AK 系列模组驱动使用说明

V1.0.13

目录

注意事项

1. 确保电路无短路，接口按照要求正确连接。

2. 驱动板输出时，会出现发热情况，请小心使用，以免烫伤。

3. 使用前请检查各零部件是否完好。如有部件缺失、老化，请停止使用并及时联系技术支持。

4. 多种可选控制方式在驱动板运行时不可切换，且不同控制方式之间通信协议不同。如需切换，请重新启动电源，然后再进行更改。使用错误的协议控制可能会使驱动板烧毁！

5. 请严格按照本文规定的工作电压、电流、温度等参数使用，否则会对产品造成永久性的损坏！

产品特色

AK 系列电机驱动板采用同级别中高性能的驱动芯片，使用 Field Oriented Control（简称 FOC）算法，搭配先进自抗扰控制技术对速度和角度进行控制。可配合 CubeMarsTool 调参软件进行参数设置并升级固件。

免责声明

感谢您购买 AK 系列模组电机。在使用之前，请仔细阅读本声明，一旦使用，即被视为对本声明全部内容的认可和接受。请严格遵守产品说明书和相关的法律法规、政策、准则安装和使用该产品。在使用产品过程中，用户承诺对自己的行为及因此而产生的所有后果负责。因用户不当使用、安装、改装造成的任何损失，CubeMars 将不承担法律责任。
CubeMars 是江西新拓实业有限公司及其关联公司的商标。本文出现的产品名称、品牌等，均为其所属公司的商标。本产品及手册为江西新拓实业有限公司版权所有。未经许可，不得以任何形式复制翻印。关于免责声明的最终解释权，归江西新拓实业有限公司所有。

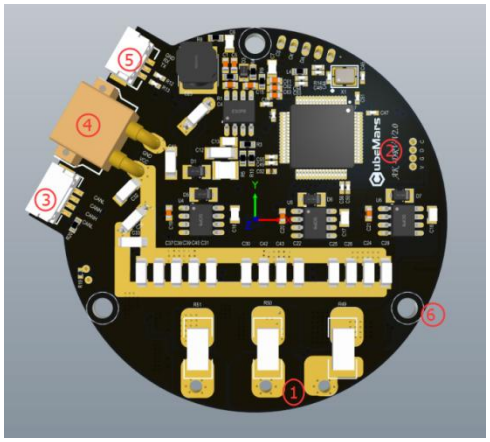
版本变更记录

日期	版本	变更内容
2021.09.01	Ver. 1.0.0	创建版本
2021.10.08	Ver.1.0.1	5.1 与 5.2 代码变更
2021.10.29	Ver.1.0.2	5.1,5.2 和 5.3 数据定义更新
2021.11.15	Ver.1.0.3	CAN 报文接收定义
2021.11.24	Ver.1.0.4	5.2 中 UART 协议更新
2021.11.30	Ver.1.0.5	5.3 中信息增加
2022.01.20	Ver.1.0.6	5.3 中 AK60-6 电机速度更改

2023.07.19	VER.1.0.10	1.红灯说明解释 2.新增 80-8 60KV MIT 参数
2023.08.29	VER.1.0.12	1.修正位置速度环例程代码 2.修改伺服模式字节顺序注明
2023.12.11	VER.1.0.13	报错完善，原点模式和发送代码修改

1. 驱动器产品信息

1.1 驱动器外观简介&产品规格



①三相动力线连接端

②硬件版本号

③CAN 通讯连接端口

④DC 电源接口

⑤串口通讯连接端口

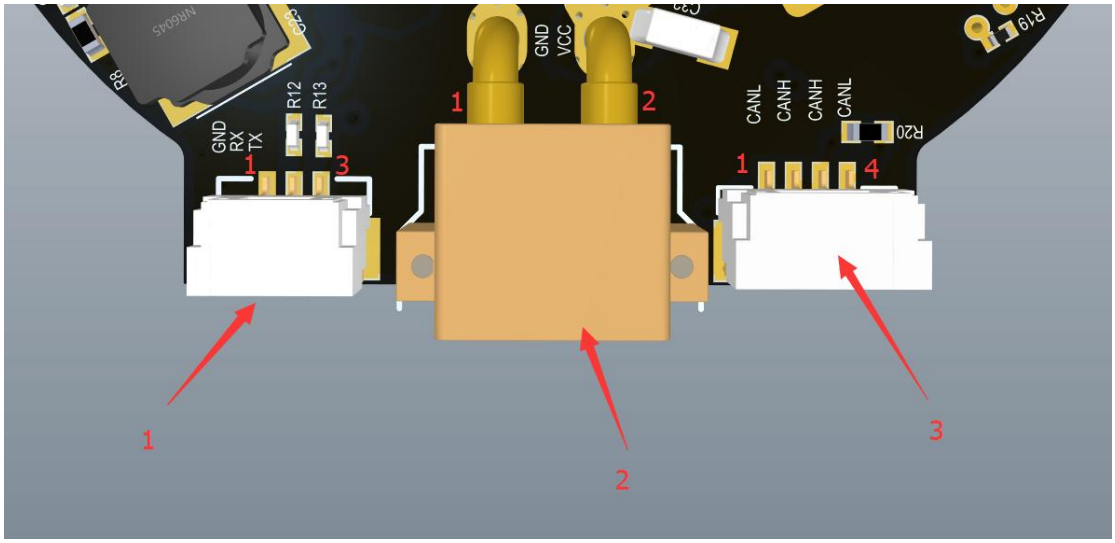
⑥安装孔

产品规格	
额定工作电压	48V
最大允许电压	52V
额定工作电流	20A
最大允许电流	60A
待机功耗	≤50mA
CAN 总线比特率	1Mbps（不建议更改）
尺寸	62mm×58mm

工作环境温度	-20℃至 65℃
控制板最大允许温度	100℃
编码器精度	14bit（单圈绝对值）

1.2 驱动器接口及定义

1.2.1 驱动器接口图



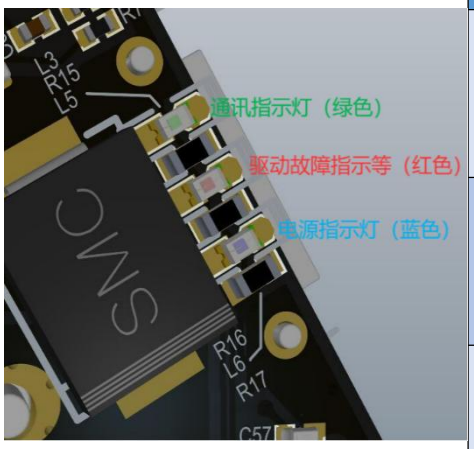
1.2.2 驱动器接口推荐品牌及型号

序号	板载接口型号	品牌	线端接口型号	品牌
1	A1257WR-S-3P	CJT(长江连接器)	A1257H-3P	CJT(长江连接器)
2	XT30PW-M	AMASS(艾迈斯)	XT30UPB-F	AMASS(艾迈斯)
3	A1257WR-S-4P	CJT(长江连接器)	A1257H-4P	CJT(长江连接器)

1.2.3 驱动器接口引脚定义

序号	接口功能	引脚	说明
1	**串口通讯**	1	串口信号地（GND）
		2	串口信号输出（TX）
		3	串口信号输入（RX）
2	**电源输入**	1	电源负极（-）
		2	电源正极（+）
3	**CAN 通讯**	1	CAN 通讯低侧（CAN_L）
		2	CAN 通讯高侧（CAN_H）
		3	CAN 通讯高侧（CAN_H）
		4	CAN 通讯低侧（CAN_L）

