32小车模板 - 保姆级使用说明

1. 简要介绍

本代码模板专为 **STM32F4** 智能小车开发设计，旨在提供一个清晰、模块化、易于扩展的开发框架。

核心特点：

- **分层架构**：底层驱动（Drivers/Core）与应用逻辑（My_App）分离，互不干扰。
- **时间片调度**：内置轻量级任务调度器（`Scheduler_Task`），告别杂乱的 `delay` 和 `while(1)` 裸奔。
- **模块化功能**：已封装电机控制、PID算法、编码器读取、按键处理、OLED显示、灰度巡线等常用功能。
- **开箱即用**：只需关注上层逻辑，无需重复造轮子。

2. 上层 API 开发调用说明

所有应用层代码位于 `My_App` 文件夹中。请根据下表快速查找所需功能：

模块 (Module)	核心文件 (File)	核心 API (Function/Variable)	说明 (Description)
调度器 (Scheduler)	scheduler.h	Scheduler_Init() Scheduler_Run()	**核心引擎。** Init用于初始化，Run放入main的while(1)中自动调度任务。
电机 (Motor)	motor_app.h	Motor_Init() Motor_Task()	**运动控制。**控制电机PWM占空比与方向。Task需放入调度器周期运行。
PID (PID)	pid_app.h	PID_Init() PID_Task()	**闭环算法。**计算速度/位置环输出。全局变量 basic_speed 控制基础速度。
编码器 (Encoder)	encoder_app.h	Encoder_Init() Encoder_Task()	**速度反馈。**初始化定时器编码器模式，Task定时读取并计算当前速度。
按键 (Key)	key_app.h	button_system_init() Key_Task()	**交互输入。**支持单击、长按及组合键（如 BTN_COMBO_ID）。
OLED (OLED)	oled_app.h	Oled_Task()	**屏幕显示。**定时刷新屏幕内容，用于显示调试参数或状态。
灰度 (Gray)	gray_app.h	Gray_Task() extern uint8_t Digital	**循迹传感器。** Task读取7路灰度状态，结果存入 Digital 变量供逻辑判断。
串口 (UART)	uart_app.h	u1_printf(...) UART_RX_BUFFER_SIZE	**通信调试。**支持格式化打印（类似printf）。接收缓冲区大小可调。
陀螺仪维特系列 (Gyro)	uart_app.h	extern WIT_Data_t wit_data	**姿态解算。**读取 wit_data.pitch/roll/yaw 获取欧拉角，ax/ay/az 获取加速度。

> **提示**：

- **初始化**：所有 `_Init` 函数只需在 `main` 的 `BEGIN 2` 区域调用一次。
- **任务运行**：所有 `_Task` 函数需添加到 `scheduler.c` 的任务列表中，并设置合适的周期（如电机/PID建议 `10ms-20ms`）。

---

3. 🔌 硬件引脚分配（Hardware Pinout）

开发前请核对以下引脚连接，确保硬件与代码匹配：

功能 (Function)	引脚 (Pin)	端口 (Port)	备注 (Note)
按键 (Keys)	MODE1	PE3	模式切换键 1
	MODE2	PE4	模式切换键 2
	MODE3	PE1	模式切换键 3
电机 (Motor)	AIN1 / AIN2	PE12 / PE13	左电机方向控制
	BIN1 / BIN2	PE15 / PE14	右电机方向控制
	STBY	PE8	电机驱动待机控制
灰度 (Gray)	Gray 0-1	PC4, PC5	循迹传感器
	Gray 2-3	PC8, PC9	循迹传感器
	Gray 4-5	PD10, PD11	循迹传感器
	Gray 6-7	PD4, PD7	循迹传感器
其他 (Misc)	W25QXX_CS	PA4	Flash 片选

---

3. 🚀 快速上手 (Quick Start)

按照以下步骤，在 5 分钟内运行起你的小车代码！

第一步：硬件配置 (CubeMX)

确保 `Core/Src` 中的底层初始化（GPIO, TIM, UART等）已通过 CubeMX 生成并匹配你的硬件引脚。

- **电机PWM**：对应定时器通道。
- **编码器**：定时器 Encoder Mode。
- **按键/LED**：GPIO Input/Output。

第二步：集成调度器 (Main.c)

打开 `Core/Src/main.c`，在主函数中添加调度器代码：

```
/* USER CODE BEGIN Includes */
#include "scheduler.h" // 引入调度器头文件
/* USER CODE END Includes */

int main(void)
{
  // ... (HAL_Init, SystemClock_Config 等自动生成的代码) ...

  /* USER CODE BEGIN 2 */
  Scheduler_Init(); // 1. 初始化调度器
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    Scheduler_Run(); // 2. 运行调度器 (不要在 while(1) 里写其他阻塞代码!)
    /* USER CODE END WHILE */
  }
```

```
}
```

第三步：配置任务（Scheduler.c）

打开 `Scheduler_Task/scheduler.c`，在 `scheduler_task` 数组中添加你要运行的任务：

```
// 任务列表定义
scheduler_task_t scheduler_task[] = {
    // {任务函数指针, 执行周期(ms), 上次运行时间(初始0)}
    {Key_Task,      10,  0}, // 每10ms扫描一次按键
    {Encoder_Task,  10,  0}, // 每10ms读取一次编码器
    {PID_Task,      20,  0}, // 每20ms计算一次PID
    {Motor_Task,    10,  0}, // 每10ms刷新一次电机输出
    {Oled_Task,     100, 0}, // 每100ms刷新一次屏幕
    {Gray_Task,     10,  0}, // 灰度巡线任务
};
```

注意：任务周期越短，优先级越高（响应越快），但不要超过*MCU*处理能力。

第四步：编写业务逻辑

进入 `My_App` 文件夹，修改对应 `.c` 文件。

- **想改PID参数？** 去 `pid_app.c` 修改 `Kp, Ki, Kd`。
- **想改按键功能？** 去 `key_app.c` 修改 `Key_Task` 中的逻辑。
- **想显示新数据？** 去 `oled_app.c` 修改 `Oled_Task`。

第五步：编译下载

点击 Keil 的 **Build (F7)**，确保 `0 Error 0 Warning`，然后 **Download (F8)**。
观察小车是否按预期运行！

开发小贴士

1. **不要使用 `HAL_Delay()`**：在任务函数中尽量避免使用延时，这会阻塞整个调度器。
2. **全局变量**：跨文件变量请在 `.h` 中用 `extern` 声明，在 `.c` 中定义。
3. **调试**：遇到问题优先看 OLED 显示，或者使用 `printf` 通过串口打印调试信息。

祝你电赛夺奖！