STM32F407VET6 云台控制系统源码使用说明书（保姆级教程）

简介： 本工程基于 STM32F407VET6 开发板，专为云台及电机控制开发设计。采用高级分层架构，内含闭环步进电机驱动、OLED显示、Flash参数存储及PID控制算法，适合电赛、智能车竞赛及嵌入式学习。

1. 代码架构梳理（文件在哪？）

打开工程，你会在左侧 Project 栏看到清晰的分层结构：

- **My_App（应用层 - 你的主战场）**
  - scheduler.c： **任务调度器**。相当于操作系统的核心，决定什么时候执行哪个任务（比如每10ms控制一次电机）。
  - pid_app.c： **PID 控制逻辑**。在这里修改电机的目标角度、速度，调整 PID 参数（Kp，Ki，Kd）。
  - key_app.c： **按键逻辑**。处理按键的长按、短按，用于切换模式或保存参数。
  - oled_app.c： **屏幕显示逻辑**。决定 OLED 屏幕上显示什么内容（如当前角度、模式）。
  - flash_app.c： **数据存储**。负责将 PID 参数保存到芯片内部 Flash，断电不丢失。
  - uart_motor.c： **电机通信解析**。处理与电机驱动板的数据交互协议。
  - interrupt.c： **中断回调**。统一管理串口接收中断等。
  - parse_app.c： **数据解析**。用于解析上位机或串口指令（如果有）。
- **My_Driver（驱动层 - 硬件搬运工）**
  - Emm_V5.c： **Emm_V5 闭环步进电机驱动**。封装了所有电机指令，是你最常用的文件。
  - oled_hardware_i2c.c： **OLED 底层驱动**。负责 I2C 通信波形。
- **Components（组件层 - 通用工具）**

    - `Pid`：**PID 算法库**。纯数学计算，不仅能控电机，还能控温、控平衡。
    - `Oled`：**绘图库**。提供画点、画线、写字等基础绘图功能。
- **Core（底层配置）**
    - `main.c`：**程序入口**。主要负责硬件初始化，然后启动调度器。
    - `gpio.c`，`tim.c`，`usart.c`：CubeMX 生成的底层硬件配置。

2. 硬件接口大全（别接错线！）

请严格按照下表连接硬件，否则可能导致无法通信或屏幕不亮。

2.1 核心外设接口

外设模块	功能	引脚/外设号	详细说明
调试器	程序下载与调试	**PA13（SWDIO）** **PA14（SWCLK）**	使用 ST-Link 或 J-Link 连接
OLED 屏幕	显示系统状态	**PB6（SCL）** **PB7（SDA）**	I2C1 接口，3.3V 供电
闭环电机（Emm_V5）	电机通信控制	**PA9（TX）** **PA10（RX）**	USART1 接口，连接驱动板的 RX/TX（交叉连接）
按键（KEY）	交互输入	**PE3（KEY1）** **PE4（KEY2）**	默认下拉，按下为高电平（根据原理图确认）
LED 指示灯	运行状态指示	**PA6 / PA7**	可用于指示系统心跳或错误
Flash 存储	参数掉电保存	**片内 Flash**	使用芯片内部扇区，无需外接芯片

2.2 系统时钟与定时器

模块	功能	配置	说明
TIM3	任务调度时钟	1ms 中断	为 Scheduler 提供时间基准
USART1	电机串口	115200 bps	开启 DMA 接收（提高效率）

3. 功能函数调用指南（复制粘贴即用）

这里列出了你开发中**真正会用到**的所有函数。

3.1 闭环步进电机控制（Emm_V5.h）

包含头文件： `#include "Emm_V5.h"`

- **初始化电机**
  - `void Emm_V5_Init(UART_HandleTypeDef *huart)`
  - **用法**：`Emm_V5_Init(&huart1);` // 在初始化代码中调用一次
  - **作用**：绑定控制电机的串口。
- **位置控制（最常用）**
  - `void Emm_V5_Pos_Control(uint8_t id, uint8_t dir, uint16_t vel, uint8_t acc, uint32_t clk)`
  - **参数**：
    - `id`：电机ID（默认1）。
    - `dir`：方向（0:逆时针，1:顺时针）。
    - `vel`：速度（RPM）。
    - `acc`：加速度（0-255，0为最大）。
    - `clk`：脉冲数（细分后，例如3600脉冲转一圈）。
  - **用法**：`Emm_V5_Pos_Control(1, 1, 500, 0, 1800);` // ID1电机，正转，速度500，转半圈(假设一圈3600)
- **速度控制（转个不停）**
  - `void Emm_V5_Vel_Control(uint8_t id, uint8_t dir, uint16_t vel, uint8_t acc)`
  - **用法**：`Emm_V5_Vel_Control(1, 0, 200, 0);` // ID1电机，反转，速度200RPM持续转动

