HIGH TORQUE
高擎机电

HIGH TORQUE

高擎机电
伺服关节模组

行 星 系 列 调 试 手 册

www.hightorque.cn

1. 电机使用须知

1.1 驱动板信息

- 工作电压：12V-38V
- 双编码器：底部集成双编码器
- 通信接口：支持CAN和CAN-FD协议
- 电源指示灯

1.2 电机的硬件接线

1. 电机CAN-FD接口与FDCAN模块相连。
2. 连接电源，电源上电后指示灯亮说明驱动板正常工作。

注：请保证电机can线线序与FDCAN模块线序正确无误，当打开上位机通讯时，can模块灯会闪烁。

2. 相关使用方法

2.1 电机校准流程

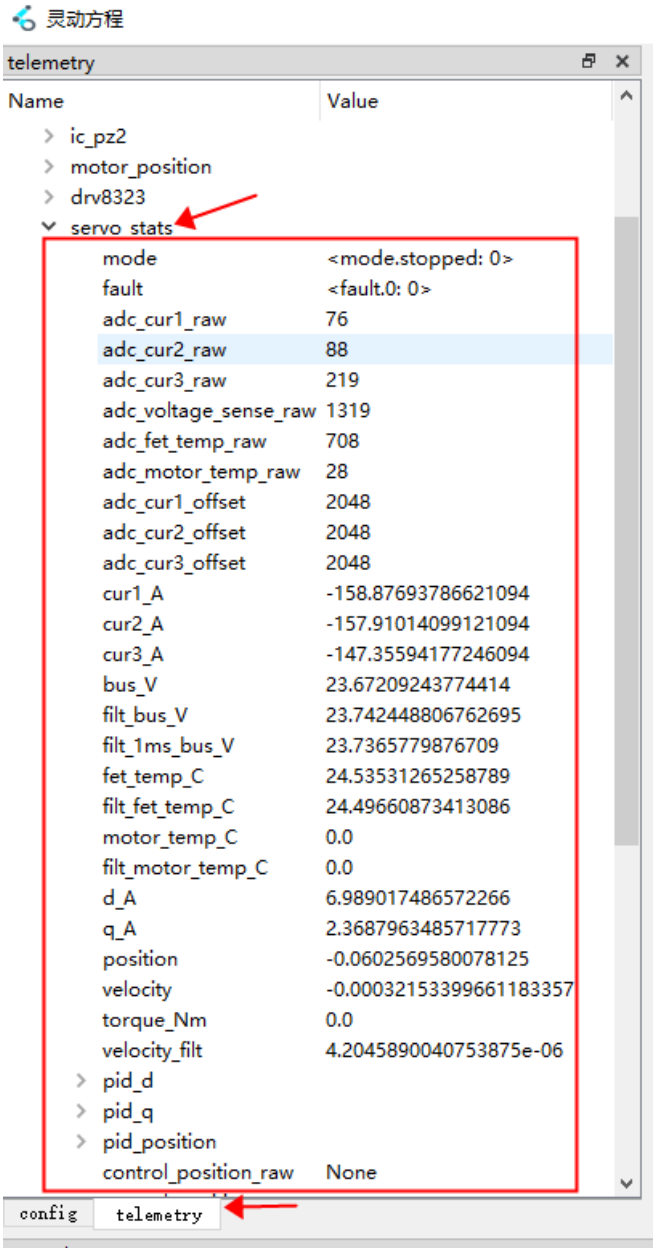
1. 硬件接线无误后，打开上位机软件，上位机会自动识别电机ID，默认初始ID是1，点击开始校准。(出厂的电机是已经校准好的)
2. 等待校准，上位机会输出校准信息，校准成功界面如下图所示。

2.2 电机参数状态查看

状态信息查看：校准成功后，打开上位机点击“**打开Tvie**”，在**telemetry**项里面，点开**servo_stats**选项卡，里面能查看电机当前各种状态等，如下图所示。

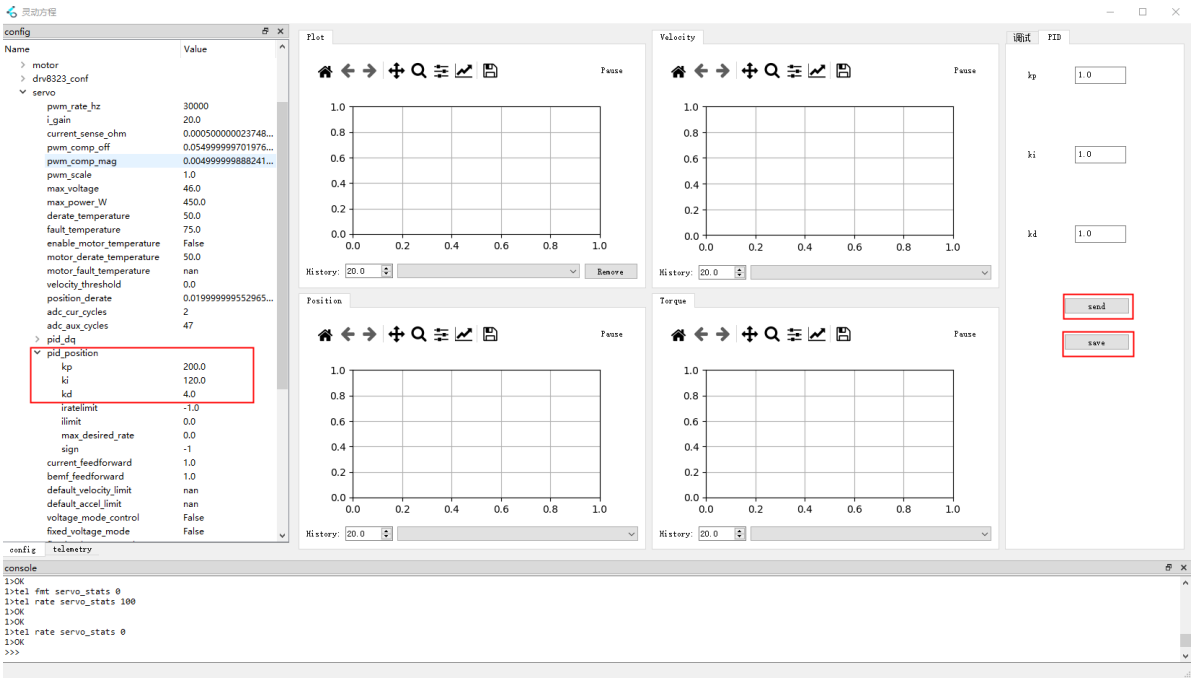
例如：

- 位置信息：在**telemetry**里面，点开**servo_stats**，找**position**,右键选择**plot right/left。**
- 速度信息：在**telemetry**里面，点开**servo_stats**，找**velocity**,右键选择**plot right/left。**
- 扭矩信息：在**telemetry**里面，点开**servo_stats**，找**torque_Nm**,右键选择**plot right/left。**



2.3 PID调参

在**config**项里配置参数后需要保存才能生效 :指令**conf write**或者在右侧PID调参中设置参数后依次点击**Send**和**save**才能保存生效，如下图所示。

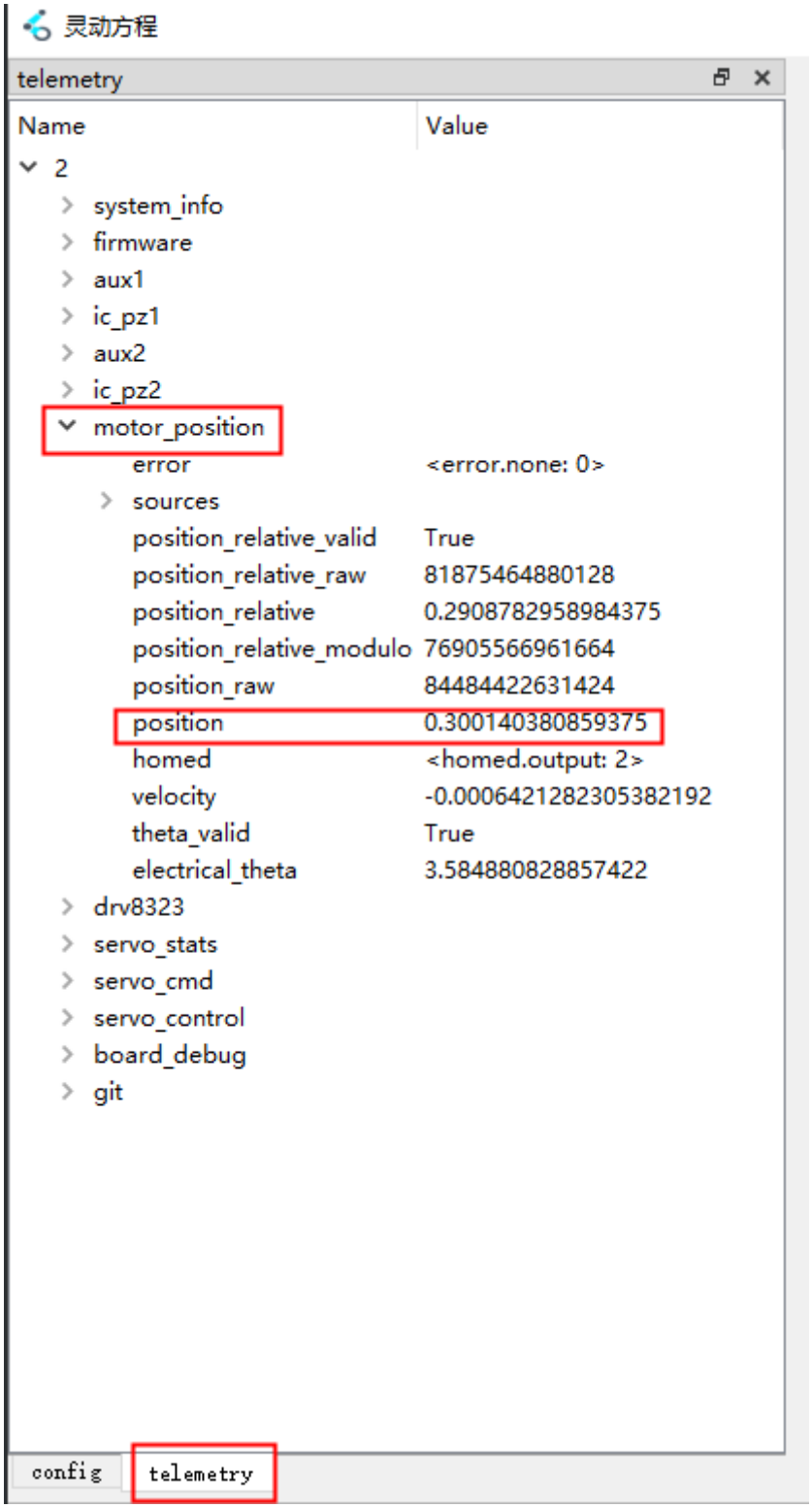


2.3.1 PID速度调参过程

1. 调出速度曲线。
2. **d pos （d pos nan 0.6 2）**指令，让电机先动起来,建议速度先设置小一点可以先给0.6，观察电机速度曲线与目标曲线的差距。
3. 开始调PID，观察速度曲线的变化。
4. 输入指令`d stop`让电机停下来，然后重新输入指令`d pos`让电机旋转，观察电机速度曲线和目标曲线的差距，不断重复步骤3，直至达到最佳。
5. 确定最终参数后，需要输入保存指令**conf write**保存参数。

