DM-H6215轮毂电机使用教程-V2.0

1 电机基本参数

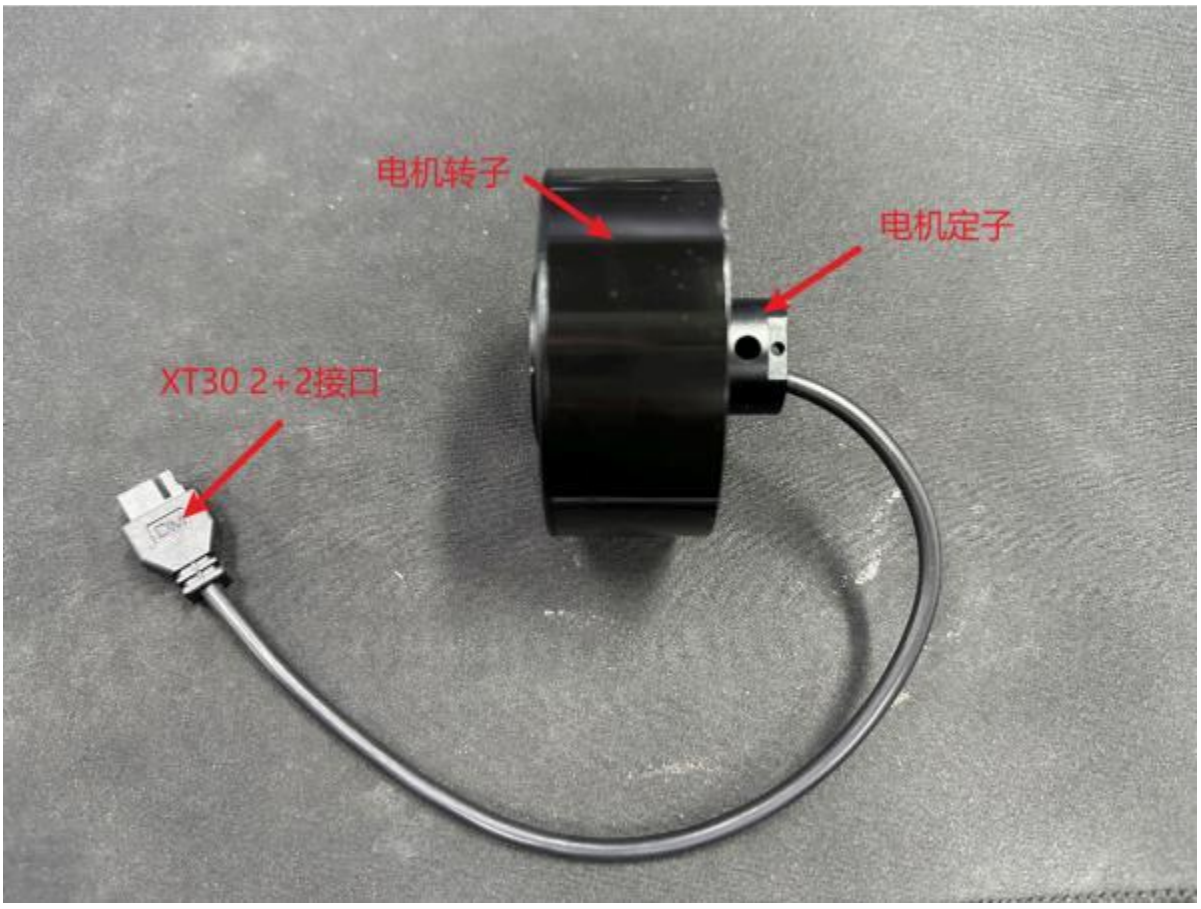
DM-H6215轮毂电机（以下简称6215）是一款一体化设计、电机内部直接出线、直驱、低速、大扭矩的轮毂电机。以下是该电机的基本参数。

电机参数	额定电压	24V
	额定电流	2.5A
	峰值电流	4.3A
	额定扭矩	1NM
	峰值扭矩	2NM
	额定转速	120rpm
	空载最大转速	320rpm
电机特征值	减速比	1：1
	极对数	14
	相电感	2000uh
	相电阻	2.5Ω
结构与重量	外径	68mm
	高度	44.5mm
	电机重量	约 360g
编码器	编码器类型	霍尔
通讯	控制接口	CAN
	调参接口	CAN