1.3 驱动器指示灯定义



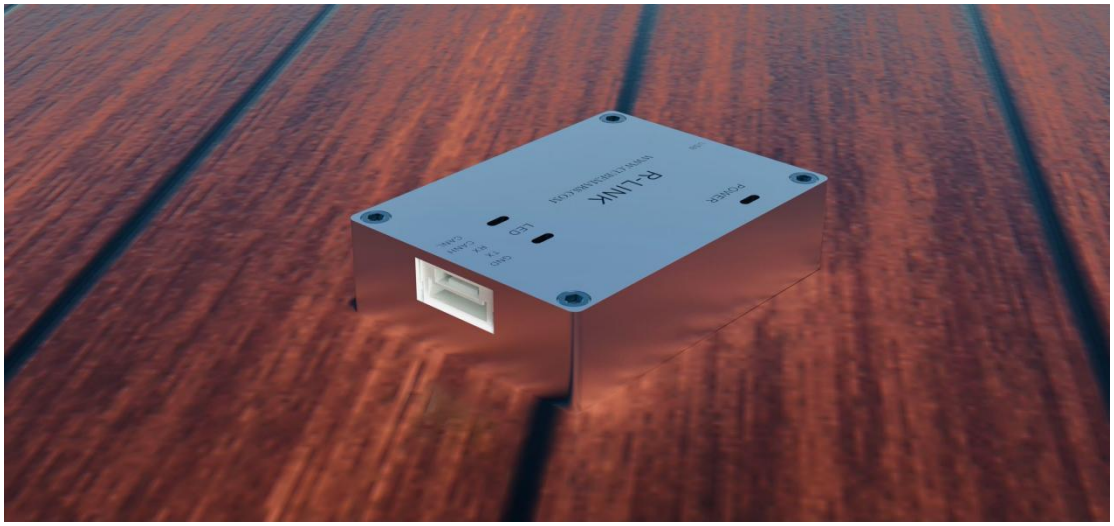
指示灯定义	
1. 电源指示灯（亮时蓝灯）	电源指示灯，用于指示驱动板电源情况，正常情况下电源插入时都会亮起蓝色，如插入电源时蓝灯不亮，请立刻移除电源，切不可再上电。
2. 通讯指示灯（亮时绿色）	通讯指示灯，用于指示驱动板通讯情况，正常情况下驱动板正常通讯时才会亮起绿色，如绿灯不亮，请先检查 CAN 通讯接线是否正常。
3. 驱动故障指示灯（亮时红色）	驱动故障指示灯，用于指示驱动板故障情况，正常情况下只有当驱动板出现故障才会亮起红色，通常情况时常灭的。当驱动故障指示灯亮起时，说明驱动板已经产生了一定损伤，应当关闭电源，切勿操作。（给驱动板通电时红灯微亮正常）

1.4 主要配件及规格

序号	项目	规格		数量	备注
1	串口通讯线	线材	24AWG-300MM-铁氟龙镀银线-黑黄绿	各 1PCS	±2MM
		插头	A1257H-3P	1PCS	
			A2541H-3P	1PCS	
2	电源线	线材	16AWG-200MM-硅胶线-红黑	各 1PCS	±2MM
		插头	XT30UPB-M	1PCS	
			XT30UPB-F	1PCS	
3	CAN 通讯线	线材	24AWG-300MM-铁氟龙镀银线-白蓝	各 1PCS	±2MM
		插头	A1257H-4P	2PCS	
			A2541H-2P	1PCS	
4	热敏电阻	MF51B103F3950-10K-3950		2PCS	
5	电解电容	120Uf-63V-10x12MM		2PCS	AK10-9 V2.0 以上标配
6	功率 MOS	BSC026N08NS5-80V-2.6mΩ TPH2R608NH-75V-2.6mΩ		12PCS	随机

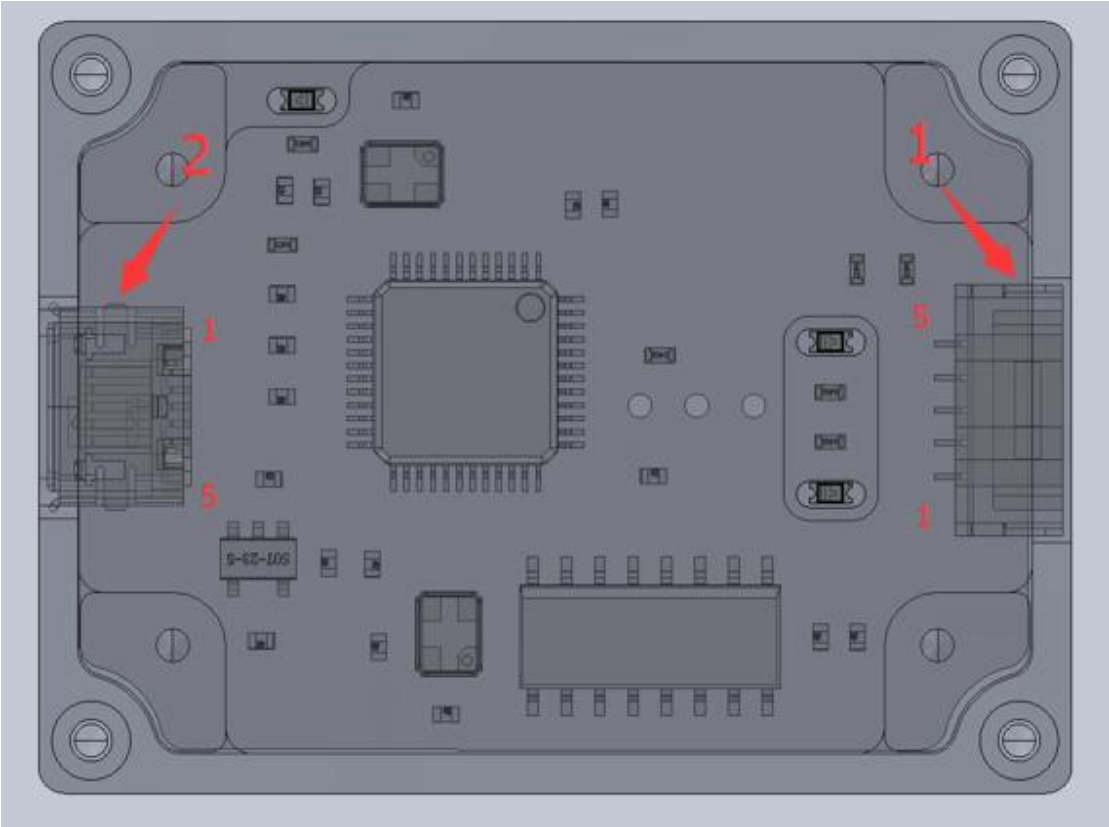
2. R-link 产品信息

2.1 R-link 外观简介&产品规格



产品规格	
额定工作电压	5V
待机功耗	≤30mA
外形尺寸	39.2x29.2x10MM
工作环境温度	-20℃至 65℃
控制板最大允许温度	85℃

2.2 R-link 接口及定义

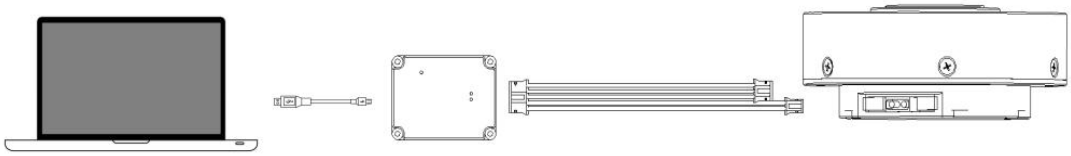


序号	接口功能	引脚	说明
1	通讯接口	1	CAN 通讯低侧（CAN_L）
		2	CAN 通讯高侧（CAN_H）
		3	串口信号输入（RX）
		4	串口信号输出（TX）
		5	串口信号地（GND）
2	USB 接口	1	VBUS
		2	D-
		3	D+
		4	ID
		5	GND

2.3 R-link 指示灯定义

序号	颜色定义	说明
1	绿色	电源指示灯，用于指示 R-link 电源情况，正常情况下电源插入时都会亮起绿色，如插入电源时绿灯不亮，请立刻移除电源，切不可再上电。
2	蓝色	串口通讯输出（TX），常灭，当 R-link 串口有数据输出时，闪烁。
3	红色	串口通讯输出（RX），常灭，当 R-link 串口有数据输入时，闪烁。

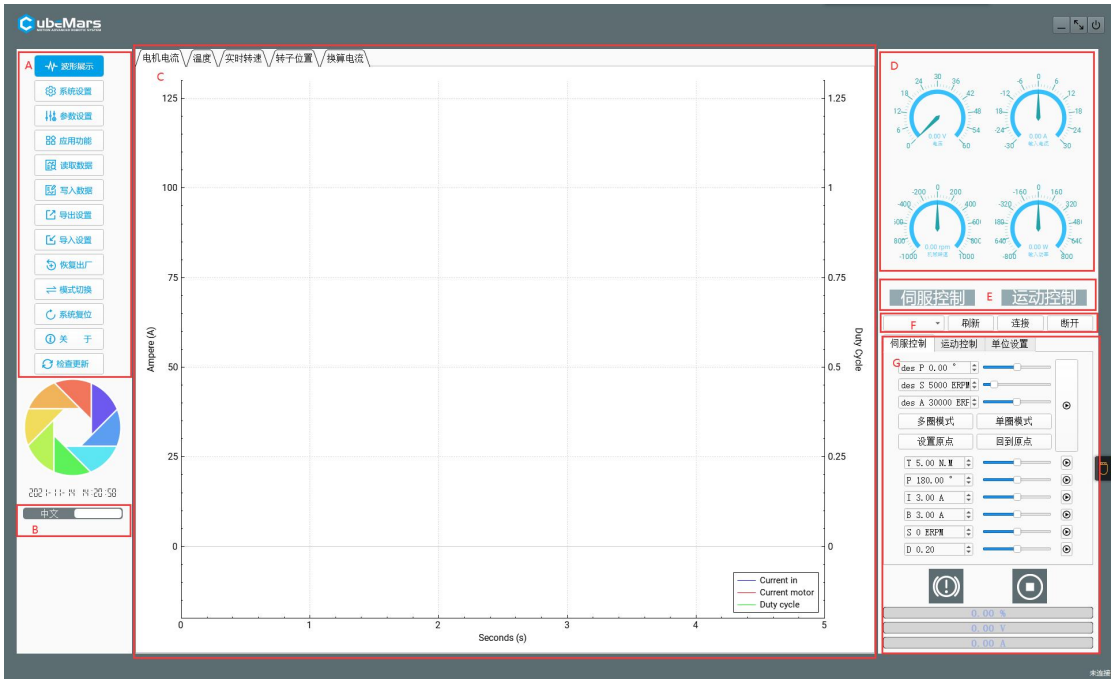
3. 驱动器与 R-link 连接及注意事项



R-Link 上的 USB 线 ----> PC 端
5Pin 口 ----> RLink5Pin 口端
4Pin 端子（CAN 口） ----> 电机上的 4Pin 口（CAN）
3Pin 端子（UART 口） ----> 电机上的 3Pin 口（UART）

