
= ROUND_OUT_AREA;
361.                 last_type_tag = 0;
362.                 i = i + 1;
363.             }
364.         }
365.         else
366.         {
367.             last_type_tag = Points_Stored[i].type_p
oint;
```

```
368.         }
369.     }
370.
371.     Points_Stored[0].type_point = ROUND_OUT_AREA;
372.     Points_Stored[finished_gpspoints - 1].type_point = ROUN
373.     D_OUT_AREA;
374.
```

## Time\_pit\_rtt.c(软件定时器)

```
1. #include "headfile.h"
2. #include "time_pit_rtt.h"
3.
4. uint32 total_time = 0;
5. rt_sem_t send_order;
6.
7. void time_pit_1ms()
8. {
9.     total_time++;
10.
11.    //每 2ms 处理一次陀螺仪数据
12.    if (total_time % 2 == 0)
13.    {
14.        //获取角速度计数据
15.        get_icm20602_gyro_spi();
16.        //直接过滤掉零飘值
17.        if (icm_gyro_z < 15 || icm_gyro_z > 15)
18.        {
19.            icm_gyro_z = 0;
20.        }
21.        //计算出此时 z 方向的角速度，单位为度每秒
22.        Data_Gyro.gyro_z = (float)(icm_gyro_z)*2000 / 32767;
```

```
23.  
24.          //一次陀螺仪积分累计  
25.          Data_Gyro.angle_sum_once = Data_Gyro.angle_sum_once - Data  
_Gyro.gyro_z / 500;  
26.          //后两个是用于刚发车时的陀螺仪控制，误差累计不大直接积分法可用  
27.          Data_Gyro.angle_sum_all = Data_Gyro.angle_sum_all - Data_G  
yro.gyro_z / 500;  
28.          Data_Gyro.angle_sum_area = Data_Gyro.angle_sum_area - Data  
_Gyro.gyro_z / 500;  
29.  
30.          //用于发车时确定起始航向  
31.          static int fst_flag = 0;  
32.  
33.          if (start_car > 0)  
34.          {  
35.              static double temp = 0.0;  
36.              //获取发车航向  
37.              if (fst_flag == 0)  
38.              {  
39.  
40.                  temp = GPSBearingAngle(Points_Use[0].lat_keypoint,  
Points_Use[0].lon_keypoint,  
41.  Points_Use[1].lat_keypoint,  
Points_Use[1].lon_keypoint) +  
42.  0.01;  
43.                  fst_flag = 1;  
44.              }  
45.  
46.              if (fabs(temp) > 1e-6)  
47.              {  
48.                  //陀螺仪法获得航向，仅仅用于第一段距离，长时间累计误差较  
大
```

```
49.             Data_Gyro.gyro_direction = fmod(temp + Data_Gyro.a  
    ngle_sum_all, 360.0);  
  
50.  
  
51.             if (Data_Gyro.gyro_direction < 0.0)  
52.             {  
53.                 Data_Gyro.gyro_direction = 360.0 + Data_Gyro.g  
    yro_direction;  
54.             }  
55.         }  
56.     }  
57.     else  
58.     {  
59.         fst_flag = 0;  
60.     }  
61. }
```

62.

```
63. //舵机频率为 50hz, 所以角度控制设定为 20ms 一次  
64. //速度限定是为了防止低速时无意义的打角  
65. if (total_time % 20 == 0 && gps_mode == 1 && (Save_Data.f_grou  
    nd_speed > 3.15))  
66. {  
67.     gps_servo_control();  
68. }  
69. //无线串口每 100ms 允许一次, 因为保障通行距离波特率值较低  
70. if (total_time % 100 == 0)  
71. {  
72.     rt_sem_release(send_order);  
73. }  
74. }  
75.  
76. //按键扫描线程初始化  
77. void time坑_1ms_init()
```

