

目录

1 代码说明.....	1
1.1 Cube 生成部分	1
1.2 驱动部分.....	1
1.3 全局配置.....	2
1.3.1 Main 函数	2
1.3.2 任务开启.....	2
1.4 主要任务.....	2
1.4.1 Gimbal Task	2
1.4.2 Chassis Task	3
1.4.3 Detect Task.....	3
2 自定义程序.....	4
2.1 模块间通信.....	4
2.1.1 CAN 通信	4
2.1.2 串口通信.....	4
3 注意事项.....	5
3.1 调试注意的问题.....	5
4 PID 参数调试	5
4.1 云台 pid 部分.....	5
4.2 底盘 pid 部分	6

1 代码说明

代码使用 STM32Cube 软件生成的 HAL 驱动库 (hardware abstract library) 和 freertos (作为 HAL 库的中间层) 为框架, cube 使用详见 [stm32cube_usage](#) 文档。

1.1 Cube 生成部分

keil 工程中除了 Application/User 中的 task 部分, 以及 bsp 中的 RM 开发板相关驱动, 其他均为 cube 软件生成的代码。除非需要修改开发板硬件配置, 一般情况下不需要修改这些自动生成的代码。用户要将自己的代码添加到这些生成代码中时, 需要添加到特定的位置, 例如:

```
/* USER CODE BEGIN 0 */  
  
//add your code  
  
/* USER CODE END 0 */
```

这样是防止使用 cube 修改工程, 重新生成代码时, 用户代码被覆盖。

1.2 驱动部分

这部分的代码都在工程的 bsp 目录下, 包含 RM 开发板硬件相关的模块和对应的 API。每一项 bsp 的功能及其实现函数, 在程序中都有相关的注释, 可以根据自己的需求修改相应内容。在完成高中生夏令营的任务部分, 除了 bsp_can 或者 bsp_uart 中需要添加机械臂控制模块的通信外, 其他部分不需要修改。以下为各模块的简介。

kb.c:

用于键盘和鼠标数据读取, 并包含键盘速度控制的斜坡。

bsp_can.c:

提供 CAN 消息发送 API 与全局 CAN 接收中断回调, 所有模块的 CAN 反馈数据在这里读取。

bsp_flash.c:

提供给 calibrate.c 使用, 包含往片内 flash 写入校准数据的 API, 如云台校准, imu 校准数据。

bsp_uart.c:

提供 uart 接收中断回调, 如遥控器、裁判系统、manifold、TX1 等设备的通信。

pid.c:

整个车子控制所使用的 pid 初始化、计算等部分。

mpu.c:

读取板载 IMU mpu6500 和 ist8310 数据，作为云台角速度的反馈。

calibrate.c:

提供校准设备的 api，如云台中点校准、陀螺仪校准。

judge_sys.c:

包含裁判系统的数据帧信息、通信协议、接口等，以及读取裁判系统 api。

1.3 全局配置

1.3.1 Main 函数

初始化了 pwm, tim, usart, can 等等，并从 flash 读取校准数据存到全局结构体变量 gAppParam，然后初始化 freertos 和 task。

目前使用 3 个 task:

- Gimbal 主要用于云台、摩擦轮、拨弹电机控制；
- Chassis 用于底盘的控制；
- Error 用于模块离线检测与报警任务。

1.3.2 任务开启

在 freertos.c 文件中，osThreadDef()和 osThreadCreate()函数定义和创建并行任务。不同的任务在程序运行时互不影响，但可以通过访问全局变量，来进行通信和控制。

1.4 主要任务

1.4.1 Gimbal Task

- 1、初始化云台相关 pid 结构体和参数；
- 2、初始化 mpu6500 和 ist8310；
- 3、从 flash 中读取校准参数（云台的中点数据）；
- 4、之后进入任务循环，控制频率 5ms

LOOP:

- 1) imu 温度的 pid 控制，保持 imu 数据的稳定；
- 2) 读取 mpu6500 数据，用于 gimbal 角速度闭环控制；
- 3) 根据射击命令，处理射击任务；
- 4) 云台模式的切换，以及云台控制的状态机，主要是云台 pitch、yaw 轴控制的目标角度变化；
- 5) 计算云台电机 PID，然后根据目前控制模式和各模块状态，判断是否输出到云台电机。

下面为云台的几个控制模式（高中生使用的这版代码中，只有云台单轴闭环模式）：

GIMBAL_INIT：云台初始化并回到中间，此时底盘静止；

GIMBAL_CLOSE_LOOP_ZGYRO：yaw 轴以固定在自身的单轴陀螺仪模块为位置反馈；

GIMBAL_RELAX：切断云台电机输出。

1.4.2 Chassis Task

- 1、初始化底盘电机速度闭环 PID；
- 2、初始化底盘跟随云台的角度闭环 PID；
- 3、设置底盘跟随 PID 的一些特殊参数，例如死区（防止抖动），超出误差范围切断输出（防止大误差情况下的自旋）；
- 4、等待一段时间（陀螺仪复位），然后进入任务循环（10ms）

LOOP:

- 1) 获得遥控和键盘数据；
- 2) 获得 chassis mode，以及获得不同模式下底盘的 vx，vy，vw 的值；
- 3) 通过速度合成，获取 4 个轮子的电机速度；
- 4) 判断当前各个模块是否正常是否在线，如果存在错误（遥控丢失）或者处于保护/静止模式，清零输出；
- 5) 将电流值分别发送给 4 个电调；