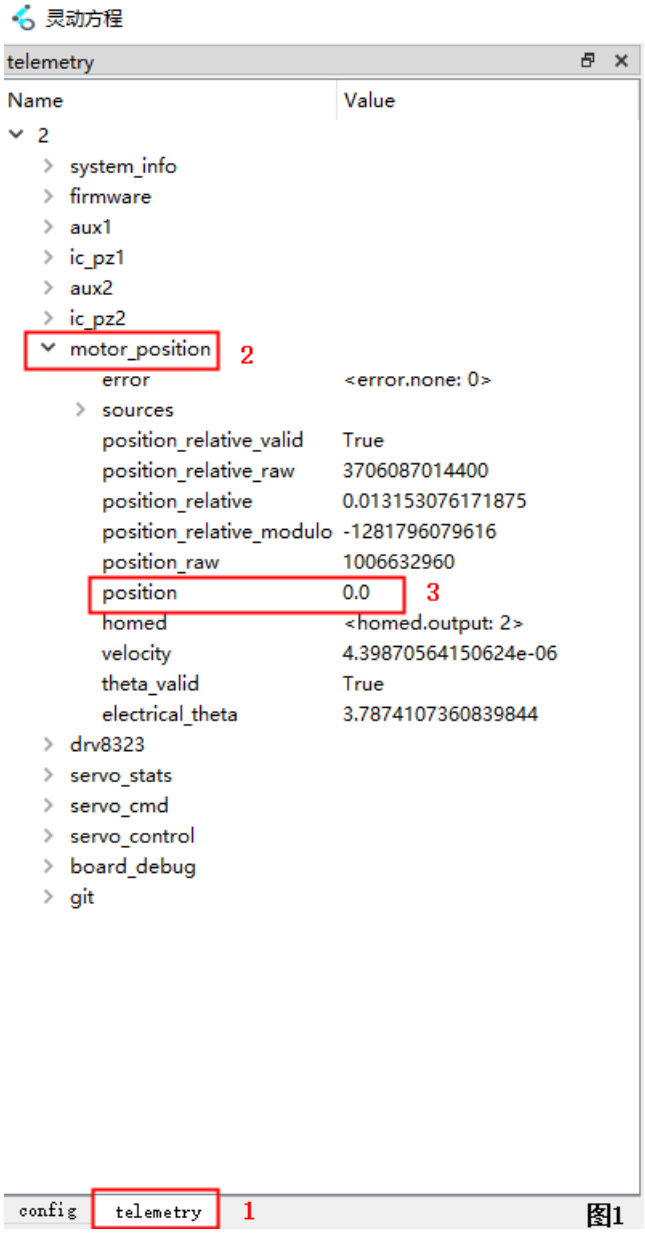
2.4 电机零点

电机零点记录的位置是-0.5-+0.5圈，超出此范围的位置重新上电之后会以单圈的位置算。可以在上位机**motor_position.position**中查看当前的位置值，当前的位置值为输出编码器的位置值，如下图所示。测试电机零点可把电机扭到-0.5-+0.5处位置，断电后重新上电，看位置是否为断电前的位置。也可以记录断电前位置，然后断电，扭动电机，重新上电，扭回去断电前的记录位置观察是否一样。



2.5 重设零位

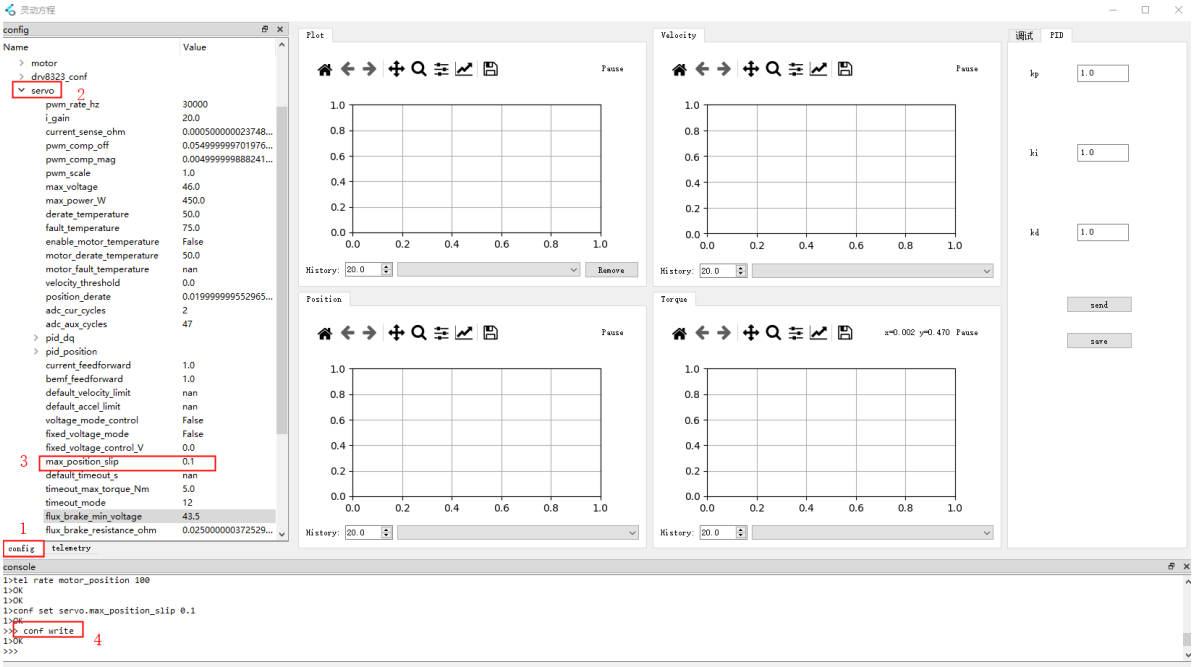
电机校准后都会有一个绝对的零位，打开上位机在**telemetry**项，电机**motor_position**选项卡，参数**position**为电机当前位置，如下图1所示，位置参数为0时即是电机转动的零位。如果想指定电机转动的位置为零位，可在上位机点击“**全部电机零位**”即可，如下图2所示，此时电机零位改变。


灵动方程
telemetry
Name
Value
2
system_info
firmware
aux1
ic_pz1
aux2
ic_pz2
motor_position
2
error
<error.none: 0>
sources
position_relative_valid
True
position_relative_raw
3706087014400
position_relative
0.013153076171875
position_relative_modulo
-1281796079616
position_raw
1006632960
position
0.0
3
homed
<homed.output: 2>
velocity
4.39870564150624e-06
theta_valid
True
electrical_theta
3.7874107360839844
drv8323
servo_stats
servo_cmd
servo_control
board_debug
git
config
telemetry
1
图1


灵动方程
CAN ID查询
canid
2
开始校准
打开Tview
当前电机零位
全部电机零位
校准信息
COM12
找到的电机id是 [2]
查询成功
图2

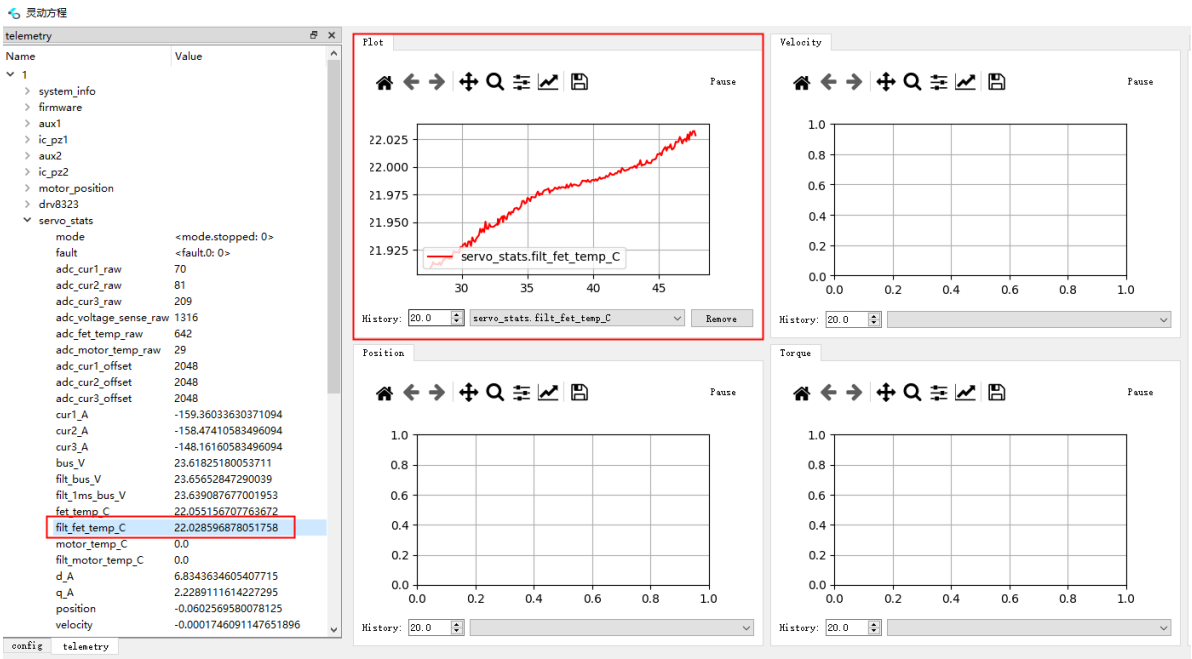
2.6 位置积分

设置电机的位置积分可在上位机的**config**项-**servo**选项卡中，找到参数**max_position_slip**，把原参数**nan**改成**0.1**，之后在命令行输入**conf write** 保存指令，上位机返回OK即可，如下图所示。修改该参数是防止电机在使用速度控制的时候被外力卡停，撤出外力之后会快速旋转不受之前的速度控制。