4.上位机使用说明

4.1 上位机界面及说明

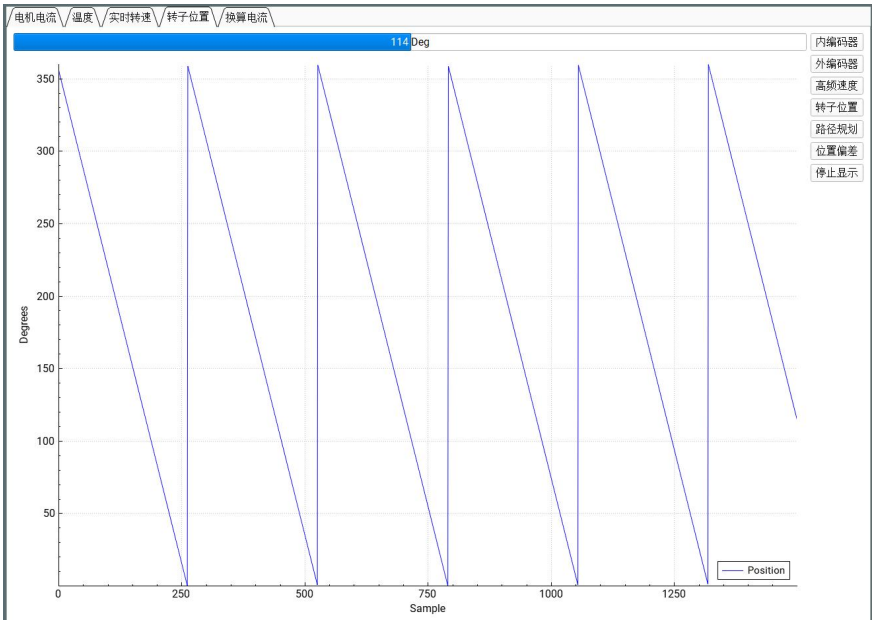


A. 主菜单栏
B. 中英文切换
C. 主页面
D. 实施数据显示
E. 当前模式
F. 串口选择
G. 控制参数

4.1.1 主菜单栏

4.1.1.1 波形展示

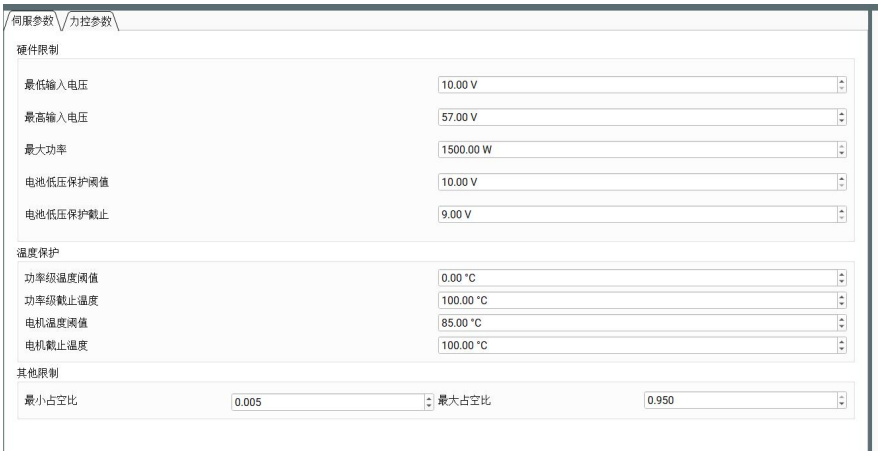
该页面支持观看实时数据反馈，并绘制图像。数据包括：电机电流、温度、实时转速、内编码器位置、外编码器位置、高频速度、转子位置、路径规划、位置偏差、DQ 电流等。



4.1.1.2 系统设置

该页面主要为更改电压、电流、功率、温度、占空比等驱动板硬件限制，主要起保护驱动板及电机功能。

⚠：请务必严格按照规定电压、电流、功率、温度使用。因违规操作本产品导致对人体造成伤害，或对驱动板及电机造成不可逆的损伤，我司将不承担任何法律责任。



4.1.1.3 参数设置

该页面主要为调整驱动板参数，包括但不限于电流环 Kp -Ki、编码器偏执、电流最大最小值、转速最大最小值、速度环 Kp-Ki-KD、减速比等参数以及编码器的校准和电机参数整定。

⚠：请务必严格按照规定电压、电流、功率、温度使用。因违规操作本产品导致对人体造成伤害，或对驱动板及电机造成不可逆的损伤，我司将不承担任何法律责任。

4.1.1.4 应用功能

该页面主要为 CAN ID 设置、CAN 通讯速率以及 CAN 通讯突然中断的设置。

4.1.1.5 读取数据

将电机当前参数保存到上位机，**⚠：每当改写电机参数时务必先确保先点击此按钮，否者电机其他参数将会出错，如果发生此类情况，请到官网下载对应电机的默认 APP 参数，再通过”导入设置”将该电机的默认参数写入电机！**

4.1.1.6 参数保存

将上位机参数保存至电机。

4.1.1.7 导出设置

将上位机参数保存为后缀名为“.McParams”和“.AppParams”两个文件，保存至电脑。

其中“.McParams”文件为：

“.AppParams”文件为：

4.1.1.8 导入设置

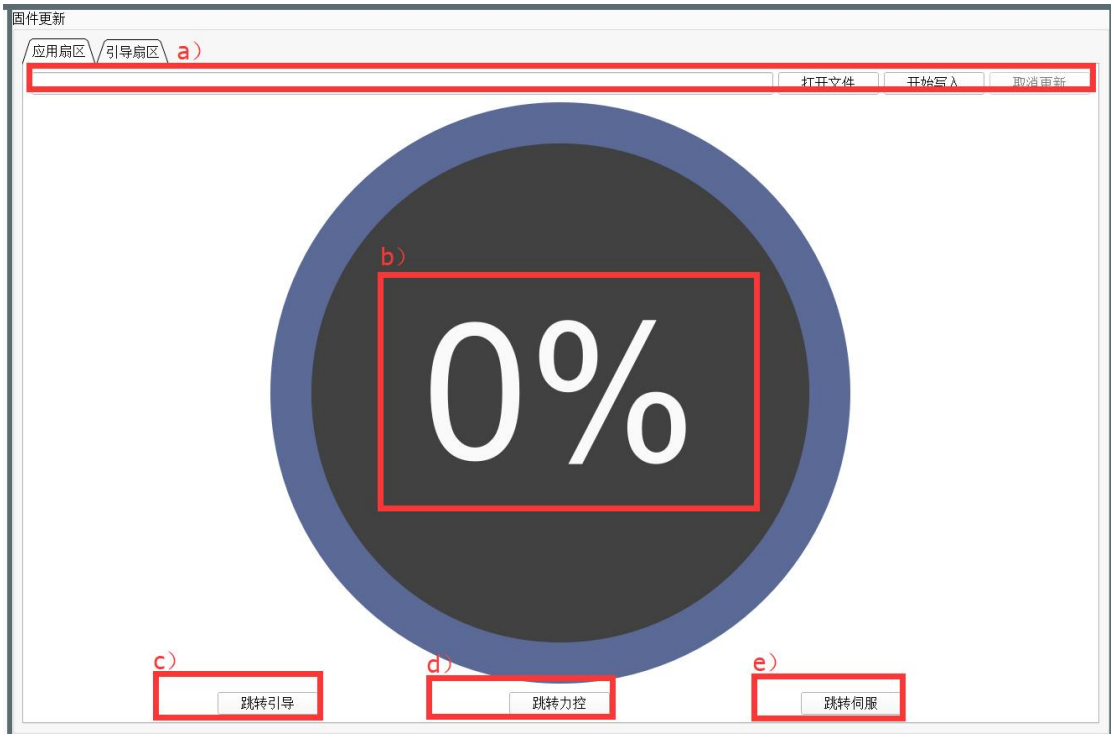
将电脑上后缀名为“.McParams”和“.AppParams”两个文件的参数上传至上位机。

