步兵二代 代码说明

Langgo

Version	Date	Note
1.0	2016-12-22	记录基本框架和主要思路。
1.1	2017-1-18	删除了 kb_task,改为 hook
		修复了若干 bug。
		新增一点点说明。

软件/库版本

Keil5	-uVision V5.21	F4-DFP=V2.6
CubePackage	STM32Cube_FW_F4_V1.13.0	
STM32CubeMX	4.16	
MCU	STM32F427IIH6	

代码说明：

Bref：

代码使用 ST 的 STM32Cube 软件 + HAL 驱动库（hardware abstract library） + freertos（作为 HAL 库的中间层。）为框架。

需要开发者参与的部分主要分为 hal_driver（cube 软件生成），bsp，task 三部分。

Bsp 部分：包含硬件相关的模块和 API，

例如 pid.c 提供 pid_t 类型的 pid 结构体定义，float pid_calc(pid_t* pid, float fdb, float ref);函数。通过调用

```
void PID_struct_init(
    pid_t* pid,
    uint32_t mode,
    uint32_t maxout,
    uint32_t intergral_limit,

    float    kp,
    float    ki,
    float    kd);
```

来初始化 pid_t 类型的结构体。

之后就可以使用

float pid_calc(pid_t* pid, float fdb, float ref);实现 pid 计算得到输出。

其他同理。相关.h 文件一般对 API 说明的很清楚。

以下为各模块的简介。

mpu.c	提供板载 IMU mpu6500 和 ist8030 IC 的读取，主要作为云台角速度的获取。
bsp_can.c	提供 CAN 消息发送 API 与全局 CAN 接受中断回调 void HAL_CAN_RxCpltCallback(CAN_HandleTypeDef* _hcan)。大部分模块的 CAN 反馈报文在这里读取与预处理。
Bsp_flash.c	提供给 calibrate.c 使用。包含往片内 flash 写入校准数据的 IO API，例如云台校准，imu 校准数据
Bt.c	提供更高一级的 uart 接口功能 API，如蓝牙，遥控器 DBUS，裁判系统接口
Arhs.c	提供 imu 的姿态解算功能，这部分功能暂时还不能使用。
Calibrate.c	提供校准设备的 Hook API， 一般在读取设备数据之后使用。

Task 相关

Main 函数：

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

一共四个 task 简介

Gimbal	云台、摩擦轮、拨单电机、控制
Chassis	底盘移动与 3510 电机控制
Error	用于模块离线错误检测与报警的任务。

Kb	用于 PC 按键判断与处理

以下分别介绍 task 以及控制流程。

Gimbal Task：

1、在 task 初始化阶段初始化 pid 结构体和参数。

2、初始化 mpu。

3、相关变量预处理。。

4、之后进入任务循环，循环间隔 5ms

While（1）{

1， imu 温度的 pid 控制。保持 imu 期间的稳定。

2， 读取 mpu 的 gyro，用于 gimbal 角速度闭环控制。

3， 对 pit。Yaw 轴的分别积分型位置控制输入和限幅。遥控器+鼠标键盘模式。

pit 轴比较容易处理。Yaw 要复杂一些。

其中 Yaw 分为两种模式。

《1》Yaw encoder mode：yaw 以自身的电机编码器为位置反馈做闭环。这种方式最简单，最直观。

《2》Yaw zgyro mode：yaw 以固定在自身的 RM-陀螺仪模块为位置反馈。这种方式需要考虑陀螺仪上电校准时不能被移动/碰撞。为了保证步兵车能走直线，主要需要依靠这种模式。

同时，这里涉及一些数据换算，以函数说明。

s16 get_relative_pos(s16 raw_ecd, s16 center_offset)；获取相对于云台校准角度后的位置。例如 pit 和 yaw 都在中间时，获得校准值 pit-offset，yaw-offset，则经过该函数返回的编码器值，当云台回中时，得到的数值就是 0,然后顺时针为正 逆时针为负。