2.7 温度限制设置

驱动板因温度过高可能会限制输出的扭矩，可通过物理和软件的方法改善。查看温度曲线可视化状态可在**telemetry**项里面，点开**servo_stats**选项卡，找到参数**filt_fet_temp_C**,右键选择**plot right/left**，如图下所示。



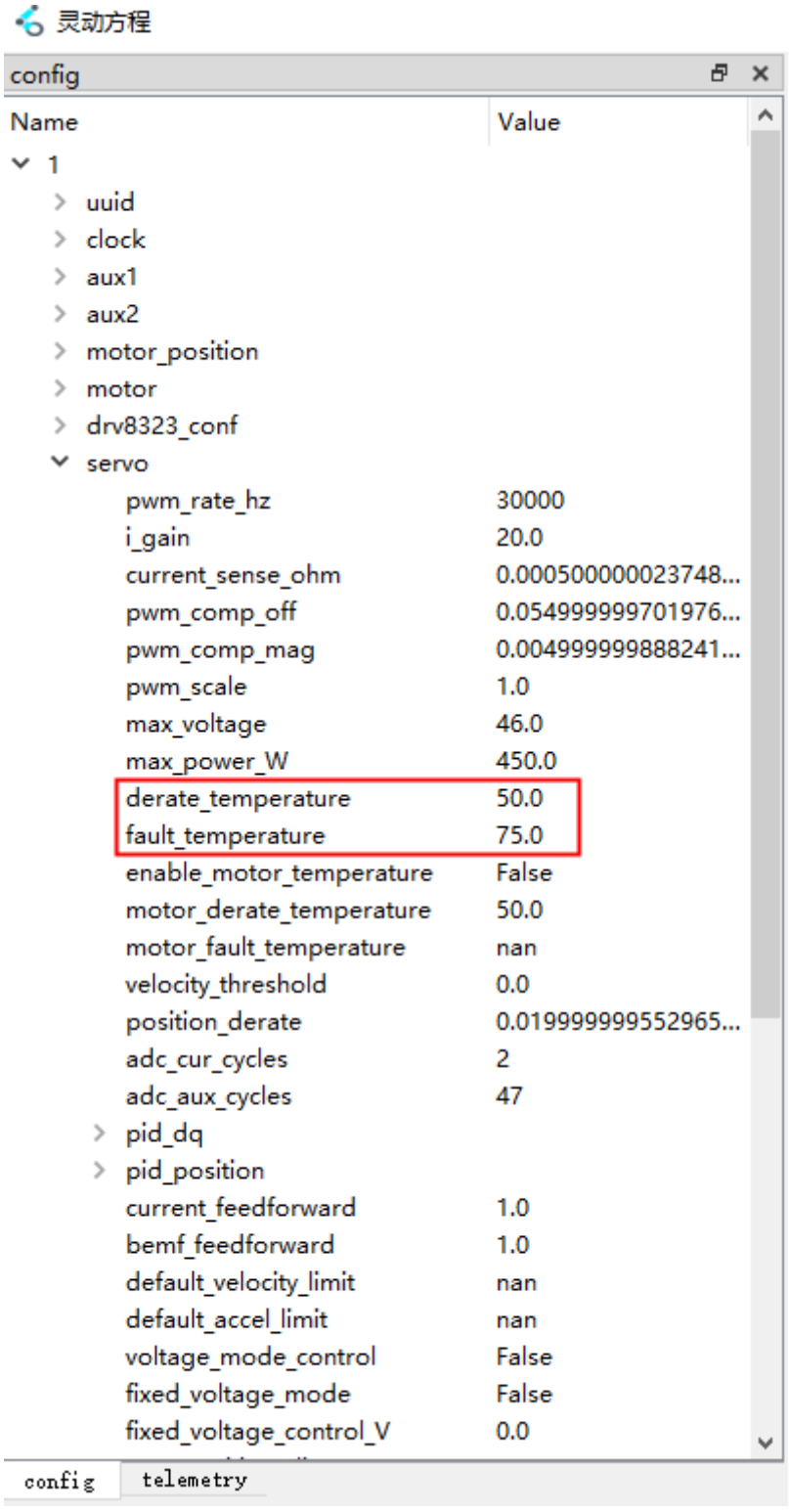
1. **物理办法：**
    - 加散热片
    - 散热装置
    - 冷空气等

2. **软件方法：**

你可以设置以下参数：

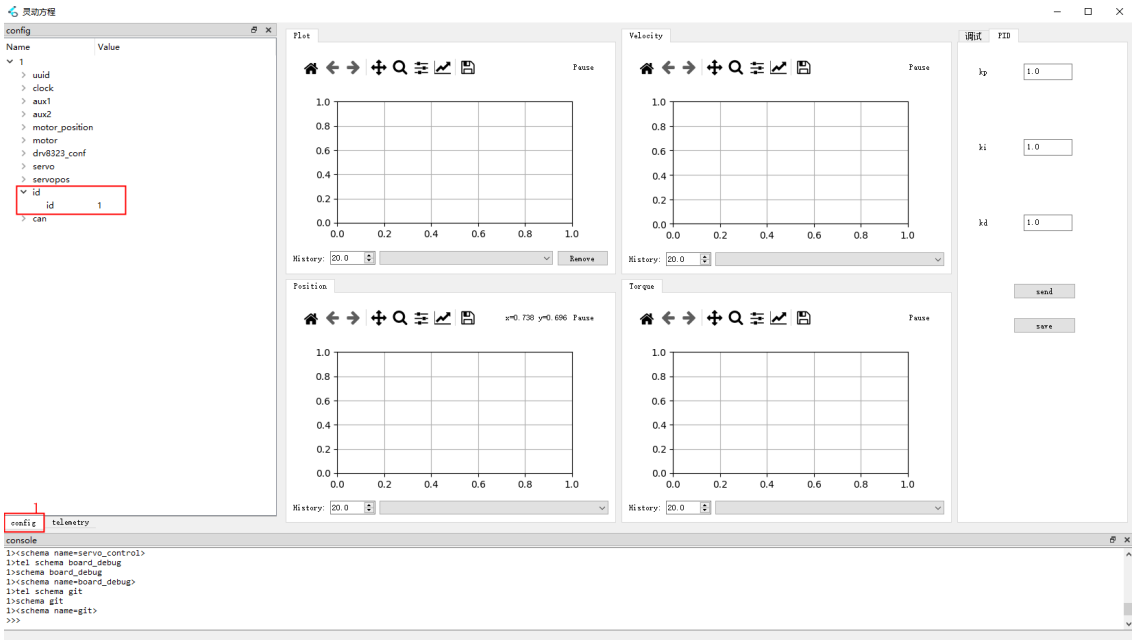
参数	功能
servo.derate_temperature	当温度达到此值时，扭矩开始受到限制
servo.fault_temperature	温度达到此值时，触发故障，停止所有扭矩

想要保证温度系数不受影响，首先要在良好的散热环境下运行（加散热片、散热铝壳、低温）等。其次，可以修改其受温度影响的扭矩参数来提高其性能**（必须在散热铝壳下进行）**，调高参数**servo.derate_temperature**，**servo.fault_temperature**的限制即可，如下图所示。