请根据以上参数合理使用电机

2 电机外观

DM-H6215轮毂电机为保证结构的紧凑一体化，没有传统达妙电机的串口调试接口，用一根XT302+2接口从内部直接出线。



电机接口与下图**所示方向**保持一致，接线时请注意接口线序。

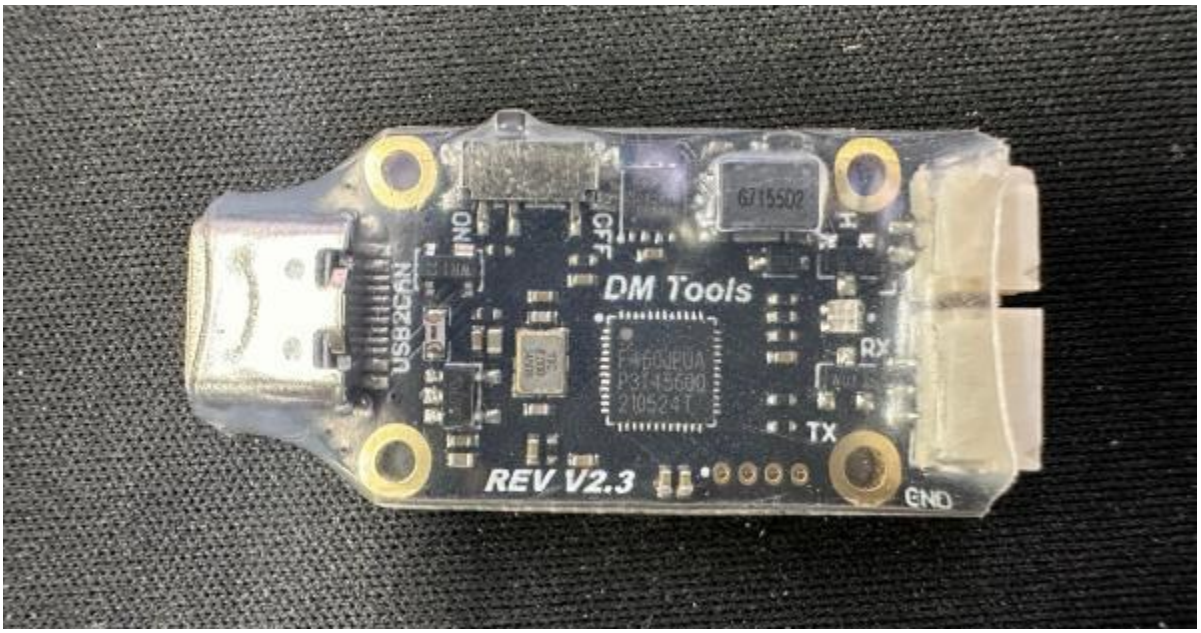
接口及线序说明

具体名称-序号	接口标注	说明
电源连接接口	CAN_L CAN_H	1、通过 XT30(2+2)-F 插头的电源连接线连接电源，额定电压为 24V，为电机供电。 2、通过 CAN 通信端子连接外部控制设备，可接收 CAN 控制命令，反馈电机状态信息。

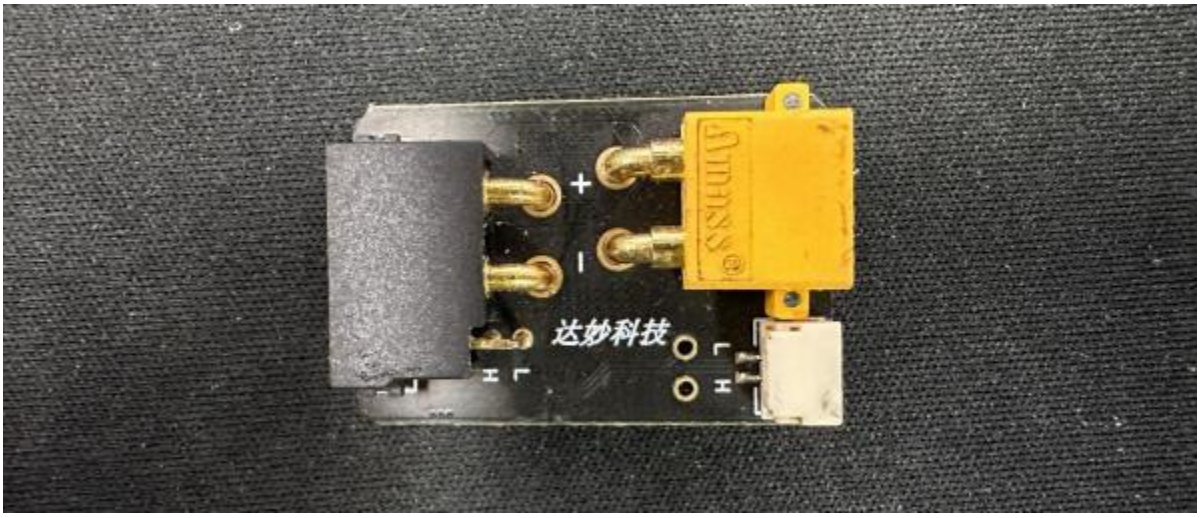
3 电机与调试模块连接

DM-H6215轮毂电机没有**串口调试接口**，无法使用**串口**对电机进行**固件升级**、**校准**、**标定参数**、**修改参数**等功能。统一使用**CAN通信**完成这些功能，**固件升级**需要使用**上位机调试助手**完成， **其他功能**可通过上**位机USB转CAN软件**或下位机直接发送CAN数据包完成。首先我们**使用USB转CAN调试模块**连接电机。

达妙科技USB转CAN调试模块CAN分析仪 MIT电机调试器 USB转CAN模块MIT驱动器 CAN通讯达妙科技-淘宝网 (taobao.com)