4.1.1.9 恢复出厂

暂未开放此功能。

4.1.1.10 模式切换

该页面主要为切换驱动板的控制方式，其中有“引导模式”、“伺服模式”、“运控模式”。
以及更新驱动板固件。



a）.导入固件区：能够导入电脑中后缀名为“.bin”文件
b）.固件更新进度条
c）.进入引导模式
d）.进入运控模式
e）.进入伺服模式

4.1.1.11 系统复位

将电机停止并重启。

4.1.1.12 关于

当前上位机的版本号及官方主页。

4.2 驱动板校准

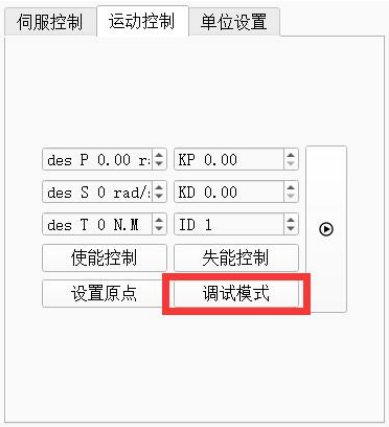
当你重新安装了驱动板在电机上，或者更换过电机三相线的线序，亦或者更新了固件之后，都必须进行校准。校准之后，便可以正常使用电机。

4. 2. 1 伺服模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，进入系统设置页面，依次单击“电气参数”、“辨识参数”、“更新参数”、“开始校准”、“更新参数”。

4. 2. 2 运控模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于运控模式时，在与上位机成功连接后，在“运动控制”界面单击“调试模式”，随后在输入栏输入“calibrate”，等待时间 30 秒左右，于此同时输出栏会实时滚动编码器的位置值，直到输出栏打印“Encoder Electrical Offset (rad)”时，电机会自动重新启动，然后串口将打印驱动信息。校准时，电压在 48V 下电流大约 1A，校准结束，电流恢复至 0.02A 左右。



4.3 控制演示

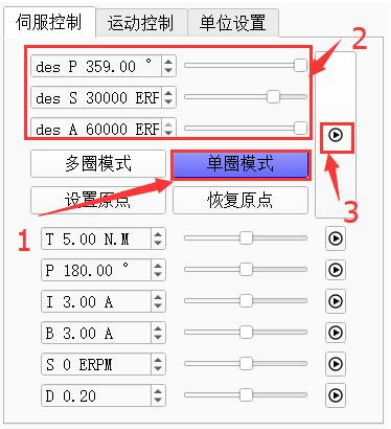
4. 3. 1 伺服模式

4.3.1.1 多圈位置速度模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，在“伺服控制”界面单击“多圈模式”，输入期望位置（此时位置为±100 圈，即-36000°-36000°）、期望速度和加速度后，电机将会按照期望速度运动直到期望位置。

4.3.1.2 单圈位置速度模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，在“伺服控制”界面单击“单圈模式”，输入期望位置（此时位置只有一圈，即 0° -359°）、期望速度和加速度后，电机将会按照期望速度运动直到期望位置。



4.3.1.3 位置模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，在“伺服控制”界面输入期望位置，电机将以最大转速到达期望位置。

4.3.1.4 速度模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，在“伺服控制”界面输入期望转速（±50000ERPM），电机将以期望转速运动。

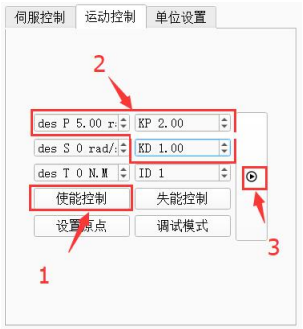
4.3.1.5 占空比模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于伺服模式时，在与上位机成功连接后，在“伺服控制”界面输入期望占空比（默认 0.005-0.95），电机将以期望占空比运动。

4.3.2 运控模式

4.3.2.1 位置模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于运控模式时，在与上位机成功连接后，在“运动控制”界面输入对应的“CAN ID”随后单击“使能控制”，即可进入电机模式，输入期望位置与 KP、KD 后，电机将会进行位置运动（默认速度 12000erpm 加速度 40000erpm）。

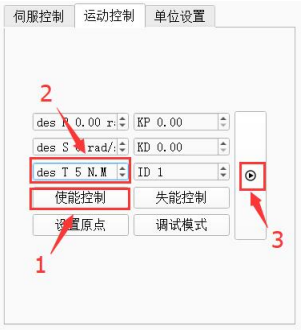


4.3.2.2 速度模式

确认电机输入电源稳定，R-LINK 连接正常，并且电机处于运控模式时，在与上位机成功连接后，在“运动控制”界面输入对应的“CAN ID”随后单击“使能控制”，即可进入电机模式，输入期望速度和 KD 后，电机将会进行速度运动。

4.3.2.3 扭矩模式

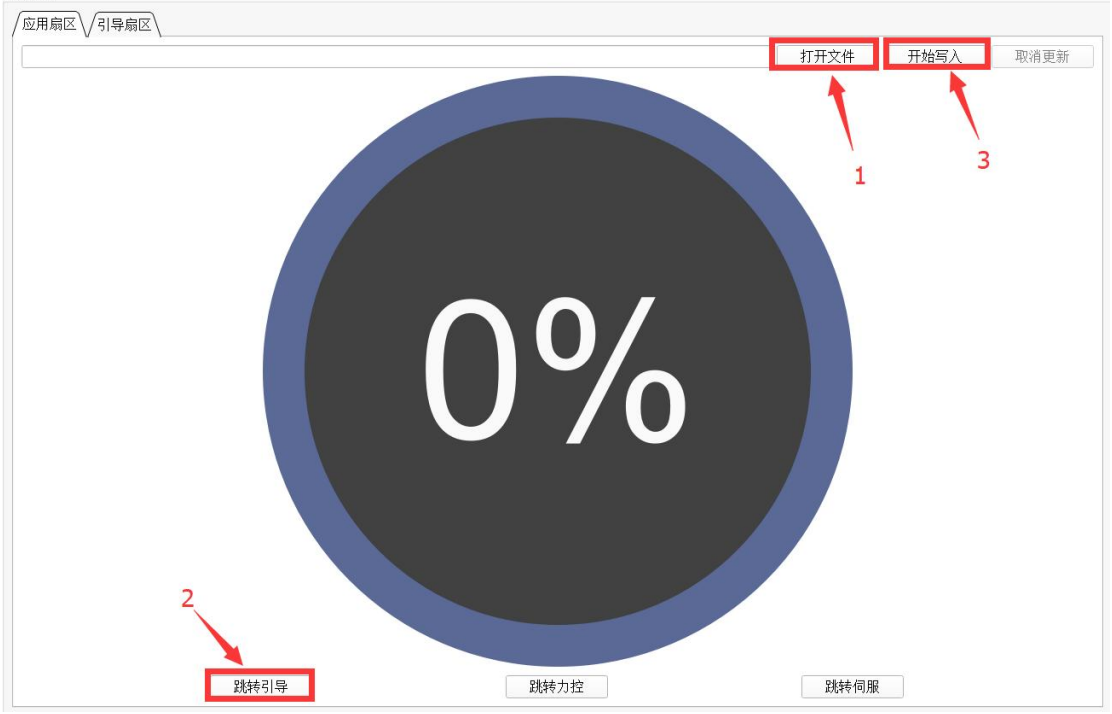
确认电机输入电源稳定，R-LINK 连接正常，并且电机处于运控模式时，在与上位机成功连接后，在“运动控制”界面输入对应的“CAN ID”随后单击“使能控制”，即可进入电机模式，输入期望扭矩后，电机将会进行扭矩运动。



4.4 固件更新

1.单击“打开文件”，选择固件，固件后缀名称一般为“.BIN”。
2.单击“跳转引导”
3.单击“开始写入”，等待进度条完成至 100%，重新启动电源，即固件更新完毕

固件更新



5. 驱动板通讯协议及说明

5.1 伺服模式控制模式及说明

伺服模式有 6 种控制模式

占空比模式：给定电机指定占空比电压，类似方波驱动形式

电流环模式：给定电机指定的 Iq 电流，由于电机输出扭矩=iq*KT，所以可以当作扭矩环使用

电流刹车模式：给定电机指定的刹车电流，让电机固定在当前位置（使用时注意电机温度）

速度模式：给定电机指定的运行速度

位置模式：给定电机指定的位置，电机将运行到该指定的位置，（默认速度 12000erpm 加速度 40000erpm)