- **停止控制**
  - `void Emm_V5_Stop_Now(uint8_t id)`
  - **用法**：`Emm_V5_Stop_Now(1); // 立即急停`
- **同步控制（多电机同时动作）**
  - `void Emm_V5_Synchronous_Control(uint8_t id, uint16_t vel, uint16_t acc, uint32_t clk, uint8_t dir)`
  - **用法**：用于多个电机需要严格同步启动到达位置的场景。
- **读取编码器/位置**
  - `void Emm_V5_Read_Encoder(uint8_t id)`
  - **用法**：`Emm_V5_Read_Encoder(1); // 发送读取指令，结果在串口接收中断中处理`

3.2 PID 算法控制（pid.h / pid_app.h）

包含头文件： `#include "pid.h"`

- **初始化 PID 对象**
  - `void PID_Init(PID_TypeDef *pid, float kp, float ki, float kd, float i_max, float out_max)`
  - **用法**：`PID_Init(&MotorPID, 1.5f, 0.01f, 0.5f, 500, 2000);`
  - **作用**：设置P、I、D参数以及积分限幅、输出限幅。
- **重置 PID**
  - `void PID_Reset(PID_TypeDef *pid)`
  - **用法**：`PID_Reset(&MotorPID); // 清除积分累积，防止系统震荡，常在停止时调用。`
- **PID 计算（核心）**
  - `float PID_Calc(PID_TypeDef *pid, float ref, float fdb)`
  - **参数**：`ref`（目标值），`fdb`（当前实际值）
  - **返回值**：控制量（如PWM值或速度值）。
  - **用法**：通常在定时器中断或调度任务中调用，如 `float output = PID_Calc(&MotorPID, target_angle, current_angle);`
- **修改 PID 参数**
  - `void PID_Set_Param(PID_TypeDef *pid, float kp, float ki, float kd)`

- **用法**：`PID_Set_Param(&MotorPID, 2.0f, 0.1f, 0.6f); // 动态调参`

3.3 OLED 显示（oled.h / oled_app.h）

包含头文件： `#include "oled.h"`

- **基础显示**
  - `OLED_Init(void)`：初始化屏幕。
  - `OLED_Clear(void)`：清屏。
  - `OLED_ShowString(uint8_t x, uint8_t y, uint8_t *str, uint8_t size)`：显示字符串。
  - `OLED_ShowNum(uint8_t x, uint8_t y, uint32_t num, uint8_t len, uint8_t size)`：显示整数。
  - `OLED_ShowFloat(uint8_t x, uint8_t y, float num, uint8_t len, uint8_t size)`：显示小数。
  - **用法**：

```
OLED_ShowString(0, 0, "Mode: PID", 16);
OLED_ShowFloat(0, 2, current_angle, 2, 16); // 显示当前角度，保留2位小数
```

3.4 Flash 参数存储（flash_app.h）

包含头文件： `#include "flash_app.h"`

- **保存参数**
  - `void Flash_Write_Data(void)`
  - **用法**：`Flash_Write_Data(); // 将当前的PID参数写入内部Flash，通常在调整完参数按下确认键时调用。`
- **读取参数**
  - `void Flash_Read_Data(void)`
  - **用法**：`Flash_Read_Data(); // 上电初始化时调用，从Flash恢复上次保存的参数。`

3.5 任务调度（scheduler.h）

包含头文件： `#include "scheduler.h"`

- **运行调度器**
  - `void Scheduler_Run(void)`
  - **用法**：直接放在 `main.c` 的 `while(1)` 循环中。它会自动管理所有任务。
- **添加/修改任务**
  - **操作**：直接修改 `My_App/scheduler.c` 中的 `Scheduler_Task_List` 数组。
  - **格式**：`{函数名, 周期(ms), 0}`
  - **示例**：

```
static scheduler_task_t Scheduler_Task_List[] =
{
    {PID_Control_Task, 10,  0}, // 每10ms执行一次PID控制
    {OLED_Show_Task,   100, 0}, // 每100ms刷新一次屏幕
    {Key_Scan_Task,    20,  0}, // 每20ms扫描一次按键
};
```

4. 快速上手步骤

1. **接线**：按第2节表格接好 OLED、电机和电源。
2. **供电**：确保电机驱动板单独供电（12V/24V），STM32 板由 USB 或 5V 供电。
3. **烧录**：使用 Keil 打开 `.uvprojx` 工程，点击 "Rebuild"，然后 "Download"。
4. **观察**：
   - OLED 应该亮起并显示主界面。
   - LED 灯应该按设定频率闪烁（证明系统在运行）。
5. **控制**：

    - 按下 **KEY1** (PE3)，观察电机是否开始运动或模式切换。
    - 按下 **KEY2** (PE4)，观察是否进入参数调整界面。

5. 常见问题排查

- **编译报错?** -> 检查 Keil 版本是否在 v5.30 以上，是否安装了 STM32F4 的 Device Pack。
- **电机只震动不转?** -> 检查电机线序 (A+/A-/B+/B-) 是否正确，或供电电流是否不足。
- **串口无数据?** -> 检查 PA9/PA10 是否交叉连接 (TX接RX，RX接TX)。
- **Flash 保存失败?** -> 确保你在 `flash_app.c` 中指定的 Flash 地址没有被程序代码覆盖 (地址通常在 Flash 也就是 ROM 的末尾)。

祝您开发顺利!