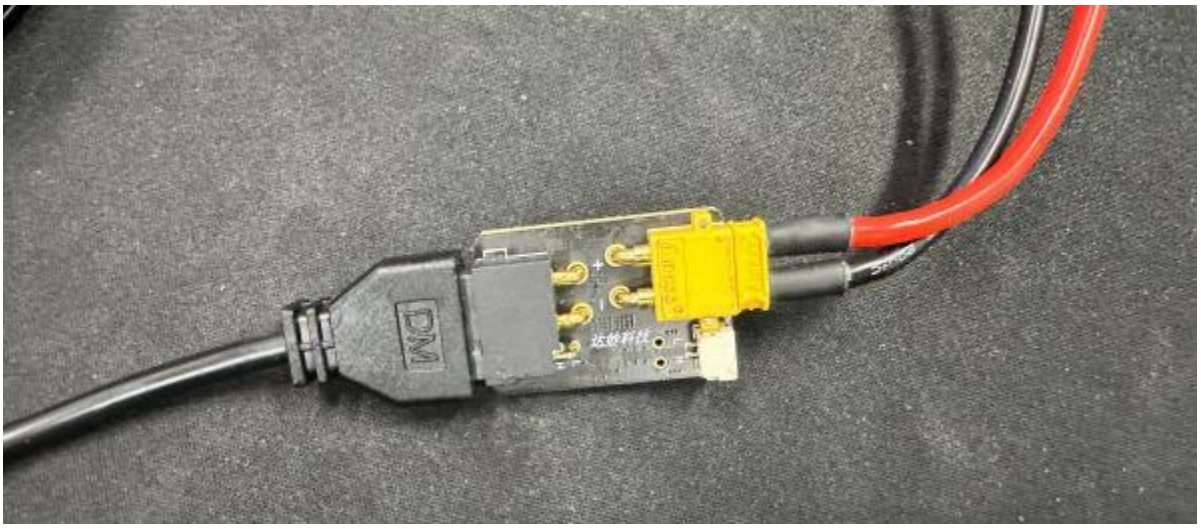


因6215电机使用XT302+2接口，在调试时需将其转接为GH1.25接口与调试助手通信，需使用**XT30 2+2转GH1.25+XT30模块**达妙科技关节电机配线 XT30 2+2 端子线CAN线GH1.25线线材专拍-淘宝网 (taobao.com)

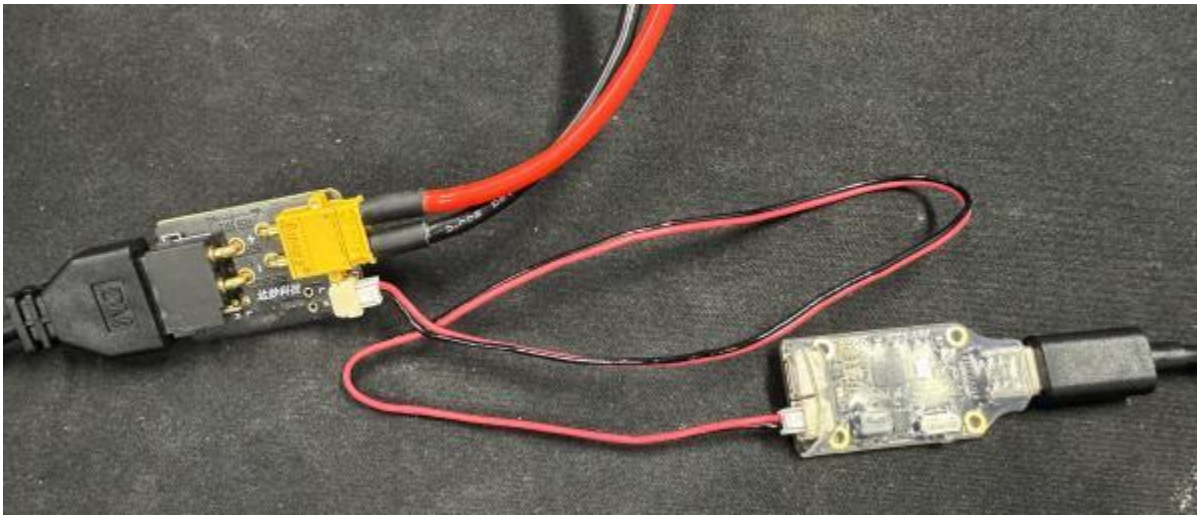


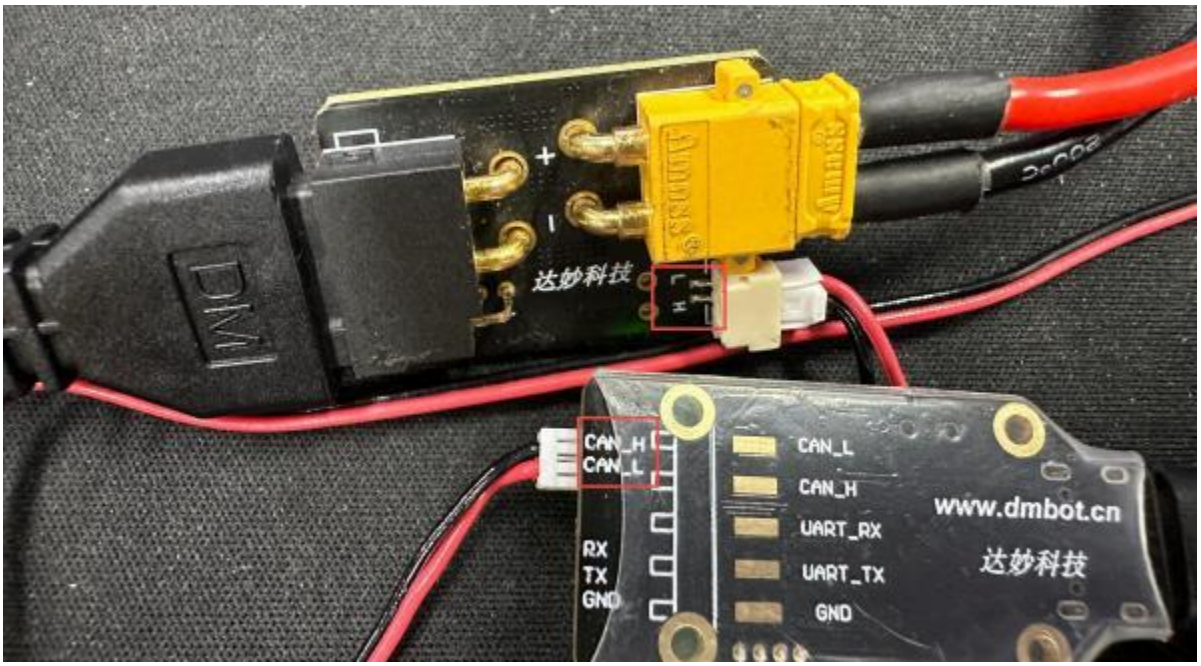
将6215电机XT30 2+2接口插入转接模块中，若使用**达妙科技官方的转接模块**，线序是对应的，若使用其他转接模块或自制模块，注意CAN通信接口线序，与电机CAN_L、CAN_H对应。