位置速度环模式：给定电机指定的位置、速度、加速度。电机将以给定的加速度 和最大速度运行到指定的位置。

伺服电机协议为 can 协议，采用扩展帧格式如下示

Can ID bits	[28]-[8]	[7]-[0]
Field name (功能定义)	Control mode （控制模式）	Source node ID （从机 ID 号）

Control mode 有{0,1,2,3,4,5,6} 7 个特征值分别对应 7 种控制模式

占空比模式：0

电流环模式：1

电流刹车模式：2

转速模式：3

位置模式：4

设置原点模式：5

位置速度环模式：6

以下提供各种模式控制电机实例

下面为各种实例调用库 函数与宏定义

```
typedef enum {
    CAN_PACKET_SET_DUTY = 0,            //占空比模式
    CAN_PACKET_SET_CURRENT,             //电流环模式
    CAN_PACKET_SET_CURRENT_BRAKE,       // 电流刹车模式
    CAN_PACKET_SET_RPM,                 // 转速模式
    CAN_PACKET_SET_POS,                 // 位置模式
    CAN_PACKET_SET_ORIGIN_HERE,         //设置原点模式
    CAN_PACKET_SET_POS_SPD,             //位置速度环模式
} CAN_PACKET_ID;

void comm_can_transmit_eid(uint32_t id, const uint8_t *data, uint8_t len) {
    uint8_t i=0;
    if (len > 8) {
        len = 8;
    }
```

```
    CanTxMsg TxMessage;
    TxMessage.StdId = 0;
    TxMessage.IDE = CAN_ID_EXT;
    TxMessage.ExtId = id;
    TxMessage.RTR = CAN_RTR_DATA;
    TxMessage.DLC = len;
    for(i=0;i<len;i++)
        TxMessage.Data[i]=data[i];
    CAN_Transmit(CHASSIS_CAN, &TxMessage);
}

void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index) {
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

void buffer_append_int16(uint8_t* buffer, int16_t number, int16_t *index) {
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}
```

5.1.1 占空比模式

占空比模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]
范围	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	占空比 25-32 位	占空比 17-24 位	占空比 9-16 位	占空比 1-8 位

```
void comm_can_set_duty(uint8_t controller_id, float duty) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(duty * 100000.0), &send_index);
    comm_can_transmit_eid(controller_id |
                          ((uint32_t)CAN_PACKET_SET_DUTY << 8), buffer, send_index);
}
```

5.1.2 电流环模式

电流环模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]
范围	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	电流 25-32 位	电流 17-24 位	电流 9-16 位	电流 1-8 位

其中，电流数值为 int32 类型，数值-60000-60000 代表-60-60A。

电流环模式发送例程

```
void comm_can_set_current(uint8_t controller_id, float current) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);
    comm_can_transmit_eid(controller_id |
                ((uint32_t)CAN_PACKET_SET_CURRENT << 8), buffer, send_index);
}
```

5.1.3 电流刹车模式

电流刹车模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]
范围	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	刹车电流 25-32 位	刹车电流 17-24 位	刹车电流 9-16 位	刹车电流 1-8 位

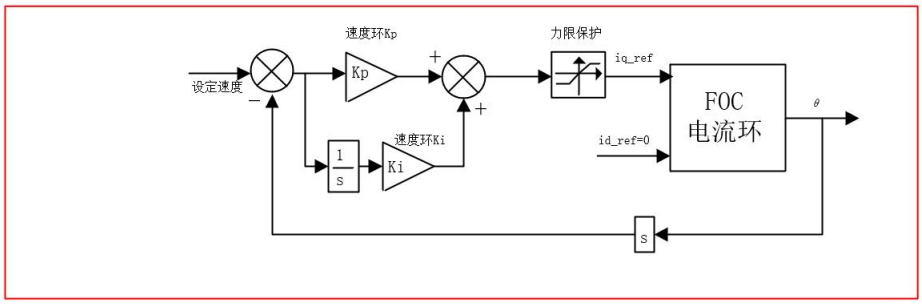
其中，刹车电流数值为 int32 类型，数值 0-60000 代表 0-60A。

电流刹车模式发送例程

```
void comm_can_set_cb(uint8_t controller_id, float current) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);
    comm_can_transmit_eid(controller_id |
            ((uint32_t)CAN_PACKET_SET_CURRENT_BRAKE << 8), buffer, send_index);
}
```

5.1.4 速度环模式

速度环简略控制框图



速度环模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]
范围	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	速度 25-32 位	速度 17-24 位	速度 9-16 位	速度 1-8 位

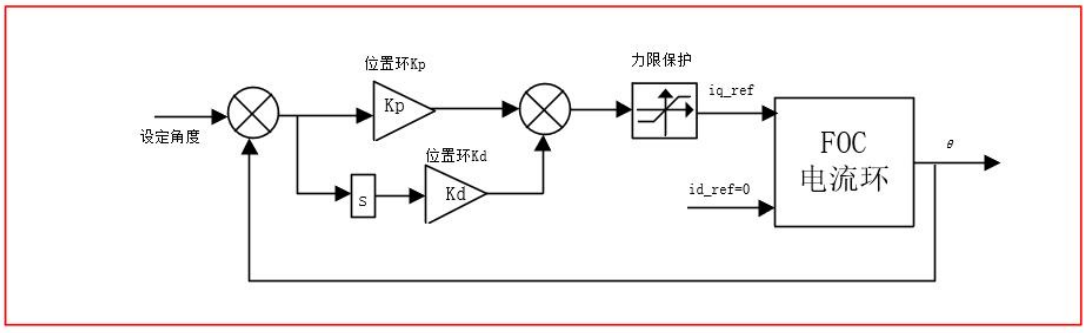
其中，速度数值为 int32 型，范围-100000-100000 代表-100000-100000 电气转速。

速度环发送例程

```
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);
}
```

5.1.5 位置环模式

位置环简略控制框图



位置环模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]
范围	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	位置 25-32 位	位置 17-24 位	位置 9-16 位	位置 1-8 位

其中，位置为 int32 型，范围-360000000-360000000 代表位置-36000°~36000°；

位置环发送例程

```
void comm_can_set_pos(uint8_t controller_id, float pos) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    comm_can_transmit_eid(controller_id |
            ((uint32_t)CAN_PACKET_SET_POS << 8), buffer, send_index);
}
```

5.1.6 设置原点模式

数据位	Data[0]
范围	0~0x02
对应变量	设置指令

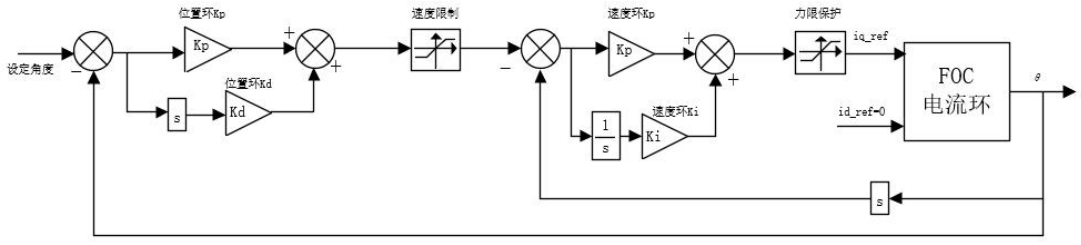
其中，设置指令为 uint8_t 型，0 代表设置临时原点(断电消除)，1 代表设置永久零点(参数自动保存);

位置环发送例程

```
void comm_can_set_origin(uint8_t controller_id, uint8_t set_origin_mode) {
    int32_t send_index = 0;
    uint8_t buffer;
    buffer[0]=set_origin_mode；
    comm_can_transmit_eid(controller_id |
            ((uint32_t) CAN_PACKET_SET_ORIGIN_HERE << 8), &buffer, send_index);
}
```

5.1.7 位置速度环模式

位置速度环简略框图



位置速度环模式发送数据定义

数据位	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
范围	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff

对应变量	位置 25-32 位	位置 17-24 位	位置 9-16 位	位置 1-8 位	速度高八位	速度低八位	加速度高八位	加速度低八位

其中，位置为 int32 型，范围-360000000~360000000 对应-位置-36000°~36000°；
其中，速度为 int16 型，范围-32768~32767 对应-327680~-327680 电气转速；
其中，加速度为 int16 型，范围 0~32767，对应 0~327670，1 单位等于 10 电气转速/s^2。

```
void comm_can_set_pos_spd(uint8_t controller_id, float pos,int16_t spd, int16_t RPA ) {
    int32_t send_index = 0;
    uint8_t buffer[8];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    buffer_append_int16(buffer,spd/10.0, & send_index);
    buffer_append_int16(buffer,RPA/10.0, & send_index);
    comm_can_transmit_eid(controller_id |
            ((uint32_t)CAN_PACKET_SET_POS_SPD << 8), buffer, send_index);
}
```