4， 根据遥控器（也就是 rc）的右拨动开关选择 yaw mode。如果是 zgyro mode，需要加一些特殊处理，例如从其他 mode 切换进来时要先进入 zgyro 校准模式。

5， 根据鼠标和左拨码开关来处理摩擦轮和拨弹电机。

其中拨弹电机使用速度+位置双闭环控制。

6， 计算 gimbal double axis PID， double loop control。

思路是：根据位置反馈计算得到 pid-pit.pos_out,然后将 pos_out 作为速度环的参考，计算 pid_pit_omg，最终得到的输出当做 iq，发送到电机去。

根据目前控制模式和各模块状态，判断是否输出到 gimbal motor。

//end of gimbal task loop，goto next time loop

}

Chassis task：

先介绍一些用到的函数。

reset_zgyro()	复位陀螺仪。复位之后要注意必须使陀螺仪静止至少 2s
move_measure_hook()	用于根据麦轮电机上的编码器测量步兵底盘移动的坐标。
mecanum_calc（。。。）	输入底盘移动的 vx，vy，omega，得到解算后 4 个轮子的速度。
get_chassis_mode_set_ref	根据遥控器来获得 chassis mode 和 vx vy。（之后这里会改）

具体实现以及参数输入输出都在 API 上写了。请看代码。

Task 任务：（10ms 控制周期）

1、 先初始化底盘速度闭环的 pid 结构体数组 pid_spd[4]。

2、 初始化底盘跟随云台的角度闭环 pid 结构体 pid_chassis_angle。

3、 设置底盘跟随的 pid 的一些特殊参数，例如死区，超出误差范围就切断输出，这是一个保护措施，防止步兵车因为超大误差引起的疯狂自旋。

4、 复位陀螺仪 等待一段时间 然后就可以开始进入任务循环。

While（1）{

1．获得 chassis mode 和 输入 vx，vy，o mega 等速度指令。

2．插入一个测量底盘移动的 hook。不过暂时用不到。

3．判断底盘 mode， 目前其实只有一种 mode，就是底盘跟随云台的编码器 0 值，即 CHASSIS_FOLLOW_GIMBAL_ENCODER。这样云台转到哪儿，底盘就跟到哪儿。按照这个思路，

chassis.omega = pid_calc(&pid_chassis_angle, yaw_relative_pos, 0);这里通过控制 chassis omega 使底盘跟随云台编码器 0 实现角度闭环。

4. 调用 mecanum_calc(chassis.vx, chassis.vy, chassis.omega,
MAX_WHEEL_SPEED, chassis.wheel_speed.s16_fmt);
获取 4 个轮子的电机速度。

5．计算底盘电机的速度闭环 pid，速度反馈由 820R 电调得到，输入速度即为经过麦轮解算函数 mecanum_calc 得到的速度。计算得到输出到电调的 iq。

6.判断当前各个模块是否正常是否在线，判断当前 mode 是否可以输出，如果存在错误或者处于保护/静止模式，就把 iq 清零。

7 通过 can 发送 iq 到 820R 驱动电机。

}

Error 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 delta_time 来判断模块是否离线/timeout err/TOE。有几个主要的 API 如下。

U32 HAL_GetTick();	获取 system time bask。
void err_detector_callback(int err_id)	这是一个 callback。应该放在收到数据后调用，用于记录模块收到数据的 system time。一般在 CAN-RX-IT 中调用，输入参数为模块 id 即可，例如 GimbalYawTOE。
global_err_detector_init();	用于初始化全局错误检测变量的函数。

以下是主要的错误检测结构体。4 字节对齐。使用了 bit filed。

```
#pragma pack(4)
typedef struct{
    volatile uint32_t last_time;    //记录上次时间
    volatile uint32_t err_exist     :1;   //1 = err_exist, 0 = everything ok
    volatile uint32_t enable        :1;  //是否使能。
    volatile uint32_t warn_pri      :6;  //错误优先级
    volatile uint32_t beep_mask  :24; //蜂鸣器或者 led 显示错误的序列。
    volatile uint32_t delta_time   :16;  //当前与上次直接的时间间隔。
    volatile uint32_t set_timeout:16;  //设定超时时间
    void (*f_err_deal)(void);//暂时没用到。保留作为错误处理函数。
}ErrorFlagTypedef;
#pragma pack()
```