使用**XT30母头线**插入转接模块的**XT30公头**对电机**24V**供电（注意正负极）



使用GH1.25-2pin线连接转接模块与调试模块，注意CAN通信接口线序对应，**CAN_L连接CAN_L,CAN_H连接CAN_H**



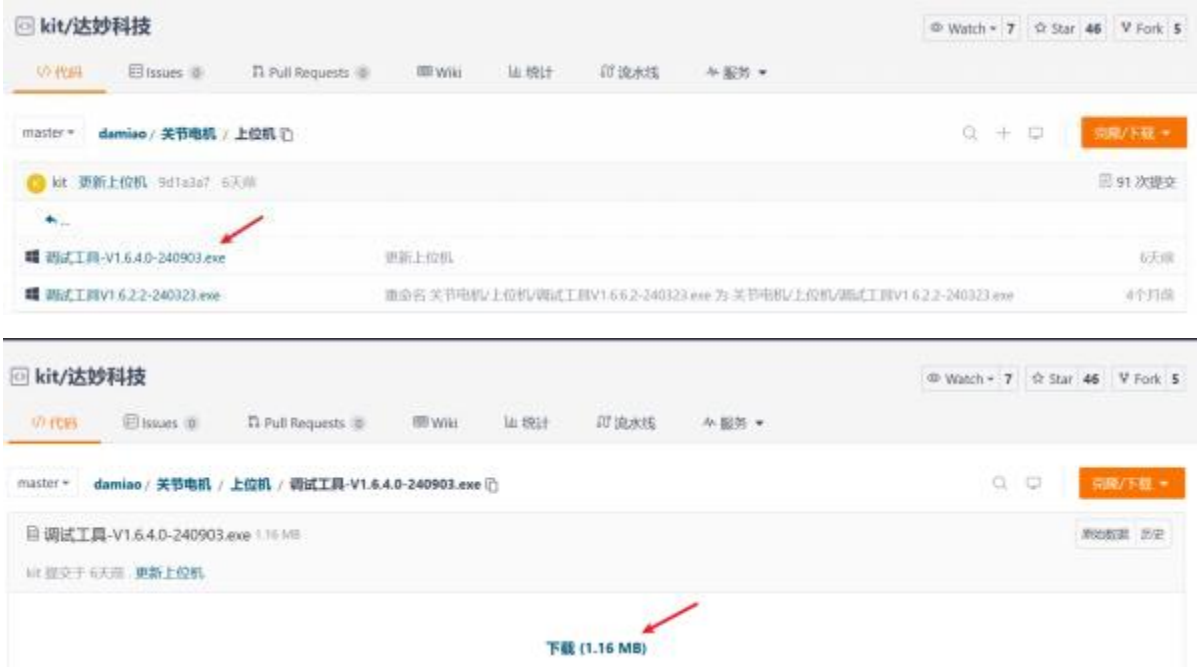


最后使用带有**数据传输功能**的**USB-A转Type-C线**连接**电脑和调试模块。**

4 电机固件升级

通常情况下，轮毂电机不需要升级，出厂即为最新版本。更新程序时，请谨慎！如需升级时，需使用**达妙科技官方调试助手** 下载地址：（https://gitee.com/kit-miao/dm-tools