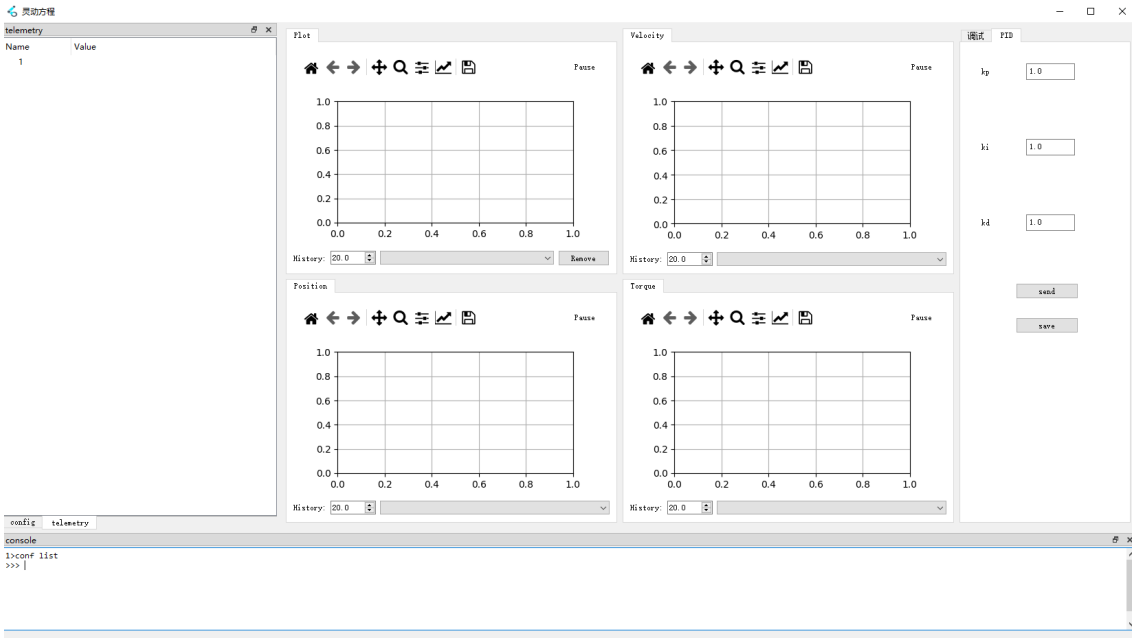


2.8 修改电机ID

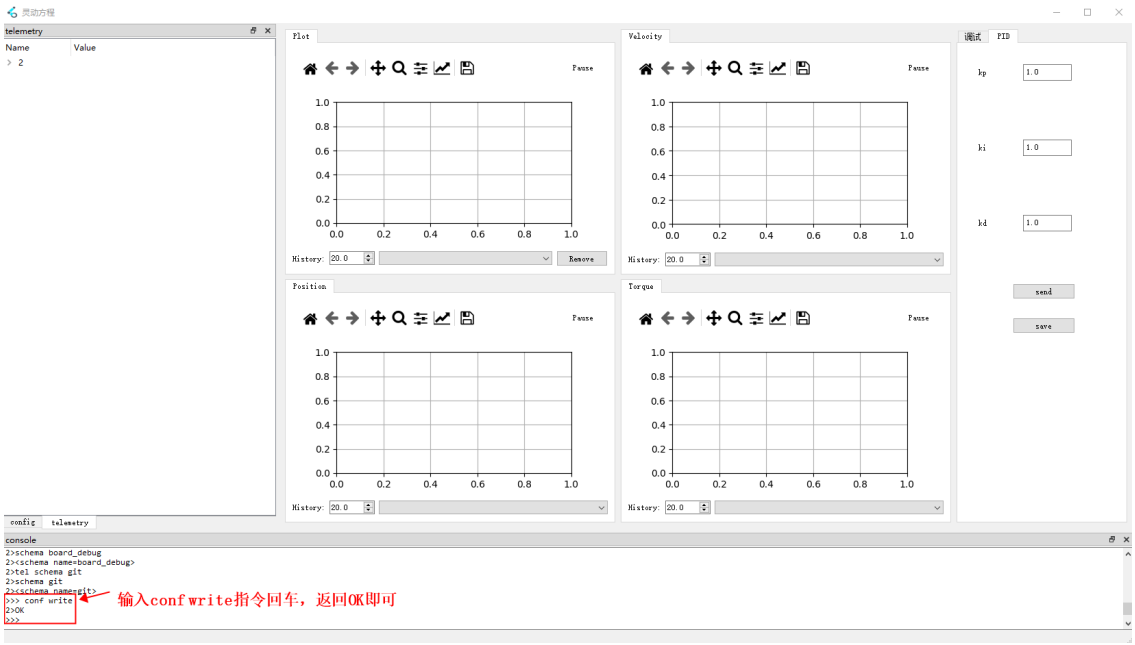
1. 连接电机，打开上位机读取电机ID后，点击“**打开Tview**”如下图所示，选择**config**项，打开id选项卡，修改id号后回车或点击空白处，控制台打印已修改的ID号。**注：修改ID的过程中不要断电！**



2. 修改后直接关闭当前窗口，再重新打开，可以发现上位机中没有驱动板信息如下图所示，因为已经修改了驱动板id为2。



3. 接下来我们把上位机的两个窗口都关闭，重新打开上位机，上位机会自动识别到更改后的ID。点击“**打开Tview**”，在上位机命令行中输入保存指令**conf write**保存ID，反馈**OK**后即为ID修改成功，如下图所示。



2.9 常用控制指令

2.9.1 FOC位置控制指令

- 语法：**d pos [pos] [vel] [max_torque] [options...]**

每个可选元素由一个前缀字符和一个值组成：

p - kp 比例：设置电机转动的钢性参数。

d - kd 比例：设置电机转动的阻尼参数。

s - 停止位置：当给定非零速度时，当控制位置达到该值时运动停止。

f - 前馈扭矩，单位 Nm。

t - timeout：如果在这么多秒内没有收到另一个命令，则进入超时模式。

v - 速度限制：给定值将在此命令期间覆盖全局速度限制。

a - 加速度限制：给定值将在此命令期间覆盖全局加速度限制。

o - 固定电压覆盖：在生效时，将控制视为使用给定的电压固定电压模式启用。

例：

指令	功能解释
d pos nan 1 3	以3n/m的扭矩,1圈/s的速度运动
d pos nan 1 3 s10	以3n/m的扭矩,1圈/s的速度运动到10的位置，然后停止

2.9.2 停止指令

指令	功能解释
d stop	让控制器进入停止状态，从而控制电机停止

2.9.3 电压模式(d pwm)

- 语法：**d pwm [phase] [magnitude] [phase_rate]**

功能解释：

- **Phase**是电向量方向（弧度）
- **mangnitdue**是强度（电压伏特）
- (可选)**phase_rate**参数控制向量转速（弧度每秒）

注意：给电压伏特时数值从0开始给，幅度0.1-0.2左右增加，太大电压可导致电机发烫甚至冒烟损坏电机，谨慎使用！

2.9.4 力矩模式

指令	功能解释
d pos nan 0 nan p0 d0 f[value]	f后参数为可调节的力矩参数

2.9.5 刹车指令

指令	功能解释
d brake	进入“刹车”状态。 在这种模式下，所有电机相位都对地短路，导致被动“制动”动作。

2.9.6 保存指令

指令	功能解释
conf write	将所有可配置参数的当前值从 RAM 写入内存

2.9.7 位置限制(d within)

- 语法：**d within [value] [value] [value]**

例：

指令	功能解释
d within -0.4 0.5 0.3	-0.4,0.5为设定的位置上限和下限当超出设定的界限时，将会以0.3的扭力去限制扭动
d within nan 0.5 0.3	无位置上限和0.5的下限，当超出设定的界限时，将会以0.3的扭力去限制扭动

3 Lively board通讯协议说明

3.1 CAN-FD协议解析

3.1.1 CAN-FD 相关说明

1. CAN-FD 波特率：
    - 仲裁段：1 Mbps
    - 数据段：最高支持 5 Mbps，也可用 1Mbps。
2. ID：由 16 位构成，其中 `0x7F` 是广播地址。
    - 高 8 位：表示**源地址**：
        - 最高位为 1 ：需要回复。
        - 最高位位 0：无需回复。
        - 其余 7 位：信号源地址。
    - 低 8 位：表示**目的地址**：
        - 最高为 0。
        - 其余 7 位表示目的地址。

例如：

1. `ID：0x8001`
    - 信号源地址为 0。
    - 目的地址为 1。
    - 最高位为 1，表示需要回复。
2. `ID：0x100`
    - 信号源地址为 1。
    - 目的地址为 0。
    - 最高位为 0，表示无需回复。

3.1.2 协议基本说明

1. 最小单位为子帧，每条子帧可以向**一个或多个**寄存器写入值，或读取数据。
2. 一条 CAN-FD 帧可由**一个或多个**子帧组成。
3. 电机各个功能实现通过在一条 FDCAN 帧中写入**一个或多个**对应功能的寄存器的值实现。
4. 可向任意寄存器写入 `int8_t、int16_t、int32_t、float` 四种基本数据类型。
5. 在**同一子帧**中的数据类型必需相同，**不同子帧**中的数据类型可以不同。
6. 所有基本类型的传输都为**小端模式**，即先发送低字节的数据，再发送高字节的数据。
7. 在一条 CAN-FD 中需要在**尾部填充**字节是应使用 `0x50`，表示无任何操作（NOP）。
8. 各种数据类型的**无限制**：
    - `int8_t`：`0x80`
    - `int16_t`：`0x8000`
    - `int32_t`：`0x80000000`
    - `float`：`NAN`

3.1.3 子帧解析

3.1.3.1 发送协议

① 模式一

```
uint8_t tdata[] = {cmd, addr, a1, a2, b1, b2...};
```

1. `cmd`：表示读写、数据类型和个数。
    1. 高四位 `cmd[7:4]` 表示读、写、回读。
        1. `0x0x`：写
        2. `0x1x`：读
        3. `0x2x`：回复
    2. 低四位 `cmd[3:0]` 表示数据类型和个数：
        1. 高两位 `cmd[3:2]` 表示数据类型：
            1. `00`：`int8_t`
            2. `01`：`int16_t`
            3. `10`：`int32_t`
            4. `11`：`float`
        2. 低两位 `cmd[1:0]` 表示数据个数。
            1. `01`：一个数据。
            2. `10`：两个数据。
            3. `11`：三个数据。
            4. `00`：**模式二**标志。
2. `addr`：起始寄存器地址。
3. `a1, a2, b1, b2...`：向寄存器中写入的数据，注意：需满足 1.2 协议基本说明的 4、5 项。
    1. `a1, a2`：需要向 `addr` 中写入的数据。
    2. `b1, b2`：需要向 `addr + 1` 中写入的数据。
    3. `...`：需要向 `add + n` 中写入的数据。

② 模式二

```
uint8_t tdata[] = {cmd, num, addr, a1, a2, b1, b2...};
```

1. `cmd`：表示读写、数据类型和个数。
    1. 高四位 `cmd[7:4]` 表示读、写、回读。
        1. `0000`：写
        2. `0001`：读
        3. `0010`：回复
    2. 低四位 `cmd[3:0]` 表示数据类型和个数：
        1. 高两位 `cmd[3:2]` 表示数据类型：
            1. `00`：`int8_t`
            2. `01`：`int16_t`
            3. `10`：`int32_t`
            4. `11`：`float`
        2. 低两位 `cmd[1:0]` 固定为 `00`，表示模式二，后一字节表示数据个数。

2. `num`：数据个数。
3. `addr`：起始寄存器地址。
4. `a1, a2, b1, b2...`：向寄存器中写入的数据，注意：需满足 1.2 协议基本说明的 4、5 项。
    1. `a1, a2`：需要向 `addr` 中写入的数据。
    2. `b1, b2`：需要向 `addr + 1` 中写入的数据。
    3. `...`：需要向 `add + n` 中写入的数据。