底盘的几个模式（高中生程序只有底盘跟随云台模式有效）：

CHASSIS_CLOSE_GYRO_LOOP：底盘跟随底盘上的单轴陀螺仪；

CHASSIS_FOLLOW_GIMBAL_ENCODER：底盘跟随云台；

CHASSIS_AUTO/CHASSIS_OPEN_LOOP：底盘开环，如云台跟随底盘模式。

1.4.3 Detect Task

这部分主要通过检测上一次收到模块数据的时间与当前系统时间之间间隔来判断

模块是否离线，如果离线，根据优先级进行相应出错处理。

- 1、初始化全局结构体 `gRxErr` 的列表；
- 2、到各个模块的接收中断中插入 `err_detector_callback()`；
- 3、进入循环检测，频率 50ms

LOOP:

- 1) 遍历 `err-list[ErrorListLength]`，如果存在 `delta time > set timeout`。就说明存在离线。如果多个模块离线，依次按照错误提示优先级对错误进行排序，取得当前错误指针；
- 2) 按照指针指向通过蜂鸣器/led 来提示错误。

`err_detector_callback()`需要放在收到数据后调用，用于记录模块收到数据的系统时间。一般在 can 接收中断中调用，输入参数为模块 id。

2 自定义程序

RM 主控板安装在云台上，直接接上舵机线下来控制机械臂，如果线很多的话，可能会不太方便。所以大家可能会使用多个控制板，或者模块来进行整个车（包括取弹机构）的控制，这就涉及到了多个板子之间的通信。

2.1 模块间通信

在主控程序中给大家提供了 `bsp_can.c` 和 `bsp_uart.c` 的驱动程序，其中包含相应通信的 api，考虑到使用、编程以及接线难易等，推荐大家使用 can 来进行模块的通信。

2.1.1 CAN 通信

如果需要接受 can 模块发送过来的数据，只需要在 `HAL_CAN_RxCpltCallback()`函数中加入接受到相应 can id 信息后的处理程序即可。

同样如果需要发送数据，在填充完 can 发送结构体 `*pTxMsg` 后，使用 hal 库自带的 `HAL_CAN_Transmit()`函数发送。

2.1.2 串口通信

通过 dma 接收数据，但只是开启串口空闲中断，即只进入 `USARTx_IRQHandler()`函数，不会进入 `DMA1_Stream1_IRQHandler()`这类函数。

uart 空闲中断时会调用 `USARTx_IRQHandler()`函数，在里面加入一个自定义的数据

帧接收解析和重启 DMA 的函数 `MyUartFrameIRQHandler(&huartx)`，这个函数完成对接收缓冲区的数据解析，符合协议则使用，否则丢弃。这里同时也提高系统接收数据的鲁棒性，防止接收的数据不完整造成的整体解析出错。

如果串口需要处理 `float` 类型或者多个混合类型的数据，可以先组合成结构体，然后使用内存拷贝到 `buffer` 中直接发送，接收的时候重新组合即可（注意大小端）。

3 注意事项

3.1 调试注意的问题

刚装完车，下载完代码，上电之后有两件事情需要注意一下

1、机械角的记录：

在刚刚完成云台的安装后，需要记录一下云台中点的机械角，即云台在中点时绝对编码器的位置。将云台扶到中点，然后在调试界面，将 `gAppParam.Gimbal.NeedCali` 参数的值设为 1 即可，这时的编码器的值会被写进 `flash`。

2、云台不受控制：

在没有改代码的情况下，云台不受控制，这时很可能是正反馈的问题。首先需要改一下位置环的输出即速度环输入的符号，如果没有恢复正常，则可能是陀螺仪的正反馈，需要改速度环角速度反馈的符号。正反馈可能是由多个因素引起的，如单轴的正反、电机的电调等等。

4 PID 参数调试

一般情况下使用现有的默认 `pid` 参数也能让车子动起来，如果想要车子的响应达到自己想要的效果，或者是更改、加入了自己的控制代码，就需要调节相应的 `pid` 参数了。

4.1 云台 `pid` 部分

在 `gimbal_pid_init()` 函数中，有云台 `pid` 的默认参数。其中有 `pitch` 轴速度、位置，`yaw` 轴速度、位置这四个闭环控制参数，注意在调参时，先调内环速度环，再调节外环位置环。另外还有一个陀螺仪温度控制的 `pid` 参数，可以不用动它。

一般情况下，只需通过 `P` 项即可实现云台的控制，`P` 越大，云台的响应越快，但 `P` 过大时会产生超调，这时云台会抖动，减小 `P` 项即可消除。

积分项 `I` 主要用于消除静态误差，对于精度有要求的地方需要加上 `I` 项，比如自动

射击，打大符等。

微分项 D 项在云台的控制中可以不加，D 相当于一个阻尼的作用，防止过调。

4.2 底盘 pid 部分

底盘 pid 的默认参数都在 `chassis_pid_param_init()` 函数中。其中设置底盘四个轮子的速度环 pid 参数相同，和云台同理，只用设置比例项 P 即可。

在底盘跟随云台时，通过设置跟随闭环 `pid_chassis_angle` 的 P 项，来提高底盘跟随的响应速度。为了防止云台在静止时底盘的轻微抖动，需要在跟随闭环上加上死区 `deadband`，大小根据实际情况确定。同时为了防止云台和底盘角度偏差过大，车子陷入疯转的情况，需要设置最大误差 `max_err`，默认设置角度为 60 度。