）**调试工具-V1.6.4.0**



下载完成后，打开调试助手

选择与调试模块连接对应的串口号，若没有检测到串口号可点击刷新串口或检查设备管理器是否识别到串口。

选择对应串口后，点击**打开串口**。

点击**固件升级**旁**绿色的UART**修改成**红色的CAN**

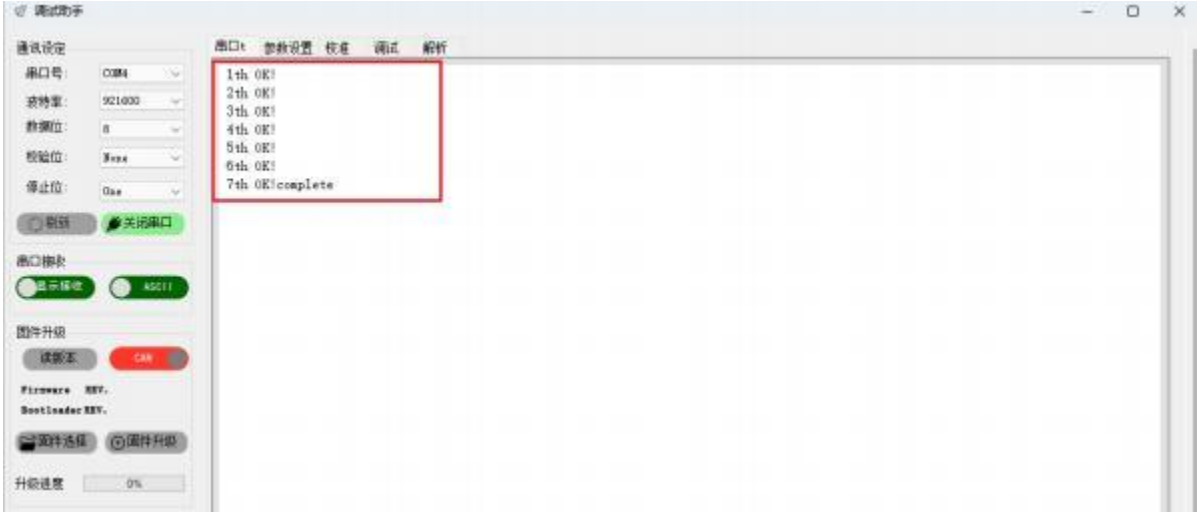
点击**固件选择**

双击选择带有hub字样的固件，当前最新版为5406版本

APP_hub_5406_02.bin	2024/12/27 22:03	BIN 文件	49 KB

选择完成后，点击固件升级

此时上位机会将固件自动下载至电机中，升级进度的进度条会从0到100，串口打印 界面也会显示升级情况



打印**complete**后代表电机固件升级成功

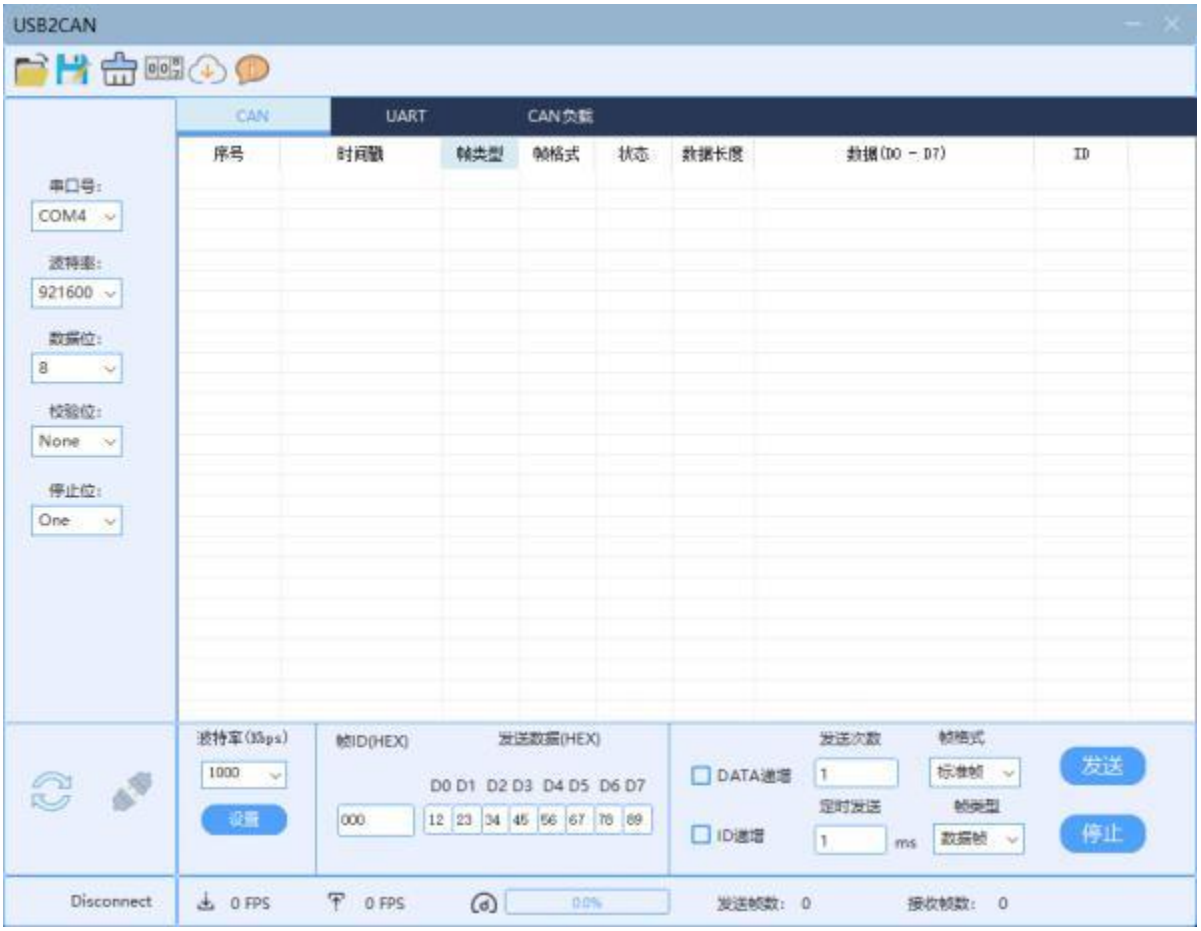
5 CAN通信修改电机参数

根据DM-H6215轮毂电机说明书[DM-H6215轮毂电机说明书V1.0-20240607.pdf (gitee.com)](#)中关于控制协议章节的说明，可使用CAN通信给电机发送指令对电机进行 **校准、标定参数、修改参数**等。