模式二实质：在模式一的 `xxx` 中数据个数写为 0 时，使用后一字节表示数据个数，其余字节都往后移动一位。

3.1.3.2 接收协议

- **接受协议模式是由发送协议的模式决定的。**
- **接收协议模式和发送协议模式是相同的。**

① 模式一

假设获取的数据是 `uint16_t`

```
uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ..., cmd1, addr1, c1, c2, c3, c4}
```

- `cmd`：
    - 高四位 `cmd[7, 4]`：`0010` 表示回复。
    - 2~3 位 `cmd[3, 2]`：表示类型。
        - `00`：`int8_t` 类型。
        - `01`：`int16_t` 类型。
        - `10`：`int32_t` 类型。
        - `11`：`float` 类型。
    - 低 2 位 `cmd[1, 0]`：表示数量。
        - `00`：表示模式二。
        - `01`：一个数据。
        - `10`：两个数据。
        - `11`：三个数据。
- `addr`：开始获取的地址。
- `a1, a2`：数据 1，小端模式。
- `b1, b2`：数据 2，小端模式。

②模式二

```
uint8_t rdata[] = {cmd, addr, num, a1, a2, b1, b2, ..., cmd1, addr1, c1, c2, c3, c4}
```

- `cmd`：
    - 高四位 `cmd[7, 4]`：`0010` 表示回复。
    - 2~3 位 `cmd[3, 2]`：表示类型。
        - `00`：`int8_t` 类型。
        - `01`：`int16_t` 类型。

        - `10`：`int32_t` 类型。
        - `11`：`float` 类型。
    - 低 2 位 `cmd[1, 0]`：表示数量。
        - `00`：表示模式二，后一个字节表示数据数量。
- `num`：数据个数。
- `addr`：开始获取的地址。
- `a1, a2`：数据 1，小端模式。
- `b1, b2`：数据 2，小端模式。

3.1.3.3 示例

Ⅰ、发送协议示例

① 模式一

```
uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x0A, 0x20, 0x00, 0x00, 0x00, 0x80, 0x10,
0x27, 0x00, 0x00, 0x50, 0x50, 0x50};
```

该 CAN-FD 帧由两个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。
    - `0x01`：第一个子帧的开头
        - 高四位为 `0000`，表示写操作。
        - 低四位：
            - 高 2 位为：`00`，表示 `int8_t` 类型。
            - 低 2 位为：`01`，表示 1 个数据。
    - `0x00`：起始寄存器地址：查表可知，`0x00` 寄存器表示电机模式设置。
    - `0x0A`：往 `0x00` 寄存器写入 `0x0A`。
2. 子帧 2：整体意思是位置不限制，速度为 0.1 转/秒
    - `0x0A`：第 2 个子帧的开头
        - 高 8 位为 `0x0`，表示写操作。
        - 低 8 位：
            - 高 2 位为：`10`，表示 `int32_t` 类型。
            - 低 2 位为：`10`，表示 2 个数据。
    - `0x20`：起始寄存器地址：查表可知，`0x20` 寄存器表示位置，`0x21` 寄存器表示速度。
    - `0x00、0x00、0x00、0x80`：小端模式，即 `0x80000000` 写入 `0x20` 寄存器，即表示电机位置无限制。
    - `0x10、0x27、0x00、0x00`：小端模式，即 `0x2710` 写入 `0x21` 寄存器，表示电机速度设置为 0.1 转/秒。
3. `0x50, 0x50, 0x50`：由于 CAN-FD 发送字节数最近的两个是 12 和 16，固还需加上 3 个字节的占位字节。

② 模式二

```
uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x08, 0x02, 0x20, 0x00, 0x00, 0x00, 0x80,
0x10, 0x27, 0x00, 0x00, 0x50, 0x50};
```

该 CAN-FD 帧由两个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。
    - `0x01`：第一个子帧的开头
        - 高 8 位为 `0001`，表示写操作。
        - 低 8 位：
            - 高 2 位为：`00`，表示 `int8_t` 类型。
            - 低 2 位为：`01`，表示 1 个数据。
    - `0x00`：起始寄存器地址：查表可知，`0x00` 寄存器表示电机模式设置。
    - `0x0A`：往 `0x00` 寄存器写入 `0x0A`。
2. 子帧 2：整体意思是位置不限制，速度为 0.1 转/秒
    - `0x08`：第 2 个子帧的开头
        - 高 8 位为 `0x0`，表示写操作。
        - 低 8 位：
            - 高 2 位为：`10`，表示 `int32_t` 类型。
            - 低 2 位为：`00`，表示模式二，后一个字节表示数据数量。
    - `0x02`：2 个数据。
    - `0x20`：起始寄存器地址：查表可知，`0x20` 寄存器表示位置，`0x21` 寄存器表示速度。
    - `0x00、0x00、0x00、0x80`：小端模式，即 `0x80000000` 写入 `0x20` 寄存器，即表示电机位置无限制。
    - `0x10、0x27、0x00、0x00`：小端模式，即 `0x2710` 写入 `0x21` 寄存器，表示电机速度设置为 0.1 转/秒。
3. `0x50，0x50`：由于 CAN-FD 发送字节数最近的两个是 12 和 16，固还需加上 2 个字节的占位字节。

II 、接收协议示例

- **接受协议模式是由发送协议的模式决定的。**
- **接收协议模式和发送协议模式是相同的。**

① 模式一

```
uint8_t rdata[] = {0x28, 0x04, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D, 0x04,
0x00, 0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};
```

该帧由一个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。
    - `0x28`：
        - 高 8 位为 `0010`，表示回复操作。
        - 低 8 位：
            - 高 2 位为：`10`，表示 `int32_t` 类型。
            - 低 2 位为：`00`，表示模式二，后一个字节表示数据数量。

- `0x04`：4 个数据。
- `0x00`：起始寄存器地址
- `0x0A，0x00，0x00，0x00`：`0x00` 寄存器的值，查表可知为电机模式，十进制为 10，查表可知为位置模式。
- `0x96，0x1D，0x04，0x00`：小端模式，十进制为 269718，即电机当前位置是 2.69718 转处。
- `0x54，0x40，0x02，0x00`：小端模式，十进制为 147540，即当前电机速度为 1.4754 转/秒。
- `0x86，0x01，0x00，0x00`：小端模式，十进制为 390，即当前实际输出力矩为：0.0039 NM。
- `0x50`：占位符。

由前三个字符可知： 此 CAN-FD 是回复 `0x18，0x04，0x00` 的。

② 模式二

```
uint8_t rdata[] = {0x2B, 0x01, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D, 0x04, 0x00,
0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};
```

该帧由一个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。
   - `0x2B`：
     - 高 8 位为 `0x2`，表示回复操作。
     - 低 8 位：
       - 高 2 位为：`10`，表示 `int32_t` 类型。
       - 低 2 位为：`11`，表示 3 个数据。
   - `0x01`：起始寄存器地址
   - `0x0A，0x00，0x00，0x00`：`0x00` 寄存器的值，查表可知为电机模式，十进制为 10，查表可知为位置模式。
   - `0x96，0x1D，0x04，0x00`：小端模式，十进制为 269718，即电机当前位置是 2.69718 转处。
   - `0x54，0x40，0x02，0x00`：小端模式，十进制为 147540，即当前电机速度为 1.4754 转/秒。
   - `0x86，0x01，0x00，0x00`：小端模式，十进制为 390，即当前实际输出力矩为：0.0039 NM。
   - `0x50`：占位符。

由前三个字符可知： 此 CAN-FD 是回复 `0x1B，0x01` 的。

Ⅲ 、帧解析示例

单个 CAN-FD 帧可用于命令伺服，并启动对某些寄存器的查询。示例框架如下所示，以十六进制编码并带有注释。

例如整个 CAN-FD 消息将是(十六进制)： (单片机发送) **0x01000a07206000200150ff140400130d**

数据	描述
01	写入单个 int8 寄存器（寄存器数量在 2 个 LSB 中编码）
00	起始寄存器号“模式”
0a	“位置”模式
07	写入 3 个 int16 寄存器（寄存器数量在 2 个 LSB 中编码）
20	寄存器 0x020
6000	位置 = 0x0060 = 96 = 3.456 度
2001	速度 = 0x0120 = 288 = 25.92 dps
50ff	前馈扭矩 = 0xff50 = -176 = 1.76 N*m
14	读取 int16 寄存器
04	读取 4 个寄存器
00	从 0x000 开始（所以 0x000 模式，0x001 位置，0x002 速度，0x003 扭矩）
13	读取 3x int8 寄存器
0d	从 0x00d 开始（所以 0x00d 电压，0x00e 温度，0x00f 故障代码）

因此，要使用 **fdcanusb 转换器**将其发送到配置为默认地址 1 的设备，您可以编写。

- `can send` **8001 01000a07206000200150ff140400130d**

这80 在ID 中用于两个目的。设置的高位强制设备响应（否则即使发送查询命令，它也不会响应）。其余位是要响应的“ID”。作为对该命令的响应，来自伺服的可能响应如下所示：

- `rcv` **100 2404000a005000000170ff230d181400**

解码，这意味着：

数据	描述
100	从设备“1”到设备“0”
24	回复 int16 值
04	4个寄存器
00	从寄存器 0 开始
0a00	在模式 10 - 位置
5000	位置是 0x0050 = 80 = 2.88 度
0001	速度为 0x0100 = 256 = 23.04 dps
70ff	扭矩为 0xff70 = -144 = -1.44 Nm
23	回复 3 个 int8 值

数据	描述
0d	从寄存器 0x00d 开始
18	电压为12V
14	温度为20℃
00	没有错

3.1.4 常用类型（单位）说明

3.1.4.1 电流（A）

数据类型	LSB	实际 (A))
`int8`	1	1
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

3.1.4.2 电压（V）

数据类型	LSB	实际 (V))
`int8`	1	0.5
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

3.1.4.3 扭矩（Nm）

- **真实扭矩 = k * tqe + d**

① 5046 扭矩（Nm）

数据类型	斜率 (k)	偏移量 (d)
`int8`	0.225234	-0.457642
`int16`	0.004497	-0.448998
`int32`	0.000439	-0.410969
`float`	0.433600	-0.519366