```
78. {
79.     rt_timer_t timer1;
80.
81.     //创建软件定时器
82.     timer1 = rt_timer_create("time_pit_1ms", time_pit_1ms, RT_NULL
83.                             , 1, RT_TIMER_FLAG_PERIODIC);
84.     //无线串口发送信号量创建
85.     send_order = rt_sem_create("send_order", 0, RT_IPC_FLAG_FIFO);
86.
87.     if (RT_NULL != timer1)
88.     {
89.         rt_timer_start(timer1);
90.     }
91. }
```

## GPS.c(GPS 数据处理)

```
1. void gps_deal_entry()
2. {
3.     while(1)
4.     {
5.         //等待开始计算的信号送达
6.         rt_sem_take(gps_calculation, RT_WAITING_FOREVER);
7.
8.         if (Save_Data.isUsefull == true && gps_mode == 1)
9.         {
10.
11.             if (passed_sections == 0 && Save_Data.f_ground_speed <
12.                 1.8 && 0)
13.                 {
14.                     //舵机固定中值，防止车辆起步产生偏移
15.                     pwm_duty_update(TIM_8,SERVO_PIN, pwm_mid_angle);
```

```
15.          //与第一个点的距离与航向计算，显示在 tft 屏幕上，为发车提
供参考
16.          error_lenth_s_p = GPSDistance(Save_Data.f_lati, Sa
ve_Data.f_longi, Points_Use[1].lat_keypoint, Points_Use[1].lon_key
point);
17.          error_direction = GPSBearingAngle(Save_Data.f_lati
, Save_Data.f_longi, Points_Use[1].lat_keypoint, Points_Use[1].lon
_keypoint);
18.      }
19.
20.
21.      if (analysis_finished == false && guiding_mode > 0)
22.      {
23.          //赛道位置微操
24.          micro_operating();
25.          //gps 自动规划
26.          // gps_auto_plannation();
27.          //赛道位置微操恢复
28.          micro_operating_recover();
29.          //路径规划结束
30.          analysis_finished = true;
31.      }
32.      else if (analysis_finished == false && guiding_mode <=
0)
33.      {
34.          //直接导航模式
35.          for (int i = 0; i < ALL_POINTS; i++)
36.          {
37.              Points_Use[i] = Points_Stored[i];
38.              analysis_finished = true;
39.          }
40.      }
```

```
41.  
42.          if (passed_sections <= run_sections && dot_read == false  
        && analysis_finished == true)  
43.          {  
44.  
45.              area_tag = Points_Use[passed_sections + 1].type_po  
              int;  
46.              //速度规划  
47.              if (passed_sections <= 1)  
48.              {  
49.                  g_speed_set = speed_max_region[3];  
50.                  setspeed_to_pwm();  
51.              }  
52.              else if (area_tag == DIRECT_AREA || area_tag == RO  
                UND_OUT_AREA)  
53.              {  
54.                  g_speed_set = speed_max_region[0];  
55.                  setspeed_to_pwm();  
56.              }  
57.              else if (area_tag == ROUND_AREA)  
58.              {  
59.                  g_speed_set = speed_max_region[1];  
60.                  setspeed_to_pwm();  
61.              }  
62.              else if (area_tag == ROUND_IN_AREA)  
63.              {  
64.                  g_speed_set = speed_max_region[2];  
65.                  setspeed_to_pwm();  
66.              }  
67.              //获取上个已通过的点的信息  
68.              Use_Data.lat_last = Points_Use[passed_sections].la  
              t_keypoint;
```