5.1 软件

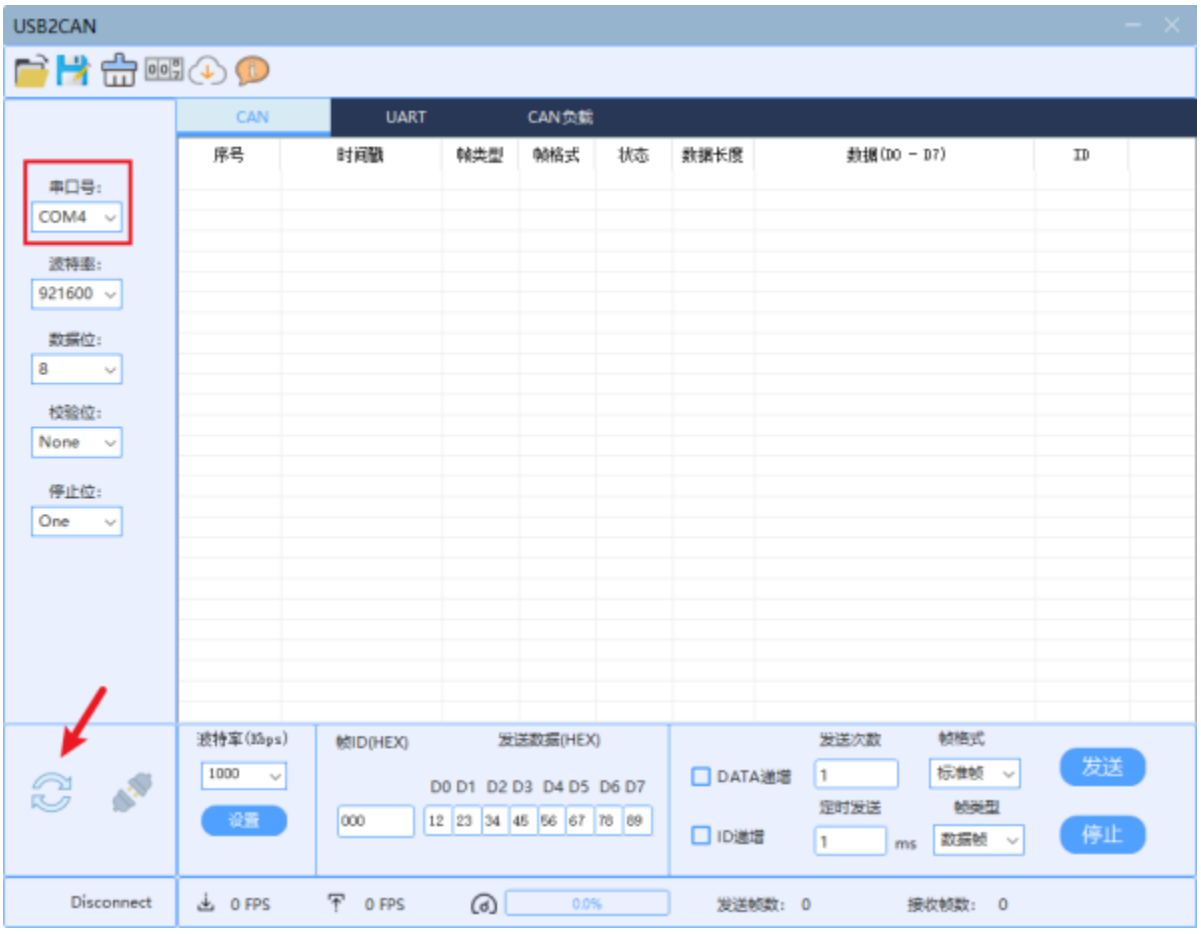
需使用**达妙科技 USB 转 CAN** 开源软件。USB2CAN 上位机 https://gitee.com/kit-miao/dm-tools/tree/master/USB%E8%BD%ACCAN/%E4%B8%8A%E4%BD%8D%E6%9C%BA