② 4538 扭矩（Nm）

数据类型	斜率 (k)	偏移量 (d)
int8	0.002024	-0.293455
int16	0.003813	-0.280438
int32	0.000368	-0.225920
float	0.414104	-0.472467

3.1.4.4 温度（℃）

数据类型	LSB	实际 (℃)
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

3.1.4.5 时间（s）

数据类型	LSB	实际 (s)
int8	1	0.01
int16	1	0.001
int32	1	0.000001
float	1	1

3.1.4.6 位置（转）

数据类型	LSB	实际 (转)	实际（°）
int8	1	0.01	3.6
int16	1	0.0001	0.036
int32	1	0.00001	0.0036
float	1	1	360

3.1.4.7 速度（转/秒）

数据类型	LSB	实际 (转/秒)
int8	1	0.01
int16	1	0.0001

数据类型	LSB	实际 (转/秒)
`int32`	1	0.00001
`float`	1	1

3.1.4.8 加速度（转/秒^2)

数据类型	LSB	实际 (转/秒^2)
`int8`	1	0.5
`int16`	1	0.01
`int32`	1	0.001
`float`	1	1

3.1.4.9 PWM 标度（无单位)

数据类型	LSB	实际
`int8`	1	1/127 - 0.007874
`int16`	1	1/32767 - 0.000030519
`int32`	1	(1/2147483647) - 4.657^10
`float`	1	1

3.1.4.10 Kp、Kd 标度（无单位)

数据类型	LSB	实际
`int8`	1	1/127 - 0.007874
`int16`	1	1/32767 - 0.000030519
`int32`	1	(1/2147483647) - 4.657^10
`float`	1	1

3.1.5 函数示例

说明:

- **下面介绍示例程序中提供的电机控制模式的简要说明。**
- **详情请看提供的示例工程。**
- **本电机所能实现的功能不止于此，如需自定义特殊功能，可根据寄存器表发挥想象。**

3.1.5.1 DQ 电压控制

说明：

- D 相电压默认为 0。
- Q 相电压由用户控制。

提供函数：

```
void set_dq_volt_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float
volt);
void set_dq_volt_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t
volt);
void set_dq_volt_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t
volt);
```

3.1.5.2 DQ 电流控制

说明：

- D 相电流默认为 0。
- Q 相电流由用户控制。

提供函数：

```
void set_dq_current_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float
current);
void set_dq_current_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int32_t current);
void set_dq_current_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int16_t current);
```

3.1.5.3 位置控制

说明：

- 以最大速度和力矩转动到指定位置。
- 正负表示方向。

提供函数：

```
void set_pos_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos);
void set_pos_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t
pos);
void set_pos_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t
pos);
```

3.1.5.4 速度控制

说明：

- 以设置速度转动。
- 正负表示方向。

提供函数：

```
void set_val_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float val);
void set_val_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t
val);
void set_val_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t
val);
```

3.1.5.5 力矩控制

说明：

- 以给定的力矩转动。

提供函数：

```
void set_torque_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float
torque);
void set_torque_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t
torque);
void set_torque_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t
torque);
```

3.1.5.6 位置、速度、最大力矩控制

说明：

- 以给定的速度转动到指定位置，并限制输出的最大力矩。

提供函数：

```
void set_pos_vel_tqe_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
float pos, float val, float torque);
void set_pos_vel_tqe_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int32_t pos, int32_t val, int32_t torque);
void set_pos_vel_tqe_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int16_t pos, int16_t val, int16_t torque);
```

3.1.5.7 位置、速度、力矩、PD 控制

说明：

- 以给定的速度转动到指定位置，并限制输出的最大力矩。
- 并可调节内部 Kp、Kd 的比例。

提供函数：

```
void set_pos_val_tqe_pd_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
float pos, float val, float tqe, float kp, float kd);
void set_pos_val_tqe_pd_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int32_t pos, int32_t val, int32_t tqe, float rkp, float rkd);
void set_pos_val_tqe_pd_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int16_t pos, int16_t val, int16_t tqe, float rkp, float rkd);
```

3.1.5.8 速度、速度限幅控制

说明：

- 以指定速度转动、若速度大于速度限幅，则以速度限幅转动。

提供函数：

```
void set_val_valmax_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int16_t val, int16_t vel_max);
```

3.1.5.9 位置、速度、加速度控制（梯形控制）

说明：

- 指定电机转动到某个位置，并限制转动过程中的最大速度和加速度。

提供函数：

```
void set_pos_valmax_acc_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
float pos, float vel_max, float acc);
void set_pos_valmax_acc_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int32_t pos, int32_t vel_max, int32_t acc);
void set_pos_valmax_acc_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor,
int16_t pos, int16_t vel_max, int16_t acc);
```

3.1.5.10 速度、加速度控制

说明：

- 以指定的加速度加速到指定速度。

提供函数：

```
void set_val_acc_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float
val, float acc);
void set_val_acc_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t
val, int32_t acc);
void set_val_acc_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t
val, int16_t acc);
```

3.1.5.11 重设零点

说明：

- 将当前位置设为零点。

提供函数：

```
void set_pos_rezero(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
```

3.1.5.12 保存设置

说明：

- 保存电机配置信息。
- 目前只用在重设零点。

提供函数：

```
void set_conf_write(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
```

3.1.5.13 电机刹车

说明：

- 电机所有相都短接到地，产生被动的“刹车”效果。

提供函数：

```
void set_motor_brake(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
```

3.1.5.14 电机停止

说明：

- 进入停止状态。

提供函数：

```
void set_motor_stop(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
```

3.1.5.15 读取电机状态

说明：

- 提供了读取电机运行状态、位置、速度、转矩四种数据的例程。
- 这里的函数是让电机发送这四种数据回来，具体解析数据的例程请看示例代码。

提供函数：

```
void read_motor_state_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
void read_motor_state_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
void read_motor_state_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
```

3.1.6 示例程序说明

- **电机示例函数都在 libelybot. c 和 libelybot. h 文件内。**
- **示例程序默认不会运行任何控制代码，需测试某一功能请在 main. c 文件内解注释或自行编写。**

需要注意的宏定义：

1. `MOTOR_MODEL`：用于选择电机型号，用于修正输入力矩。
    1. `MOTOR_MODEL = 5046`：5046 电机。
    2. `MOTOR_MODEL = 4538`：4538 电机。
    3. `MOTOR_MODEL = 50471`：5047 单极电机。

    4. `MOTOR_MODEL = 50472`：5047双极电机。
2. `POS_FLAG`：用于**测试位置模式**，共三种模式，三种模式只是数据类型不同，效果都是 -0.5 转~0.5 转来回旋转。
    1. `POS_FLAG = 1`：float
    2. `POS_FLAG = 2`：int32
    3. `POS_FLAG = 3`：int16
3. `READ_MOTOR_FLAG`：用于改变读取**电机状态的数据类型。**
    1. `READ_MOTOR_FLAG = 1`：float
    2. `READ_MOTOR_FLAG = 2`：int 32
    3. `READ_MOTOR_FLAG = 3`：int 16
4. `POS_REZERO`：用于测试电机的**重置零点**功能，
    1. 效果是电机上电 3 秒后将当前位置设为零点。
    2. 注意：需让电机停止后再重置零位，否则无效
    3. 需测试此功能将 POS_REZERO 解注释即可。
5. `MOTOR_STOP`：用于测试电机**停止**功能。
    1. 需测试此功能将 MOTOR_STOP 解注释即可。
    2. 效果是电机上电 3 秒后将停止。
    3. 注意：需配合电机控制函数使用，启用 MOTOR_STOP 宏不会改变任何电机控制函数。
6. `MOTOR_BRAKE`：用于测试电机**刹车**功能功能。
    1. 需测试此功能将 MOTOR_BRAKE 解注释即可。

3.1.7 寄存器功能说明

寄存器地址	寄存器名称	r/w	寄存器说明
0x000	模式	R/W	电机运行模式（具体模式见 3.2.8电机运行模式）
0x001	位置	R	电机输出轴位置
0x002	速度	R	电机输出轴速度
0x003	转矩	R	电机的输出轴转矩
0x004	Q 相电流	R	Q 相电流
0x005	D 相电流	R	D 相电流
0x006	保留		保留
0x00d	电压	R	输入电压
0x00e	温度	R	温度
0x00f	错误代码	R	具体错误代码见表3报错代码说明
0x010	PWM相位A	R/W	PWM 模式下，控制 A 相的原始 PWM 值
0x011	PWM相位B	R/W	PWM 模式下，控制 B 相的原始 PWM 值
0x012	PWM相位C	R/W	PWM 模式下，控制 C 相的原始 PWM 值

寄存器地址	寄存器名称	r/w	寄存器说明
0x014	电压相位A	R/W	电压模式下，控制施加到 A 相的电压
0x015	电压相位B	R/W	电压模式下，控制施加到 B 相的电压
0x016	电压相位C	R/W	电压模式下，控制施加到 C 相的电压
0x018	电压 FOC 角度	R/W	电压模式下，控制所需的电角度（没有乘以极对数）
0x019	电压 FOC 电压	R/W	电压聚焦模式下，控制所需的施加相电压
0x01a	D 电压	R/W	DQ 电压模式下，控制 D 相电压
0x01b	Q 电压	R/W	DQ 电压模式下，控制 Q 相电压
0x01c	Q 电流	R/W	DQ 电流模式下，控制 Q 相电流
0x01d	D 电流	R/W	DQ 电流模式下，控制 D 相电流
0x020	位置指令	R/W	位置模式下，控制位置
0x021	速度命令	R/W	位置模式下，控制速度
0x022	前馈扭矩	R/W	位置模式下，控制前馈转矩
0x023	Kp 比例	R/W	位置模式下，将比例控制项缩小给定因子
0x024	Kd 比例	R/W	位置模式下，将微分控制项缩小给定因子
0x025	最大扭矩	R/W	位置模式下，控制的最大扭矩
0x026	停止位置	R/W	位置模式下，并且命令非零速度时，到达给定位置时停止运动
0x027	保留		保留
0x028	速度限制	R/W	全局速度限制
0x029	加速度限制	R/W	全局加速度限制
0x030	比例扭矩	R	PID 控制器中比例项的转矩
0x031	积分扭矩	R	PID 控制器中积分项的转矩
0x032	微分扭矩	R	PID 控制器中微分项的转矩
0x033	前馈扭矩	R	PID 控制器中的前馈
0x034	总控制扭矩	R	位置模式下，总控制扭矩
0x040	下限	R/W	范围模式下，它控制最小允许位置
0x041	上限	R/W	范围模式下，它控制最大允许位置
0x042	前馈扭矩		0x022 寄存器的映射