```
69.           Use_Data.lon_last = Points_Use[passed_sections].lon_keypoint;
70.           //获取下一个目标点的信息
71.           Use_Data.lat_next = Points_Use[passed_sections + 1].lat_keypoint;
72.           Use_Data.lon_next = Points_Use[passed_sections + 1].lon_keypoint;
73.           //两点间的航向
74.           Use_Data.TwoDotsDirec = GPSBearingAngle(Use_Data.lat_last, Use_Data.lon_last,
75.   Use_Data.lat_next, Use_Data.lon_next);
76.           //因为刚切换完点，之前的陀螺仪累计值清零
77.           Data_Gyro.angle_sum_once = 0;
78.           //表明已经读取过信息，避免代码重复执行
79.           dot_read = true;
80.       }
81.       //计算与下一个点的距离
82.       Use_Data.distan_next_point_now = GPSDistance(Save_Data.f_lati, Save_Data.f_longi,
83.   Use_Data.lat_next, Use_Data.lon_next);
84.       //计算与下一导航点的航向
85.       Use_Data.angle_next_heading_now = GPSBearingAngle(Save_Data.f_lati, Save_Data.f_longi,
86.   Use_Data.lat_next, Use_Data.lon_next);
87.       //当速度大于一定值时，
88.       if (Save_Data.f_ground_speed > 3.15)
89.       {
90.           //第一次发车，控制量复位
91.           if (start_car == 0)
```

```
92.         {
93.             start_car = 1;
94.             Data_Gyro.angle_sum_all = 0.0;
95.             Data_Gyro.angle_sum_area = 0.0;
96.             error_icm_last = 0.0;
97.         }
98.         //计算航向误差，进行角度控制
99.         GpsAngleError();
100.        //导航点切换
101.        GpsDotSelection();
102.        //陀螺仪角度累计归零
103.        Data_Gyro.angle_sum_once = 0.0;
104.
105.        error_icm_last = 0.0;
106.    }
107.    //如果导航点已经跑完，便可以自动停车了
108.    if (passed_sections >= run_sections && passed_sections > 0)
109.    {
110.
111.        g_speed_set = 0.0;
112.        setspeed_to_pwm();
113.    }
114.}
115. else if (gps_mode == -2)
116. {
117.
118.    gps_mode = 0;
119.
120.    distance_from_lastpoint = 0.00001;
121.
122.    now_area = 0;
```

```
123.  
124.         error_icm_last = 0.0;  
125.  
126.         error_lenth_s_p = 0.0;  
127.  
128.         passed_sections = 0;  
129.  
130.         start_car = 0;  
131.  
132.         dot_read = false;  
133.  
134.         area_tag = 0;  
135.  
136.         analysis_finished = false;  
137.     }  
138.     rt_thread_mdelay(20);  
139. }
```

140. }

141.

142.

```
143. void gps_deal_init()  
144. {  
145.     rt_thread_t gps_tid;  
146.  
147.     gps_tid = rt_thread_create("gps_deal", // 线程名称  
148.                             gps_deal_entry, // 线程入口函数  
149.                             RT_NULL, // 线程参数, RT_NULL 表示无参, 类似于 void  
150.                             1024 * 10, // 10K 个字节的栈空间, 留有一定的余  
量 (MM32F3277G9P 芯片的 RAM 空间有 128KB)  
151.                             17, // 线程优先级为 17, 数值越小, 优先级越高。  
152.                             50); // 时间片为 50ms  
153.
```

---

```
154.     //启动显示线程
155.     if(RT_NULL != gps_tid)
156.     {
157.         rt_thread_startup(gps_tid);
158.     }
159. }
```

## 附录 B.参考文献

- [1]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.
- [2]陈瑞雪,王宣怀,王庭琛.实时操作系统 RT-Thread 启动流程剖析[J].现代电子技术,022,45(12):36-42.

[3]吴畏. 基于高精度 GPS 的智能驾驶车辆自主循迹系统的设计与研究[D].长安大学,2021.

[4]田园. 基于 GPS 与自主定位的智能车循迹算法研究与验证[D].安徽工程大学,2020.

[5]许明宇,王宣怀,汪恒.面向 ARM Cortex-M 系列 MCU 的 RT-Thread 驻留方法[J].现代电子技术,2022,45(04):7-12.