5.2 伺服模式电机报文格式

5.2.1 伺服模式 CAN 上传报文协议
伺服模式下电机 CAN 报文使用定时上传模式，上传频率可设置为 1~500HZ，上传字节为 8 字节

数据位	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
范围	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
对应变量	位置高八位	位置低八位	速度高八位	速度低八位	电流高八位	电流低八位	电机温度	报错代码

其中，位置为 int16 型，范围-32000~32000 代表位置-3200°~3200°；
其中，速度为 int16 型，范围-32000~32000 代表-320000~320000rpm 电气转速；
其中，电流为 int16 类型，数值-6000~6000 代表-60~60A
其中，温度为 int8 型，范围-20~127 代表驱动板温度-20℃~127℃；
其中，报错代码为 uint8 型，0 表示无故障，1 表示电机过温度故障，2 表示过电流故障，3 表示过压故障，4 表示欠压故障，5 表示编码器故障，6 表示 mos 管过温度故障，7 表示电机堵转；

以下为报文接受实例

```
void motor_receive(float* motor_pos,float*
motor_spd,float* cur,int_8* temp,int_8* error,rx_message)
  {
      int16_t pos_int = (rx_message)->Data[0] << 8 | (rx_message)->Data[1]);
      int16_t spd_int = (rx_message)->Data[2] << 8 | (rx_message)->Data[3]);
```

```
    int16_t cur_int = (rx_message)->Data[4] << 8 | (rx_message)->Data[5]);
    &motor_pos= (float)( pos_int * 0.1f); //电机位置
    &motor_spd= (float)( spd_int * 10.0f);//电机速度
    &motor_cur= (float) ( cur_int * 0.01f);//电机电流
    &motor_temp= (rx_message)->Data[6] ;//电机温度
    &motor_error= (rx_message)->Data[7] ;//电机故障码
  }
```

5.2.2 伺服模式串口报文协议

伺服模式串口收发报文协议如下

帧头（0x02）	数据长度（不含帧头帧尾和校验位）	数据帧	数据位	校验高 8 位	校验低 8 位	帧尾（0x03）

校验位计算代码参考 32 页

数据帧定义

```
typedef enum {
    COMM_FW_VERSION = 0,
    COMM_JUMP_TO_BOOTLOADER,
    COMM_ERASE_NEW_APP,
    COMM_WRITE_NEW_APP_DATA,
    COMM_GET_VALUES,      //获取电机运行参数
    COMM_SET_DUTY,        //电机以占空比模式运行
    COMM_SET_CURRENT,   //电机以电流环模式运行
    COMM_SET_CURRENT_BRAKE, //电机电流刹车模式模式运行
    COMM_SET_RPM,          //电机以速度环模式运行
    COMM_SET_POS,          //电机以位置环模式运行
    COMM_SET_HANDBRAKE,    //电机以手刹电流环模式运行
    COMM_SET_DETECT,       //电机实时反馈当前位置指令
    COMM_ROTOR_POSITION=22,//电机反馈当前位置
    COMM_GET_VALUES_SETUP=50,//电机单个或多个参数获取指令
    COMM_SET_POS_SPD=91,      //  电机以位置速度环模式运行
    COMM_SET_POS_MULTI=92,    //  设置电机运动为单圈模式
    COMM_SET_POS_SINGLE=93,   //   设置电机运动为多圈模式 范围±100 圈
    COMM_SET_POS_UNLIMITED=94, //保留
    COMM_SET_POS_ORIGIN=95, //设置电机原点
} COMM_PACKET_ID;
```

一、获取电机参数实例

串口指令 ：02 01 04 40 84 03 // 获取电机参数指令 电机接收后反馈一次电机状态

电机回传串口指令实例：02 49 04 01 66 FC D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 FF FF FF F3 00 F6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 16 D7 AD 00 0A 6F 19 40 7E 00 00 00 00 00 00 00 00 00 00 00 00 00 04 4D 53 03 02 05 16 00 1A B6 03 C9 B5 03

// 02（帧头）+49（数据长度）+ 04（数据帧） + mos 温度（2byte）+电机温度（2byte）+输出电流（4byte）+输入电流（4byte）+Id 电流（4byte）+Iq 电流（4byte）+电机油门值（2byte）+电机转速（4byte）+输入电压（2byte）+保留值（24byte）+电机状态码（1byte）+电机外环位置值（4byte）+电机控制 ID 号（1byte）+温度保留值（6byte）+Vd 电压值（4byte）+Vq 电压值（4byte）+CRC+03(帧尾)

电机发出的部分参数转化公式如下
MOS 温度=(float)buffer_get_int16(data, &ind) / 10.0)
电机温度=(float)buffer_get_int16(data, &ind) / 10.0)
输出电流=(float)buffer_get_int32(data, &ind) / 100.0)
输入电流=(float)buffer_get_int32(data, &ind) / 100.0)
Id 电流=(float)buffer_get_int32(data, &ind) / 100.0)
Iq 电流=(float)buffer_get_int32(data, &ind) / 100.0)
电机油门值=(float)buffer_get_int16(data, &ind) / 1000.0)
电机转速=(float)buffer_get_int32(data, &ind))
输入电压=(float)buffer_get_int16(data, &ind) / 10.0)
电机外环位置=(float)buffer_get_int32(data, &ind) / 1000000.0)
电机 ID 号=data
电机 Vd 电压=(float)buffer_get_int32(data, &ind) / 1000.0)
电机 Vq 电压=(float)buffer_get_int32(data, &ind) / 1000.0)

电机反馈位置报文指令
串口指令：02 02 0B 04 9C 7E 03 // 电机接收后隔 10ms 发送一次当前位置

电机反馈位置值发送实例（需要提前向电机发反馈位置报文指令，电机接收后每隔 10ms 发送一次当前位置）
串口指令：02 05 16 00 1A B6 64 D5 F4 03
Pos=(float)buffer_get_int32(data, &ind) / 1000.0

电机单个或多个参数获取指令实例
串口指令 02 05 32 00 00 00 01 58 4C 03 // 获取电机温度指令
指令说明： 此指令可以获取单个 或者多个电机参数 获取参数由数据段 4 字节 位决定，对应位位 1 时，电机将回传对应位的电机参数，为 0 时除去该字段
位对应的电机参数如下表格

32-19 位	18 位	17 位	16 位	10-15 位	9 位	8 位	7 位
保留值	电机 ID（1byte）	电机位置（4byte）	电机错误标志（1byte）	保留值	输入电压（2byte）	电机转速（4byte）	占空比（2byte）

6 位	5 位	4 位	3 位	2 位	1 位		
Iq 电流（4byte）	Id 电流（4byte）	输入电流（4byte）	输出电流（4byte）	电机温度（2byte）	MO 温度（2byte）		

电机收到该指令后会发出对应参数
实例：02 03 32 00 81 2A 6C 03 // 反馈电机温度
电机发出的部分参数转化公式如下
MOS 温度=(float)buffer_get_int16(data, &ind) / 10.0)
电机温度=(float)buffer_get_int16(data, &ind) / 10.0)
输出电流=(float)buffer_get_int32(data, &ind) / 100.0)
输入电流=(float)buffer_get_int32(data, &ind) / 100.0)
电机油门值=(float)buffer_get_int16(data, &ind) / 1000.0)
电机转速=(float)buffer_get_int32(data, &ind))
输入电压=(float)buffer_get_int16(data, &ind) / 10.0)
电机位置=(float)buffer_get_int32(data, &ind) / 1000000.0)
电机 ID 号=data
电机错误状态码

```
typedef enum {
    FAULT_CODE_NONE = 0,
    FAULT_CODE_OVER_VOLTAGE,// 过压
    FAULT_CODE_UNDER_VOLTAGE,// 欠压
    FAULT_CODE_DRV,// 驱动故障
    FAULT_CODE_ABS_OVER_CURRENT,// 电机过流
    FAULT_CODE_OVER_TEMP_FET,// MOS 过温
    FAULT_CODE_OVER_TEMP_MOTOR,//电机过温
    FAULT_CODE_GATE_DRIVER_OVER_VOLTAGE,// 驱动过压
    FAULT_CODE_GATE_DRIVER_UNDER_VOLTAGE,// 驱动欠压
    FAULT_CODE_MCU_UNDER_VOLTAGE,// MCU 欠压
    FAULT_CODE_BOOTING_FROM_WATCHDOG_RESET,// 欠压
    FAULT_CODE_ENCODER_SPI,// SPI 编码器故障
    FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE,//编码器超限
    FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE,//编码器超限
    FAULT_CODE_FLASH_CORRUPTION,// FLASH 故障
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_1,// 电流采样通道 1 故障
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_2,// 电流采样通道 2 故障
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_3,// 电流采样通道 1 故障
    FAULT_CODE_UNBALANCED_CURRENTS,// 电流不平衡

} mc_fault_code;
```