寄存器地址	寄存器名称	r/w	寄存器说明
0x043	Kp 比例		0x023 寄存器的映射
0x044	Kd 比例		0x024 寄存器的映射
0x045	最大扭矩		0x025 寄存器的映射

3.1.8 电机运行模式

模式序号	模式名称
0	停止，清除错误
1	错误
2，3，4	准备运行
5	PWM 模式
6	电压模式
7	foc电压模式
8	DQ电压模式
9	DQ电流模式
10	位置模式
11	超时模式
12	零速模式
13	范围模式
14	测量电感模式
15	刹车模式
16	保留
17	保留

3.2 CAN协议解析

3.2.1 CAN 相关说明

1. CAN 波特率：
    - 仲裁段：1 Mbps
    - 数据段：1 Mbps。
2. ID：由 16 位构成，其中 `0x7F` 是广播地址。

- 高 8 位：表示**源地址**：
  - 最高位为 1 ：需要回复。
  - 最高位位 0：无需回复。
  - 其余 7 位：信号源地址。
- 低 8 位：表示**目的地址**：
  - 最高为 0。
  - 其余 7 位表示目的地址。

例如：

1. `ID：0x8001`
   - 信号源地址为 0。
   - 目的地址为 1。
   - 最高位为 1，表示需要回复。
2. `ID：0x100`
   - 信号源地址为 1。
   - 目的地址为 0。
   - 最高位为 0，表示无需回复。

3.2.2 模式说明

3.2.2.1普通模式 (位置和速度不能同时控制)

```
uint8_t cmd[] = {0x07, 0x07, pos1, pos2, val1, val2, tqe1,tqe2};
```

- 普通协议由：指令位（2 字节）+位置（2 字节）+速度（2 字节）+力矩（2 字节）共 8 字节构成。
- `0x07 0x07`：普通模式，可控制速度和力矩、位置和力矩（见3.1.5.1 普通模式）。
- 协议中**位置、速度、力矩**数据都为**小端模式**，即低字节先发，高字节后发，
  - 如 `pos = 0x1234` 中，`pos1 = 0x34`，`pos2 = 0x12`。
- 此模式可分为**两种**控制方式：
  - 位置、力矩控制（此时 `val=0x8000`，表示无限制）。
  - 速度、力矩控制 （此时 `pos=0x8000`，表示无限制）。

3.2.2.2 力矩模式

```
uint8_t cmd[] = {0x05, 0x13, tqe1, tqe2};
```

- 力矩模式协议由：指令位（2 字节）+力矩 （2 字节）。
- `0x05 0x13`：纯力矩模式，后面接两字节的力矩数据。(见 3.1.5.2 力矩模式)。
- 协议中力矩数据为小端模式，即低字节先发，高字节后发。
  - 如 `tqe = 0x1234` 中，`tqe1 = 0x34`，`tqe2 = 0x12`。

3.2.2.3 协同控制模式（位置、速度、力矩可以同时控制）

```
uint8_t cmd[] = {0x07, 0x35, val1, val2, tqe1, teq2, pos1,pos2};
```

- 协同控制模式协议：指令位（2 字节）+速度（2 字节）+力矩（2 字节）+位置（2 字节）共 8 字节构成。

- `0x07 0x35`：协同控制模式，已指定速度转动到指定位置，并限制最大力矩。
- 此模式中给如参数 `0x8000` 表示**无限制**（无限制的速度和力矩即为最大值）。
  - 如 `val = 5000, tqe = 1000, pos = 0x8000`：表示电机以 0.5 转/秒的转速一直转动，最大力矩为 0.1NM。
- 协议中位置、速度、力矩数据都为小端模式，即低字节先发，高字节后发，
  - 如 `pos = 0x1234` 中，`pos1 = 0x34`，`pos2 = 0x12`。

3.2.3 电机状态数据读取

1. 读取电机状态部分的协议和 CAN-FD 中的协议是一样的，唯一的区别是 CAN 受到 8 字节数据段的限制。
2. 寄存器地址和功能说明请查看**寄存器功能、电机运行模式、报错代码说明. xlsx**文件。
3. 由于 CAN 受 8 字节数据段限制，一帧 CAN 最多返回的电机信息有限：
   1. 一个寄存器的 `float` 类型或 `int32_t` 的电机信息。
   2. 3 个地址连续的 `int16_t` 类型电机信息。
   3. 6 个地址连续的 `int8_t` 类型的电机信息。
4. 例程中提供了 `int16_t` 的查询电机位置、速度、力矩信息的示例函数和电机信息解析（例程中使用的是 C 语言的共用体直接复制了 CAN 中第 3 到第 8 字节的数据）。

3.2.3.1 发送协议说明

```
uint8_t tdata[] = {cmd, addr, cmd1, addr1, cmd2, add2};
```

大致含义为： 从 `addr` 读取 `cmd[0, 1]` 个 `cmd[3, 2]` 类型的数据。

- `cmd`：
  - 高四位 `[7, 4]`：`0001` 表示读取。
  - 2~3 位 `[3, 2]`：表示类型。
    - `00`：`int8_t` 类型。
    - `01`：`int16_t` 类型。
    - `10`：`int32_t` 类型。
    - `11`：`float` 类型。
  - 低 2 位 `[1, 0]`：表示数量。
    - `01`：一个数据。
    - `10`：两个数据。
    - `11`：三个数据。
- `addr`：开始获取的地址。
  可以将多个 `cmd , addr` 拼接在一起，一次性读取地址不连续和不同类型的数据。

3.2.3.2 接受协议说明

假设获取的数据是 `uint16_t`

```
uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ..., cmd1,addr1, c1, c2, c3, c4}
```

- `cmd`：
- 高四位 `[7, 4]`：`0010` 表示回复。

    - 2~3 位 `[3, 2]`：表示类型。
    - `00`：`int8_t` 类型。
        - `01`：`int16_t` 类型。
    - `10`：`int32_t` 类型。
        - `11`：`float` 类型。
- 低 2 位 `[1, 0]`：表示数量。
    - `01`：一个数据。
    - `10`：两个数据。
        - `11`：三个数据。
- `addr`：开始获取的地址。
- `a1, a2`：数据 1，小端模式。
- `b1, b2`：数据 2，小端模式。

3.2.3.3 示例

1. 我们需要读取位置、速度和扭矩数据。
2. 从寄存器 excel 表中可知：位置、速度和转矩的数据地址分别为：01，02，03。
3. 由此可知，我们可以从地址 01 开始连读 3 个数据，考虑到 CAN 一次最大传输 8 字节的数据，而 `cmd + addr` 占两个字节，所以数据类型最多可以选择 `int16_t` 类型。
4. 由上可知 `cmd` 的二进制为：`0001 1011`，十六进制为：`0x17`。
5. 需要从地址 01 开始读取，故 `addr` 为 `0x01`。
6. 需要发送的总数据为 `uint8_t tdata[] = {0x17, 0x01}`。

示例代码如下：

```
/**
* @brief 读取电机
* @param id
*/
void motor_read(uint8_t id)
{
 static uint8_t tdata[8] = {0x17, 0x01};
 CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}
```

```
uint8_t cmd[] = {0x17, 0x01};
```

整体含义是：从地址 `0x01` 处开始，读取 3 个 `int16_t` 的寄存器（查表可知，地址 `0x01~0x03` 的寄存器分别表示位置、速度和力矩），故此命令为查询电机的位置、速度、力矩信息。

- `0x17`：
    - `0x17[7:4]` 的二进制为 `0001`：表示读。
    - `0x17[3:2]` 的二进制为 `01`：表示数据类型为 `int16_t`。
    - `0x17[1:0]` 的二进制为 `11`：表示数据个数为 3。
- `0x01`：
    - 从 `0x01` 地址开始。

相应接受数据示例：

```
uint8_t rdata[] = {0x27, 0x01, 0x38, 0xf6, 0x09, 0x00, 0x00,0x00};
```

- `0x27`：对应发送的 `0x17`。
- `0x01`：从地址 `0x01` 开始。
- `0x38 0xf6`：位置数据：`0xf638`，即 `-2505`。
- `0x09 0x00`：速度数据：`0x0009`，即 `9`。
- `0x00 0x00`：力矩数据：`0x0000`，即 `0`。

3.2.4 电机停止

说明：

1. 使电机停止。
2. 对应上位机指令 `d stop`

```
/**
 * @brief 电机停止
 */
void motor_stop(uint8_t id)
{
    uint8_t tdata[] = {0x01, 0x00, 0x00};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}
```

3.2.5 示例函数

3.2.5.1 普通模式

1. 位置控制

```
/**
* @brief 位置控制
* @param id 电机ID
* @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
* @param 力矩
*/
void motor_control_Pos(uint8_t id,int32_t pos,int16_t tqe)
{
uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00,
0x00, 0x80, 0x00};
*(int16_t *)&tdata[2] = pos;
 *(int16_t *)&tdata[6] = tqe;
uint32_t ext_id = (0x8000 | id);
CAN_Send_Msg(ext_id, tdata, 8);
}
```

2. 速度控制

```
/**
* @brief 位置控制
* @param id 电机ID
* @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
```

```
* @param 力矩
*/
void motor_control_Pos(uint8_t id,int32_t pos,int16_t tqe)
{
uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00,
0x00, 0x80, 0x00};
*(int16_t *)&tdata[2] = pos;
 *(int16_t *)&tdata[6] = tqe;
uint32_t ext_id = (0x8000 | id);
CAN_Send_Msg(ext_id, tdata, 8);
}
```

