WTR_IPM_ESC Can_Command

> **需要准备的材料**
>
> - 能够收发 CAN 指令的 PC/MCU;

系统状态

电调包含以下状态:

- **停止状态(STOP);**
- **调试运行状态(DEBUG_RUN);**
- **运行状态(RUN);**
- **校准状态(TEST);**
- **错误状态(FAULT);**

使用 STM32 MCU 进行电调控制

基本配置

库文件在 `WTR_IPM_LIB` 文件夹中。

> ***根据 MCU 是 FDCAN/CAN 接口选择使用的库文件***:
>
> - FDCAN: `FDCAN_Library`;
> - CAN: `CAN_Library`

- MCU 配置
  1. **配置 CAN 通信波特率为 500k bit/s.**
  2. **打开 RX0 中断。**

FDCAN_Library/CAN_Library 中的库接口

宏定义

```
#define IPM_ESC_CAN_Handler hfdcan1
```

该宏定义应当连接 CubeMX 中已经配置好的 CAN/FDCAN。定义为 `hfdcanx` / `hcanx`。

接收数据

电调将会以 10 Hz 的频率回传速度和位置数据。

> ***该位置为电机校准后的累积位置，如果上电后未运行过校准指令，该位置的零点将会是电机上电一刻的位置。***
>
> ***详情见*** *`Uart_Command.md`*。

该数据保存在全局变量 `ipm_esc_msg` 中:

```
typedef struct
{
    float position; // 位置(rad)(绝对位置,自动包含圈数)
    float speed;    // 速度(rad/s)
    uint8_t id;     // 电调 ID
} IPM_ESC_RX_MSG;
```

为使得 MCU 获取到传回的数据，需要明确定义中断回调函数:

```c
// CAN
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    uint8_t RxData[8];
    CAN_RxHeaderTypeDef RxHeader;
    if (hcan->Instance == IPM_ESC_CAN_Handler.Instance) {
        HAL_CAN_GetRxMessage(&IPM_ESC_CAN_Handler, CAN_RX_FIFO0, &RxHeader, RxData);
        if (ipm_esc_it_flag == 0) {
            ipm_esc_it_flag = 1;
            IPM_ESC_Decode(&ipm_esc_msg, RxHeader.StdId, RxData);
            ipm_esc_it_flag = 0;
        }
    }
}

// FDCAN
void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)
{
    uint8_t RxData[8];
    FDCAN_RxHeaderTypeDef sRxHeader;
    if (hfdcan->Instance == IPM_ESC_CAN_Handler.Instance && (RxFifo0ITs ==
FDCAN_IT_RX_FIFO0_NEW_MESSAGE)) {
        HAL_FDCAN_GetRxMessage(&IPM_ESC_CAN_Handler, FDCAN_RX_FIFO0, &sRxHeader, RxData);
        if (ipm_esc_it_flag == 0) {
            ipm_esc_it_flag = 1;
            IPM_ESC_Decode(&ipm_esc_msg, sRxHeader.Identifier, RxData);
            ipm_esc_it_flag = 0;
        }
    }
}
```

使用以下函数可以读取特定 ID 电调传回的数据(推荐使用)，这些函数需要循环运行以连续读取：

```c
/**
 * @brief   获取电机位置
 * @param   id          电调 ID
 * @return  电机位置(rad)(绝对位置,自动包含圈数)
 */
float IPM_ESC_Get_Position(uint8_t id);

/**
 * @brief   获取电机速度
 * @param   id          电调 ID
 * @return  电机速度(rad/s)
 */
float IPM_ESC_Get_Speed(uint8_t id);
```

发送指令

> ***CAN发送指令在错误状态(FAULT)和校准状态(TEST)下无效。***
>
> ***UART发出的校准指令会使得电调进入校准状态(TEST)***，详情见 `Uart_Command.md`。

1. 速度/位置设置

```c
/**
 * @brief    设置电机速度
 * @param    speed    目标速度(rad/s)
 * @param    id       电调 ID
 */
void IPM_ESC_Set_Speed(float speed, uint8_t id);

/**
 * @brief    设置电机位置
 * @param    position     目标位置(rad)
 * @param    id           电调 ID
 */
void IPM_ESC_Set_Position(float position, uint8_t id);
```

> ***运行前强烈建议使用 UART 的 `calibration` 指令进行一次校准并用 UART 的 `save` 指令保存校准参数。***
>
> ***该函数会使电调进入运行状态(RUN),此时 UART 任何指令失效。***
>
> ***该函数需要保证 5 Hz 以上的发送频率，电调侧在 2000ms 内未收取到数据将会自动进入停止(STOP)状态。***

2. 停止设置

```c
/**
 * @brief    停止电机
 * @param    id           电调 ID
 */
void IPM_ESC_Stop(uint8_t id);
```

> ***该函数如果连续运行等同于屏蔽 UART 指令，因此单次发送即可。***
>
> ***该函数会使电调进入停止状态(STOP)，失去位置和速度控制功能。***

使用 PC 进行电调控制

需要准备 USB-CAN 并且准备对应上位机。

接收数据如下:

ID	byte[0]	byte[1]	byte[2]	byte[3]	byte[4]	byte[5]	byte[6]	byte[7]
电调ID	位置字节0	位置字节1	位置字节2	位置字节3	速度字节0	速度字节1	速度字节2	速度字节3

> ***位置和速度为4字节数据，采用大端的float-array转换，可以使用float-array工具进行解码读取:***
>
> 解码工具链接

发送数据如下(x表示任意):

1. 停止指令:

ID	byte[0]	byte[1]	byte[2]	byte[3]	byte[4]	byte[5]	byte[6]	byte[7]
电调ID	0x01	x	x	x	x	x	x	x

2. 位置控制指令:

ID	byte[0]	byte[1]	byte[2]	byte[3]	byte[4]	byte[5]	byte[6]	byte[7]
电调ID	0x02	位置字节0	位置字节1	位置字节2	位置字节3	x	x	x

3. 速度控制指令:

ID	byte[0]	byte[1]	byte[2]	byte[3]	byte[4]	byte[5]	byte[6]	byte[7]
电调ID	0x03	速度字节0	速度字节1	速度字节2	速度字节3	x	x	x