二、控制命令实例：

占空比发送模式实例
串口指令：02 05 05 00 00 4E 20 29 F6 03 // 0.20 占空比

串口指令：02 05 05 FF FF B1 E0 77 85 03　// -0.20 占空比

Duty=(float)buffer_get_int32(data, &ind) / 100000.0)　　//值为 接收 4 位数据/10000.0

电流环发送模式实例
串口指令：02 05 06 00 00 13 88 8B 25 03　　// 5 A IQ 电流
串口指令：02 05 06 FF FF EC 78 E3 05 03　　// - 5 A IQ 电流
Current=(float)buffer_get_int32(data, &ind) / 1000.0　　//值为接收 4 位数据/1000.0

刹车电流发送模式实例
串口指令：02 05 07 00 00 13 88 21 74 03　　// 5A 刹车电流
串口指令：02 05 07 FF FF EC 78 49 54 03　　// - 5A 刹车电流
I_Brake=(float)buffer_get_int32(data, &ind) / 1000.0　　//值为接收 4 位数据/1000.0

速度环发送模式实例
串口指令：02 05 08 00 00 03 E8 2B 58 03　　// 1000 ERPM 电气转速
串口指令：02 05 08 FF FF FC 18 43 78 03　　// - 1000 ERPM 电气转速
Speed=(float)buffer_get_int32(data, &ind)　　//值为接收 4 位数据

位置环发送模式实例
串口指令：02 05 09 0A BA 95 00 1E E7 03　　//电机转动到 180 度
串口指令：02 05 09 05 5D 4A 80 7B 29 03　　//电机转动到 90 度
Pos=(float)buffer_get_int32(data, &ind) / 1000000.0　　//值为接收 4 位数据/1000000.0

手刹电流发送模式实例
串口指令：02 05 0A 00 00 13 88 00 0E 03　　//5A HB 电流 电气转速
串口指令：02 05 0A FF FF EC 78 68 2E 03　　//5A HB 电流 电气转速
HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0　　//值为接收 4 位数据/1000.0

位置速度环发送模式实例
串口指令：02 0D 5B 00 02 BF 20 00 00 13 88 00 00 75 30 A5 AC 03
/*
180 度 转速 5000ERPM 加速度 30000/S
数据段为　位置+ 速度 + 加速度
*/
Pos=(float)buffer_get_int32(data, &ind) / 1000.0) // 位置值为接收 4 位数据/1000.0
Speed=(float)buffer_get_int32(data, &ind) //值为接收 4 位数据
Acc_Speed=(float)buffer_get_int32(data, &ind)//值为接收 4 位数据

设置多圈模式发送实例
串口指令：02 05 5C 00 00 00 00 9E 19 03　//设电机位置环为多圈运行模式 ±100 圈

设置单圈模式发送实例
串口指令：02 05 5D 00 00 00 00 34 48 03 //设电机位置环为单圈运行模式 0-360 度

设置当前位置为 0 点发送实例
串口指令：02 02 5F 01 0E A0 03 //设电机当前位置环为位置环零参考点

串口最短距离回零指令
串口指令：02 05 65 00 00 00 00 3A 8B 03 //让电机最短距离回归相对零位

串口校验：

```
unsigned short crc16(unsigned char *buf, unsigned int len) {
  unsigned int i;
  unsigned short cksum = 0;
  for (i = 0; i < len; i++) {
    cksum = crc16_tab[(((cksum >> 8) ^ *buf++) & 0xFF)] ^ (
cksum << 8);
  }
  return cksum;
}
const unsigned short crc16_tab[] = { 0x0000, 0x1021, 0x2042,
  0x3063, 0x4084,
0x50a5, 0x60c6, 0x70e7, 0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c,
  0xd1ad,
0xe1ce, 0xf1ef, 0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294,
  0x72f7,
0x62d6, 0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff,
  0xe3de,
0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
  0xa56a,
0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d, 0x3653,
  0x2672,
0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4, 0xb75b, 0xa77a,
  0x9719,
0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc, 0x48c4, 0x58e5, 0x6886,
  0x78a7,
0x0840, 0x1861, 0x2802, 0x3823, 0xc9cc, 0xd9ed, 0xe98e, 0xf9af,
  0x8948,
0x9969, 0xa90a, 0xb92b, 0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71,
  0x0a50,
0x3a33, 0x2a12, 0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58,
  0xbb3b,
```

```
0xab1a,  0x6ca6,  0x7c87,  0x4ce4,  0x5cc5,  0x2c22,  0x3c03,  0x0c60,
  0x1c41,
0xedae,  0xfd8f,  0xcdec,  0xddcd,  0xad2a,  0xbd0b,  0x8d68,  0x9d49,
  0x7e97,
0x6eb6,  0x5ed5,  0x4ef4,  0x3e13,  0x2e32,  0x1e51,  0x0e70,  0xff9f,
  0xefbe,
0xdfdd,  0xcffc,  0xbf1b,  0xaf3a,  0x9f59,  0x8f78,  0x9188,  0x81a9,
  0xb1ca,
0xa1eb,  0xd10c,  0xc12d,  0xf14e,  0xe16f,  0x1080,  0x00a1,  0x30c2,
  0x20e3,
0x5004,  0x4025,  0x7046,  0x6067,  0x83b9,  0x9398,  0xa3fb,  0xb3da,
  0xc33d,
0xd31c,  0xe37f,  0xf35e,  0x02b1,  0x1290,  0x22f3,  0x32d2,  0x4235,
  0x5214,
0x6277,  0x7256,  0xb5ea,  0xa5cb,  0x95a8,  0x8589,  0xf56e,  0xe54f,
  0xd52c,
0xc50d,  0x34e2,  0x24c3,  0x14a0,  0x0481,  0x7466,  0x6447,  0x5424,
  0x4405,
0xa7db,  0xb7fa,  0x8799,  0x97b8,  0xe75f,  0xf77e,  0xc71d,  0xd73c,
  0x26d3,
0x36f2,  0x0691,  0x16b0,  0x6657,  0x7676,  0x4615,  0x5634,  0xd94c,
  0xc96d,
0xf90e,  0xe92f,  0x99c8,  0x89e9,  0xb98a,  0xa9ab,  0x5844,  0x4865,
  0x7806,
0x6827,  0x18c0,  0x08e1,  0x3882,  0x28a3,  0xcb7d,  0xdb5c,  0xeb3f,
  0xfb1e,
0x8bf9,  0x9bd8,  0xabbb,  0xbb9a,  0x4a75,  0x5a54,  0x6a37,  0x7a16,
  0x0af1,
0x1ad0,  0x2ab3,  0x3a92,  0xfd2e,  0xed0f,  0xdd6c,  0xcd4d,  0xbdaa,
  0xad8b,
0x9de8,  0x8dc9,  0x7c26,  0x6c07,  0x5c64,  0x4c45,  0x3ca2,  0x2c83,
  0x1ce0,
0x0cc1,  0xef1f,  0xff3e,  0xcf5d,  0xdf7c,  0xaf9b,  0xbfba,  0x8fd9,
  0x9ff8,
0x6e17,  0x7e36,  0x4e55,  0x5e74,  0x2e93,  0x3eb2,  0x0ed1,  0x1ef0
  };


//int16 数据位整理
void buffer_append_int16(uint8_t* buffer, int16_t number, int32_t *index)
{
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}
```

```
//uint16 数据位整理
void buffer_append_uint16(uint8_t* buffer, uint16_t number, int32_t
*index) {
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

//int32 数据位整理
void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index)
{
        buffer[(*index)++] = number >> 24;
        buffer[(*index)++] = number >> 16;
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

//uint32 数据位整理
void buffer_append_uint32(uint8_t* buffer, uint32_t number, int32_t
*index) {
        buffer[(*index)++] = number >> 24;
        buffer[(*index)++] = number >> 16;
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

//int64 数据位整理
void buffer_append_int64(uint8_t* buffer, int64_t number, int32_t *index)
{
        buffer[(*index)++] = number >> 56;
        buffer[(*index)++] = number >> 48;
        buffer[(*index)++] = number >> 40;
        buffer[(*index)++] = number >> 32;
        buffer[(*index)++] = number >> 24;
        buffer[(*index)++] = number >> 16;
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

//uint64 数据位整理
void buffer_append_uint64(uint8_t* buffer, uint64_t number, int32_t
*index) {
        buffer[(*index)++] = number >> 56;
        buffer[(*index)++] = number >> 48;
```

```
        buffer[(*index)++] = number >> 40;
        buffer[(*index)++] = number >> 32;
        buffer[(*index)++] = number >> 24;
        buffer[(*index)++] = number >> 16;
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

//CRC 校验
unsigned short crc16(unsigned char *buf, unsigned int len) {
    unsigned int i;
    unsigned short cksum = 0;
    for (i = 0; i < len; i++) {
        cksum = crc16_tab[(((cksum >> 8) ^ *buf++) & 0xFF)] ^ (cksum << 8);
    }
    return cksum;
}

 //数据包的整理发送
void packet_send_packet(unsigned char *data, unsigned int len, int handler_num) {
    int b_ind = 0;
    unsigned short crc;
    if (len > PACKET_MAX_PL_LEN) {
        return;
    }
    if (len <= 256) {
        handler_states[handler_num].tx_buffer[b_ind++] = 2;
        handler_states[handler_num].tx_buffer[b_ind++] = len;
    } else {
        handler_states[handler_num].tx_buffer[b_ind++] = 3;
        handler_states[handler_num].tx_buffer[b_ind++] = len >> 8;
        handler_states[handler_num].tx_buffer[b_ind++] = len & 0xFF;
    }

    memcpy(handler_states[handler_num].tx_buffer + b_ind, data, len);
    b_ind += len;

    crc = crc16(data, len);
    handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc >> 8);
    handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc & 0xFF);
    handler_states[handler_num].tx_buffer[b_ind++] = 3;

    if (handler_states[handler_num].send_func) {
        handler_states[handler_num].send_func(handler_states[handler_num].tx_buffer,
```

```
b_ind);
    }
}
```

