**Vesc电调使用文档（使用CAN和APP）**

作者：杨键翌 2023/10/12

首先下载VESC TOOL：软件作者放在文件夹中。

由于我们需要用CAN对VESC电调进行通信和控制，所以不采用最新版本的上位机软件(作者已踩坑),所以使用旧版的VESC TOOL。

**上位机的配置：**

1. APP to USE: 选择No APP
2. VESC ID: id，在can总线仲裁机制中，id越小的响应优先，我们统一规定使用101，102，103…

③Timeout：控制信号超时时间，如果超过此时间未接收到控制信号，则VESC停止输出

④Timeout Break Current:控制信号超时(即vesc在规定时间内没有接收到can信号)刹车电流，为0时不刹车

⑤Can Status Message Mode：打开对应得CAN发送消息，vesc通过can时，主动发送数据到CAN总线中，对应下面结构体中的数据，选择12345

//VESC状态参数结构体

typedef uint32\_t systime\_t;

typedef struct {

    int id;

    systime\_t rx\_time;

    float rpm;

    float current;

    float duty;

} can\_status\_msg;

typedef struct {

    int id;

    systime\_t rx\_time;

    float amp\_hours;

    float amp\_hours\_charged;

} can\_status\_msg\_2;

typedef struct {

    int id;

    systime\_t rx\_time;

    float watt\_hours;

    float watt\_hours\_charged;

} can\_status\_msg\_3;

typedef struct {

    int id;

    systime\_t rx\_time;

    float temp\_fet;

    float temp\_motor;

    float current\_in;

    float pid\_pos\_now;

} can\_status\_msg\_4;

typedef struct {

    int id;

    systime\_t rx\_time;

    float v\_in;

    int32\_t tacho\_value;

} can\_status\_msg\_5;

⑥CAN State Rate: CAN状态消息得发送频率，选择默认500Hz

⑦CAN\_Baud Rate:配置VESC上CAN的Bound Rate，由于我们还经常使用大疆电机，所以我们直接配置为1MHz

⑧UAVCAN为一种航天通信协议，无需配置

**STM32F407的CAN滤波器配置，以及CAN的数据格式：**

在user\_can\_init中：

/\*\*

 \* @brief 配置CAN滤波器

\*/

void User\_Open\_Can2(void)

{

    CAN\_FilterTypeDef Filter;

    Filter.FilterBank = 15; //过滤器组别15

    Filter.FilterMode = CAN\_FILTERMODE\_IDMASK;

    Filter.FilterScale = CAN\_FILTERSCALE\_32BIT;

    Filter.FilterIdHigh         = (((uint32\_t)CAN\_RxExtId<<3)&0xffff0000)>>16;              /\* 要过滤的ID高位 \*///0x0000

    Filter.FilterIdLow          = (((uint32\_t)CAN\_RxExtId<<3)|CAN\_ID\_EXT|CAN\_RTR\_DATA)&0xffff0000; /\* 要过滤的ID低位 \*///0x0000

    Filter.FilterMaskIdHigh = 0x0000;   //拓展帧标识符

    Filter.FilterMaskIdLow = 0x0000;

    Filter.FilterFIFOAssignment = CAN\_RX\_FIFO0;

    Filter.FilterActivation = ENABLE;   //使能过滤器

    Filter.SlaveStartFilterBank = 14;

    if (HAL\_CAN\_ConfigFilter(&hcan2, &Filter) != HAL\_OK)

    {

            /\* Filter configuration Error \*/

            Error\_Handler();

    }

    if (HAL\_CAN\_Start(&hcan2) != HAL\_OK)

    {

        /\* Start Error \*/

        Error\_Handler();

    }

    if (HAL\_CAN\_ActivateNotification(&hcan2, CAN\_IT\_RX\_FIFO0\_MSG\_PENDING) != HAL\_OK)

    {

        /\* Start Error \*/

        Error\_Handler();

    }

    if (HAL\_CAN\_ActivateNotification(&hcan2, CAN\_IT\_TX\_MAILBOX\_EMPTY) != HAL\_OK)

    {

        /\* Start Error \*/

        Error\_Handler();

    }

}

**使用拓展帧发送数据：**控制指令在下列的枚举中

//can命令枚举

typedef enum {

    CAN\_PACKET\_SET\_DUTY                     = 0,

    CAN\_PACKET\_SET\_CURRENT                  = 1,

    CAN\_PACKET\_SET\_CURRENT\_BRAKE            = 2,

    CAN\_PACKET\_SET\_RPM                      = 3,

    CAN\_PACKET\_SET\_POS                      = 4,

    CAN\_PACKET\_FILL\_RX\_BUFFER               = 5,

    CAN\_PACKET\_FILL\_RX\_BUFFER\_LONG          = 6,

    CAN\_PACKET\_PROCESS\_RX\_BUFFER            = 7,

    CAN\_PACKET\_PROCESS\_SHORT\_BUFFER         = 8,

    CAN\_PACKET\_STATUS                       = 9,

    CAN\_PACKET\_SET\_CURRENT\_REL              = 10,

    CAN\_PACKET\_SET\_CURRENT\_BRAKE\_REL        = 11,

    CAN\_PACKET\_SET\_CURRENT\_HANDBRAKE        = 12,

    CAN\_PACKET\_SET\_CURRENT\_HANDBRAKE\_REL    = 13,

    CAN\_PACKET\_STATUS\_2                     = 14,

    CAN\_PACKET\_STATUS\_3                     = 15,

    CAN\_PACKET\_STATUS\_4                     = 16,

    CAN\_PACKET\_PING                         = 17,

    CAN\_PACKET\_PONG                         = 18,

    CAN\_PACKET\_DETECT\_APPLY\_ALL\_FOC         = 19,

    CAN\_PACKET\_DETECT\_APPLY\_ALL\_FOC\_RES     = 20,

    CAN\_PACKET\_CONF\_CURRENT\_LIMITS          = 21,

    CAN\_PACKET\_CONF\_STORE\_CURRENT\_LIMITS    = 22,

    CAN\_PACKET\_CONF\_CURRENT\_LIMITS\_IN       = 23,

    CAN\_PACKET\_CONF\_STORE\_CURRENT\_LIMITS\_IN = 24,

    CAN\_PACKET\_CONF\_FOC\_ERPMS               = 25,

    CAN\_PACKET\_CONF\_STORE\_FOC\_ERPMS         = 26,

    CAN\_PACKET\_STATUS\_5                     = 27,

CAN\_PACKET\_ID;