打开下载好的**USB转CAN调试软件**

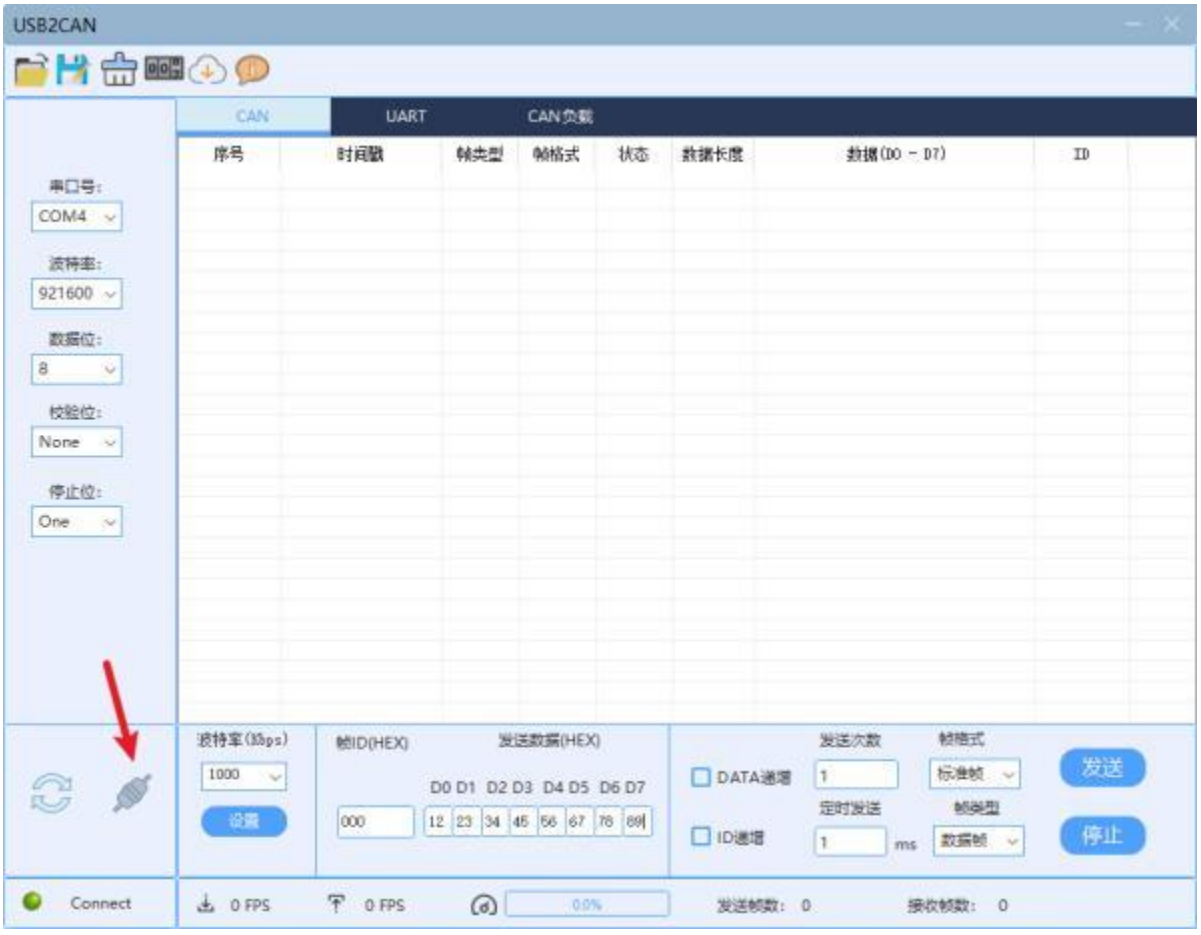


5.2 串口通信设置

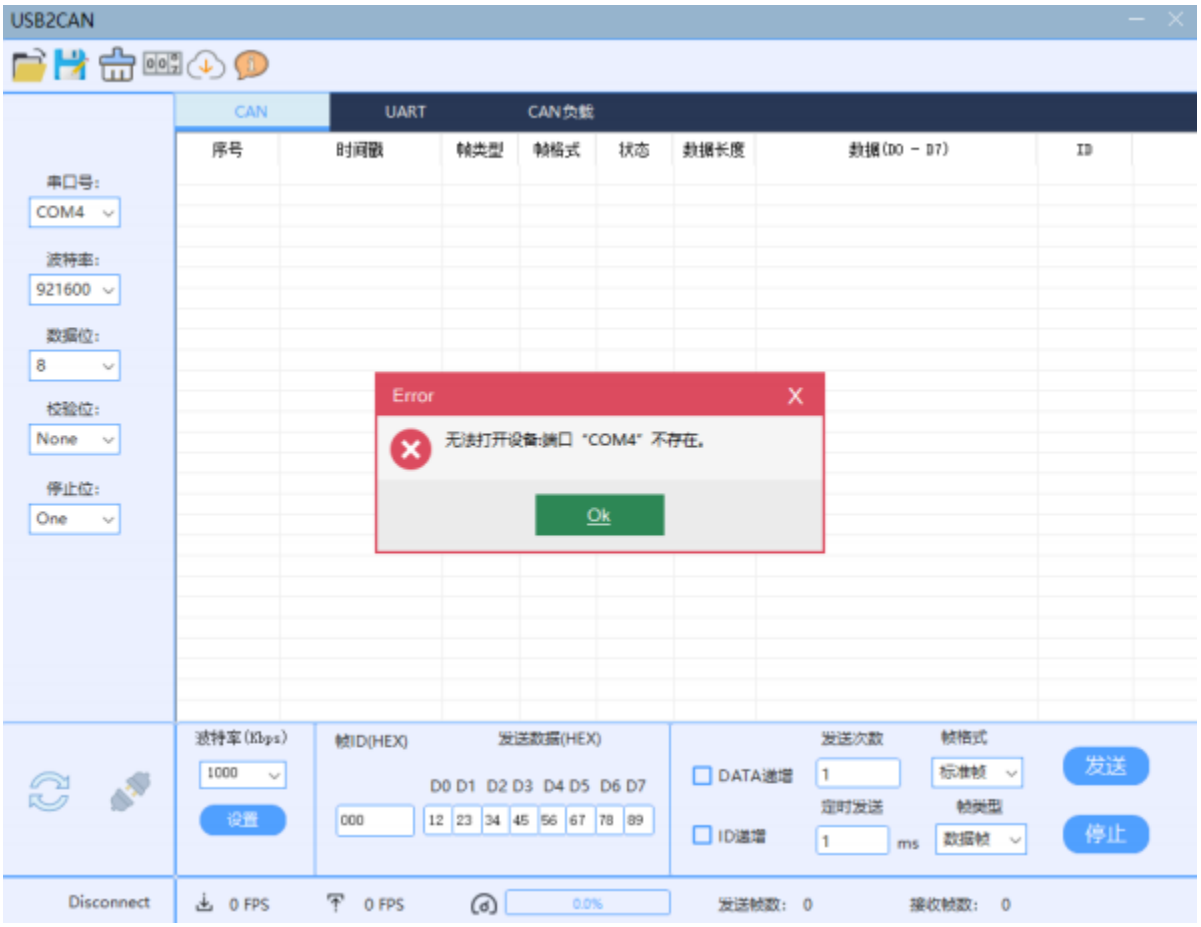
选择与调试模块相连的串口号，若没有找到，点击下面的刷新。**波特率：921600，数据位：8，校验位： None 停止位：One** 保持默认即可。