5.3 运控模式通讯协议

特殊 can 代码

进入电机控制模式 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }
退出电机控制模式 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFD }
设置电机当前位置为 0 点 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFE }

注意：使用 CAN 通信控制电机时必须先进入电机控制模式！

ps：（如果是在无状态下想要读取当前的状态，发送的命令是
{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }）

运控模式驱动板接收数据定义

标识符： 设置的电机 ID 号（默认为 1） 帧类型：标准帧
帧格式： DATA DLC: 8 字节

数据域	DATA[0]	DATA[1]	DATA[2]	DATA[3]	
数据位	7-0	7-0	7-0	7-4	3-0
数据内容	电机位置高 8 位	电机位置低 8 位	电机速度高 8 位	电机速度低 4 位	KP 值高 4 位

数据域	DATA[4]	DATA[5]	DATA[6]		DATA[7]
数据位	7-0	7-0	7-4	3-0	0-7
数据内容	KP 值低 8 位	KD 值高 8 位	KD 值低 4 位	电流值高 4 位	电流值低 8 位

运控模式驱动板发送数据定义

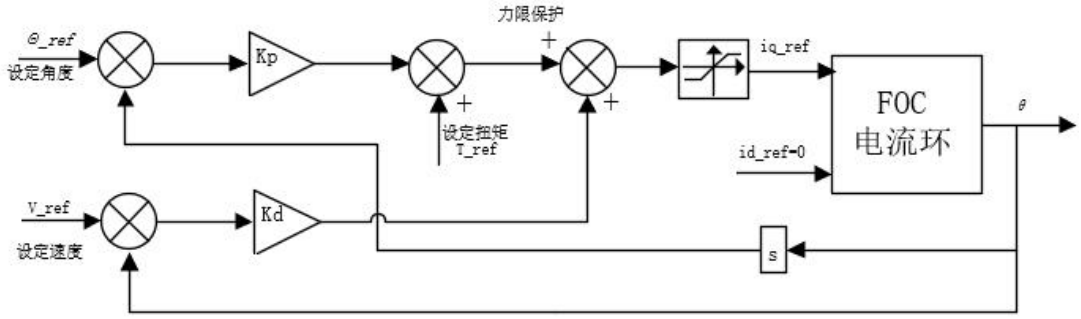
标识符： 0X00+驱动器 ID 帧类型：标准帧
帧格式： DATA DLC: 8 字节

数据域	DATA[0]	DATA[1]	DATA[2]	DATA[3]	DATA[4]
数据位	7-0	7-0	7-0	7-0	7-4
数据内容	驱动器 ID 号	电机位置高 8 位	电机位置低 8 位	电机速度高 8 位	电机速度低 4 位

数据域	DATA[4]	DATA[5]	DATA[6]	DATA[7]
数据位	3-0	7-0	7-0	7-0
数据内容	电流值高 4 位	电流值低 8 位	电机温度	电机错误标志

CAN 速率：1 MHZ

运控模式简略框图



参数范围

模组电机	AK10-9	AK60-6	AK70-10	AK80-6	AK80-9	AK80-80	AK80-8
电机位置(rad)	-12.5f-12.5f						
电机速度（rad/s）	-50.0f-50.0f	-45.0f-45.0f	-50.0f-50.0f	-76.0f-76.0f	-50.0f-50.0f	-8.0f-8.0f	-37.5f-37.5f
电机扭矩（N.M）	-65.0f-65.0f	-15.0f-15.0f	-25.0f-25.0f	-12.0f-12.0f	-18.0f-18.0f	-144.0f-144.0f	-32.0f-32.0f
Kp 范围	0-500						
Kd 范围	0-5						

运控模式收发代码例程

发送例程代码

```
void pack_cmd(CANMessage * msg, float p_des, float v_des, float kp, float kd, float t_ff){
    /// limit data to be within bounds ///
    float P_MIN =-12.5f;
    float P_MAX =12.5f;
    float V_MIN =-30.0f;
    float V_MAX =30.0f;
    float T_MIN =-18.0f;
    float T_MAX =18.0f;
    float Kp_MIN =0;
    float Kp_MAX =500.0f;
    float Kd_MIN =0;
    float Kd_MAX =5.0f;
    float Test_Pos=0.0f;

    p_des = fminf(fmaxf(P_MIN, p_des), P_MAX);
    v_des = fminf(fmaxf(V_MIN, v_des), V_MAX);
    kp = fminf(fmaxf(Kp_MIN, kp), Kp_MAX);
    kd = fminf(fmaxf(Kd_MIN, kd), Kd_MAX);
    t_ff = fminf(fmaxf(T_MIN, t_ff), T_MAX);
    /// convert floats to unsigned ints ///

    int p_int = float_to_uint(p_des, P_MIN, P_MAX, 16);
    int v_int = float_to_uint(v_des, V_MIN, V_MAX, 12);
    int kp_int = float_to_uint(kp, KP_MIN, KP_MAX, 12);
    int kd_int = float_to_uint(kd, KD_MIN, KD_MAX, 12);
    int t_int = float_to_uint(t_ff, T_MIN, T_MAX, 12);

    /// pack ints into the can buffer ///
    msg->data[0] = p_int>>8;              //位置高 8
    msg->data[1] = p_int&0xFF;    //位置低 8
    msg->data[2] = v_int>>4;              //速度高 8 位
    msg->data[3] = ((v_int&0xF)<<4)|(kp_int>>8);   //速度低 4 位 KP 高 4 位
    msg->data[4] = kp_int&0xFF;   //KP 低 8 位
    msg->data[5] = kd_int>>4;     //Kd 高 8 位
    msg->data[6] = ((kd_int&0xF)<<4)|(t_int>>8);       //KP 低 4 位扭矩高 4 位
    msg->data[7] = t_int&0xff;     //扭矩低 8 位
```

```
}
```

发包时所有的数都要经以下函数转化成整型数之后再发给电机。

```
int float_to_uint(float x, float x_min, float x_max, unsigned int bits){
    /// Converts a float to an unsigned int, given range and number of bits ///
    float span = x_max - x_min;
    if(x < x_min) x = x_min;
    else if(x > x_max) x = x_max;
    return (int) ((x- x_min)*((float)((1<<bits)/span)));
}
```

接收例程代码

```
void unpack_reply(CANMessage msg){
    /// unpack ints from can buffer ///
    int id = msg.data[0]; //驱动 ID 号
    int p_int = (msg.data[1]<<8)|msg.data[2];                    //电机位置数据
    int v_int = (msg.data[3]<<4)|(msg.data[4]>>4);               //电机速度数据
    int i_int = ((msg.data[4]&0xF)<<8)|msg.data[5];              //电机扭矩数据
    Int T_int = msg.data[6] ;
    /// convert ints to floats ///
    float p = uint_to_float(p_int, P_MIN, P_MAX, 16);
    float v = uint_to_float(v_int, V_MIN, V_MAX, 12);
    float i = uint_to_float(i_int, -I_MAX, I_MAX, 12);
    float T =T_int;
    if(id == 1){
      postion = p;                    //根据 ID 号读取对应数据
      speed = v;
      torque = i;
      Temperature = T-40;     //温度范围-40~215
    }
  }
```

收包时所有的数都要经以下函数转化成浮点型。

```
float uint_to_float(int x_int, float x_min, float x_max, int bits){
    /// converts unsigned int to float, given range and number of bits ///
    float span = x_max - x_min;
    float offset = x_min;
    return ((float)x_int)*span/((float)((1<<bits)-1)) + offset;
}
```