**CAN数据发送函数：**

void comm\_can\_transmit\_eid(uint32\_t controller\_id, uint8\_t \*data, uint8\_t len)

{

    CAN\_TxHeaderTypeDef         TxHeader;   //can数据发送句柄

    uint32\_t txmailbox;

    if (len > 8)

    {

        len = 8;

    }

    TxHeader.DLC = (uint8\_t)8 ;     //数据长度

    TxHeader.RTR = CAN\_RTR\_DATA ;

    TxHeader.IDE = CAN\_ID\_EXT ;     //使用拓展帧

    TxHeader.ExtId = controller\_id; //VESC电调id

    while (HAL\_CAN\_GetTxMailboxesFreeLevel(&hcan2) == 0)    //等待发送邮箱空

        HAL\_Delay(1);                                       //防止炸邮箱

    if (HAL\_CAN\_AddTxMessage(&hcan2, &TxHeader, data, &txmailbox)!= HAL\_OK)

    {

        Error\_Handler();

    }

}

例子：

通过can总线使用转速模式驱动电机

参数：

void comm\_can\_set\_rpm(uint8\_t controller\_id, float rpm)

{

    int32\_t send\_index = 0;

    uint8\_t buffer[4];

    buffer\_append\_int32(buffer, (int32\_t)rpm, &send\_index);

    comm\_can\_transmit\_eid(controller\_id |

                            ((uint32\_t)CAN\_PACKET\_SET\_RPM << 8),

                            buffer, send\_index);

}

其中的controller\_id是can驱动器的can id，即上位机设置的，CAN\_PACKET\_SET\_RPM是枚举类型中的命令

**从CAN总线中接收电机数据**

void can\_get\_data(CAN\_RxHeaderTypeDef \*RxHeader, uint8\_t \*RxData)

{

    can\_status\_msg   status\_msg;    //储存VESC发来的can数据

    can\_status\_msg\_2 status\_msg\_2;

    can\_status\_msg\_3 status\_msg\_3;

    can\_status\_msg\_4 status\_msg\_4;

    can\_status\_msg\_5 status\_msg\_5;

    uint16\_t cmd;

    uint8\_t  id = (RxHeader->ExtId & 0xFF); //截取ID

    cmd = (RxHeader->ExtId >> 8);   //获取对应的帧头

    //对can数据包进行解码

    switch (cmd)

    {

        case CAN\_PACKET\_STATUS:

        {

            int index = 0;  //记录数组操作到哪个位置

            status\_msg.id = id;

            status\_msg.rx\_time = 0; //没啥用

            status\_msg.rpm = buffer\_get\_uint32(RxData,&index);

            status\_msg.current = (float)(buffer\_get\_int16(RxData,&index)/10.0f);

            status\_msg.duty = (float)(buffer\_get\_int16(RxData,&index)/1000.0f);

            // test\_id = value->controller\_id;

            break;

        }

        case CAN\_PACKET\_STATUS\_2:

        {

            int index = 0;

            status\_msg\_2.id = id;

            status\_msg\_2.rx\_time = 0;

            status\_msg\_2.amp\_hours = (float)buffer\_get\_int32(RxData, &index) / 1e4;

            status\_msg\_2.amp\_hours\_charged =  (float)buffer\_get\_int32(RxData, &index) / 1e4;

            break;

        }

        case CAN\_PACKET\_STATUS\_3:

        {

            int index = 0;

            status\_msg\_3.id = id;

            status\_msg\_3.rx\_time = 0;

            status\_msg\_3.watt\_hours = (float)buffer\_get\_int32(RxData, &index) / 1e4;

            status\_msg\_3.watt\_hours\_charged = (float)buffer\_get\_int32(RxData, &index) / 1e4;

            break;

        }

        case CAN\_PACKET\_STATUS\_4:

        {

            int index = 0;

            status\_msg\_4.id = id;

            status\_msg\_4.rx\_time = 0;

            status\_msg\_4.temp\_fet = (float)buffer\_get\_int16(RxData,&index) / 10.0f;

            status\_msg\_4.temp\_motor = (float)buffer\_get\_int16(RxData,&index) / 10.0f;

            status\_msg\_4.current\_in = (float)buffer\_get\_int16(RxData,&index) / 10.0f;

            status\_msg\_4.pid\_pos\_now = (float)buffer\_get\_int16(RxData,&index) / 50.0f;

            break;

        }

        case CAN\_PACKET\_STATUS\_5:

        {

            int index = 0;

            status\_msg\_5.id = id;

            status\_msg\_5.rx\_time = 0;

            status\_msg\_5.tacho\_value = buffer\_get\_int32(RxData, &index);

            status\_msg\_5.v\_in =  (float)buffer\_get\_int16(RxData, &index) / 1e1;

            break;

        }

        default: break;

    }

}

直接将该函数放入CAN的接收中断回调函数中即可，其中的STATUS命令即开头所讲的Status Message Mode