3.2.5.2 力矩模式

```
/**
* @brief 力矩模式
* @param id 电机ID
* @param tqe 力矩
*/
void motor_control_tqe(uint8_t id,int32_t tqe)
{
uint8_t tdata[8] = {0x05, 0x13, 0x00, 0x80, 0x20,
0x00, 0x80, 0x00};
 *(int16_t *)&tdata[2] = tqe;
CAN_Send_Msg(0x8000 | id, tdata, 4);
}
```

3.2.5.3 协同控制模式

```
/**
* @brief 电机位置-速度-前馈力矩(最大力矩)控制，int16型
* @param id 电机ID
* @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
* @param val 速度：单位 0.00025 转/秒，如 val = 1000 表示 0.25转/秒
* @param tqe 最大力矩
*/
void motor_control_pos_val_tqe(uint8_t id, int16_t pos,
int16_t val, int16_t tqe)
{
 static uint8_t tdata[8] = {0x07, 0x35, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00};
 *(int16_t *)&tdata[2] = val;
 *(int16_t *)&tdata[4] = tqe;
 *(int16_t *)&tdata[6] = pos;
 CAN_Send_Msg(0x8000 | id, tdata, 8);
}
```

3.2.6 STM32H730 完整示例代码

libelybot_can. h

```
#ifndef _LIBELYBOT_CAN_H
#define _LIBELYBOT_CAN_H
```

```
#include "main.h"

/* NAN 表示不限制 */
#define  INI8_NAN   0x80
#define  INT16_NAN  0x8000
#define  INT32_NAN  0x80000000


typedef struct
{
    uint32_t id;
    int16_t position;
    int16_t velocity;
    int16_t torque;
}motor_state_s;

typedef struct
{
    union
    {
        motor_state_s motor;
        uint8_t data[24];
    };
}motor_state_t;


extern motor_state_t motor_state;
extern uint8_t motor_read_flag;


uint8_t CAN_Send_Msg(uint32_t id, uint8_t *msg, uint8_t len);
void fdcan_filter_init(FDCAN_HandleTypeDef *fdcanHandle);

void motor_control_Pos(uint8_t id, int32_t pos, int16_t tqe);
void motor_control_Vel(uint8_t id, int16_t vel, int16_t tqe);
void motor_control_tqe(uint8_t id, int32_t tqe);
void motor_control_pos_val_tqe(uint8_t id, int16_t pos, int16_t val, int16_t
tqe);


void motor_read(uint8_t id);


#endif
```

libelybot_can. c

```
#include "libelybot_can.h"
#include "fdcan.h"
#include <string.h>


FDCAN_RxHeaderTypeDef fdcan_rx_header1;
```

```c
uint8_t fdcan1_rdata[24] = {0};


motor_state_t motor_state;
uint8_t motor_read_flag = 0;


uint8_t CAN_Send_Msg(uint32_t id, uint8_t *msg, uint8_t len)
{
    FDCAN_TxHeaderTypeDef TxHeader;
    uint8_t TxData[8] = {0};

    TxHeader.Identifier = id;   //设置扩展ID
    TxHeader.IdType = FDCAN_EXTENDED_ID;    //使用扩展ID
    TxHeader.TxFrameType = FDCAN_DATA_FRAME; //数据帧
    TxHeader.DataLength = FDCAN_DLC_BYTES_8; //数据长度
    TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE;// 错误指示状态
    TxHeader.BitRateSwitch = FDCAN_BRS_OFF; //比特率切换关闭，不适用于经典CAN
    TxHeader.FDFormat = FDCAN_CLASSIC_CAN; //经典CAN格式
    TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS;// 不适用发送事件FIFO
    TxHeader.MessageMarker = 0; //消息标记


    //复制数据到发送缓冲区
    for(int i = 0; i < len; i++)
    {
        TxData[i] = msg[i];
    }

    // 发送CAN指令
    if(HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxHeader, TxData) != HAL_OK)
    {
        // 发送失败处理
        Error_Handler();
        return 1; // 返回非零值以表示发送失败
    }
    return 0; // 发送成功
}


void fdcan_filter_init(FDCAN_HandleTypeDef *fdcanHandle)
{
    if (HAL_FDCAN_ConfigGlobalFilter(fdcanHandle, FDCAN_ACCEPT_IN_RX_FIFO0,
FDCAN_ACCEPT_IN_RX_FIFO0, FDCAN_FILTER_REMOTE, FDCAN_FILTER_REMOTE) != HAL_OK)
    {
        Error_Handler();
    }

    if (HAL_FDCAN_ActivateNotification(fdcanHandle,
FDCAN_IT_RX_FIFO0_NEW_MESSAGE | FDCAN_IT_TX_FIFO_EMPTY, 0) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_FDCAN_ConfigTxDelayCompensation(fdcanHandle, fdcanHandle-
>Init.DataPrescaler * fdcanHandle->Init.DataTimeSeg1, 0);
```

```
    HAL_FDCAN_EnableTxDelayCompensation(fdcanHandle);

    if (HAL_FDCAN_Start(fdcanHandle) != HAL_OK)
    {
        Error_Handler();
    }

    //HAL_FDCAN_Start(fdcanHandle);
}


/**
 * @brief 位置控制
 * @param id  电机ID
 * @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
 * @param 力矩：单位：0.01 NM，如 torque = 110 表示最大力矩为 1.1NM
 */
void motor_control_Pos(uint8_t id, int32_t pos, int16_t tqe)
{
    uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00, 0x00, 0x80, 0x00};

    *(int16_t *)&tdata[2] = pos;
    *(int16_t *)&tdata[6] = tqe;

    uint32_t ext_id = (0x8000 | id);
    CAN_Send_Msg(ext_id, tdata, 8);
}


/**
 * @brief 速度控制
 * @param id 电机ID
 * @param vel 速度：单位 0.00025 转/秒，如 val = 1000 表示 0.25 转/秒
 * @param tqe 力矩：单位：0.01 NM，如 torque = 110 表示最大力矩为 1.1NM
 */
void motor_control_Vel(uint8_t id, int16_t vel, int16_t tqe)
{
    uint8_t tdata[8] = {0x07, 0x07, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};

    *(int16_t *)&tdata[4] = vel;
    *(int16_t *)&tdata[6] = tqe;

    uint32_t ext_id = (0x8000 | id);
    CAN_Send_Msg(ext_id, tdata, 8);
}


/**
 * @brief 力矩模式
 * @param id 电机ID
 * @param tqe 力矩：单位：0.01 NM，如 torque = 110 表示最大力矩为 1.1NM
 */
void motor_control_tqe(uint8_t id, int32_t tqe)
{
    uint8_t tdata[8] = {0x05, 0x13, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};
```

```
    *(int16_t *)&tdata[2] = tqe;

    CAN_Send_Msg(0x8000 | id, tdata, 4);
}


/**
 * @brief 电机位置-速度-前馈力矩(最大力矩)控制，int16型
 * @param id  电机ID
 * @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
 * @param val 速度：单位 0.00025 转/秒，如 val = 1000 表示 0.25 转/秒
 * @param tqe 最大力矩：单位：0.01 NM，如 torque = 110 表示最大力矩为 1.1NM
 */
void motor_control_pos_val_tqe(uint8_t id, int16_t pos, int16_t val, int16_t
tqe)
{
    static uint8_t tdata[8] = {0x07, 0x35, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

    *(int16_t *)&tdata[2] = val;
    *(int16_t *)&tdata[4] = tqe;
    *(int16_t *)&tdata[6] = pos;

    CAN_Send_Msg(0x8000 | id, tdata, 8);
}


/**
 * @brief 读取电机
 * @param id
 */
void motor_read(uint8_t id)
{
    static uint8_t tdata[8] = {0x17, 0x01};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}


static uint8_t Fdcan_Dlc_To_Len(uint32_t dlc)
{
    uint8_t len = 0;
    uint8_t tab_dlc_to_len[] = {12, 16, 20, 24, 32, 48, 64};

    if (dlc <= FDCAN_DLC_BYTES_8)
    {
        len = dlc >> 16;
    }
    else
    {
        len = tab_dlc_to_len[(dlc >> 16) - 9];
    }

    return len;
}
```

```
void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)
 // FDCAN FIFO 0 回调函数
{
    uint8_t len = 0;
    if(hfdcan->Instance == FDCAN1)
    {

        HAL_FDCAN_GetRxMessage(hfdcan, FDCAN_RX_FIFO0, &fdcan_rx_header1,
fdcan1_rdata);
        if (fdcan_rx_header1.DataLength != 0)
        {
            len = Fdcan_Dlc_To_Len(fdcan_rx_header1.DataLength);
            motor_state.motor.id = fdcan_rx_header1.Identifier;  // 获取电机 id
            memcpy(&motor_state.data[4], &fdcan1_rdata[2], len - 2);  // 获取电机
状态数据
            motor_read_flag = 1;
        }
    }
}
```

main. c

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint32_t tick_500ms = 0;
    /* USER CODE END 1 */

    /* MPU Configuration--------------------------------------------------------
*/
    MPU_Config();

    /* MCU Configuration--------------------------------------------------------
*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick.
*/
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_FDCAN1_Init();
```

```
    MX_USART1_UART_Init();
    MX_USART3_UART_Init();
    /* USER CODE BEGIN 2 */

    fdcan_filter_init(&hfdcan1);
    //    if(HAL_FDCAN_Start(&hfdcan1) != HAL_OK)
    //    {
    //        //启动失败管理
    //        Error_Handler();
    //    }
    motor_control_pos_val_tqe(1, INT16_NAN, 1000, 100);
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        if (HAL_GetTick() - tick_500ms >= 2000)
        {
            tick_500ms = HAL_GetTick();
            //motor_control_pos_val_tqe(1, INT16_NAN, 1000, 100);
            HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
            DEBUG_PRINT("AAA");
            motor_read(1);
        }

        if (motor_read_flag == 1)
        {
            motor_read_flag = 0;
            DEBUG_PRINT("motor %x %d  %d %d",motor_state.motor.id,
motor_state.motor.position, motor_state.motor.velocity,
motor_state.motor.torque);
        }
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```