全局错误检测结构体。包含各个模块的 list，当前错误的 string，id 以及指针。

```
typedef struct{
    volatile ErrorFlagTypedef* err_now;
    volatile ErrorFlagTypedef err_list[ErrorListLength];
    char *str; //for oled display
    TOE_ID_e id;
}GlbRxErrTypeDef;
```

模块 Id 枚举：

```
typedef enum{
    NoTimeOutErr = 0,
```

ChassisGyroTOE,
ChassisMoto1TOE,
ChassisMoto2TOE,
ChassisMoto3TOE,
ChassisMoto4TOE,
CurrentSampleTOE,
DbusTOE,
JudgeTOE,
GimbalYawTOE,
GimbalPitTOE,
PokeMotoTOE,　　//拨弹电机。
ErrorListLength，}TOE_ID_e;

Task 介绍：（50ms 周期）

1、 初始化全局结构体 gRxErr 的列表，

例如 gRxErr.err_list[DbusTOE].err_exist = 0;

gRxErr.err_list[DbusTOE].warn_pri = 10;　　//max !

gRxErr.err_list[DbusTOE].beep_mask = Bi_Bi;

gRxErr.err_list[DbusTOE].set_timeout = 100; //ms

gRxErr.err_list[DbusTOE].enable = 1;

2、（到各个模块的 RX IT 中插入 err_detector_callback）

开始进入循环检测。

while(1){

1，遍历 err-list[ErrorListLength], 如果存在 delta time > set timeout。就说明存在错误了。然后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。

}

Kb task：（10ms 周期）

这部分非常简单。首先

介绍一个根据有限状态机设计的按键单击检测。例如鼠标左键按下再松开才算完整的一次按下，避免抖动引起的意图单击实际连击的情况。

u8　KeyStateMachine(u8* psta, u8 condition)。每 10ms 扫描一次，根据函数返回值得到按键单击与否。具体实现就是保存按键状态，根据当前按键输入决定状态转移与输出。

key_mouse_task：

1、 延时 1s。等待系统稳定

2 进入循环

While（1）

{

1. 根据遥控器结构体变量 rc.kb.code 得到一个 map 到 16 个按键的变量。
2. 根据 WSAD 键来积分，同时有限幅。不能直接给定一个 vx 或 vy 是因为手感问题，所以需要加上一个平缓的曲线，防止底盘行动时抖动。

//end of loop

}

代码中值得一谈的其他部分：
关于 uart 的空闲中断，通过 dma 接收数据的思路。
首先重写了一个
static int UART_Receive_DMA_No_IT(UART_HandleTypeDef* huart, u8* pData, u32 Size)
这个函数开启串口空闲中断（即接收完一串数据后硬件会发生 IDLE 中断）。有一点要注意的是，这里只开启 UART 的全局中断，对应的 IRQhandler 是 void USART1_IRQHandler(void)。不开启 UART 的 DMA channel 的 DMA IT。仅仅只是使用 DMA 来传输数据，不触发 DMA 接收完毕的 IT。void DMA1_Stream1_IRQHandler(void)这类函数 handler 也是不会进入的。这是为了减少 dma 频繁中断造成的系统任务切换，提高系统运行效率。
使用 idle it 需要在真正的 uart IRQhandler（）里面加入一个自定义的数据帧接收解析和重启 DMA 的函数，例如 LanggoUartFrameIRQHandler(&huart1);这个函数完成对接收缓冲区（一个数组而已）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。解析数据之后应该记得要清楚 idle flag，同时重启 DMA。将接收指针重新指向接收缓冲区的起始位置。这些操作调用 API uart_reset_idle_rx_callback(huartx);实现。
另外还有一点就是串口直接如果需要处理 float 类型的数据，就使用联合体或者直接内存拷贝 4 个字节（注意大小端）然后发送。接收的时候重新组合就行了。