打开串口，点击下方连接按钮

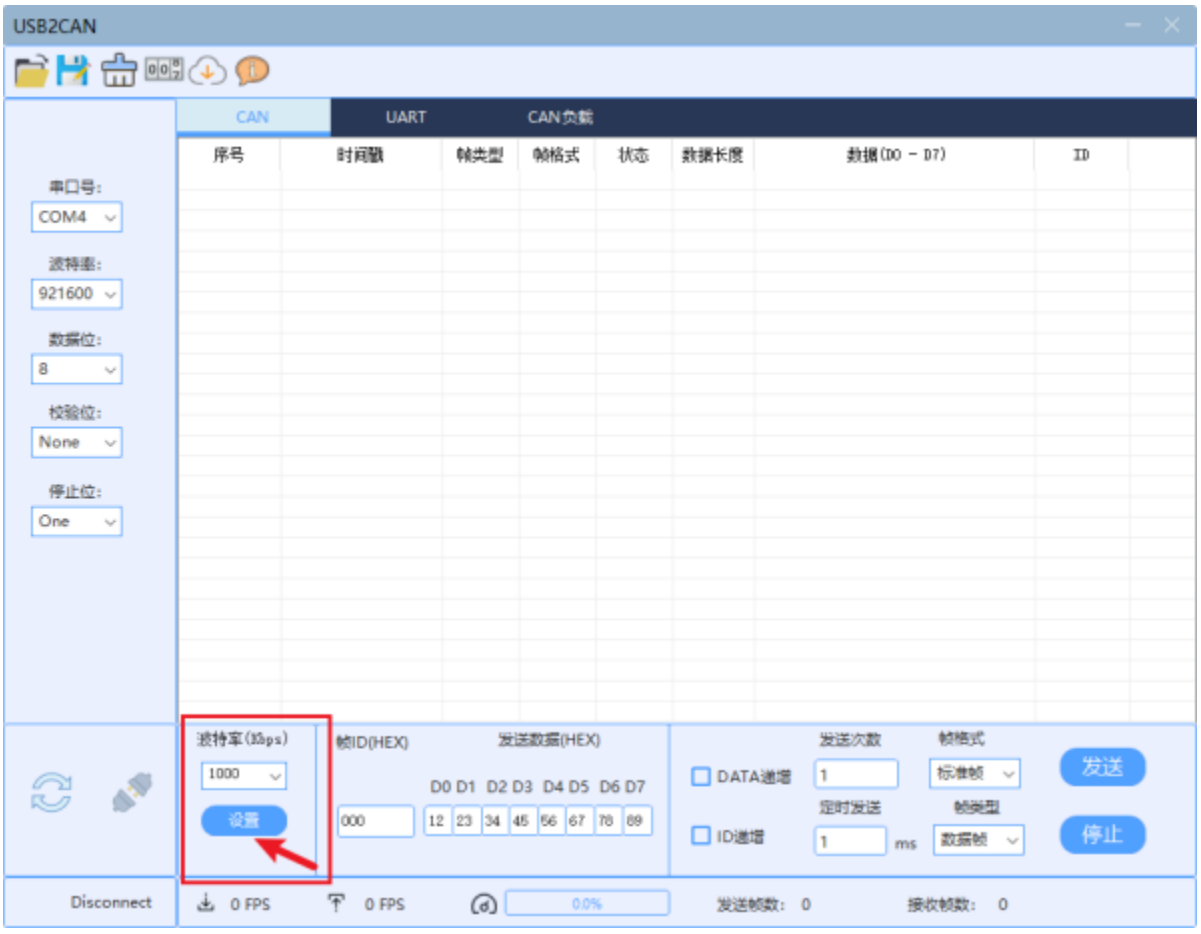


若提示串口不存在，请再次检查**串口号是否正确、串口模块与电脑是否连接、串口驱动是否安装。**

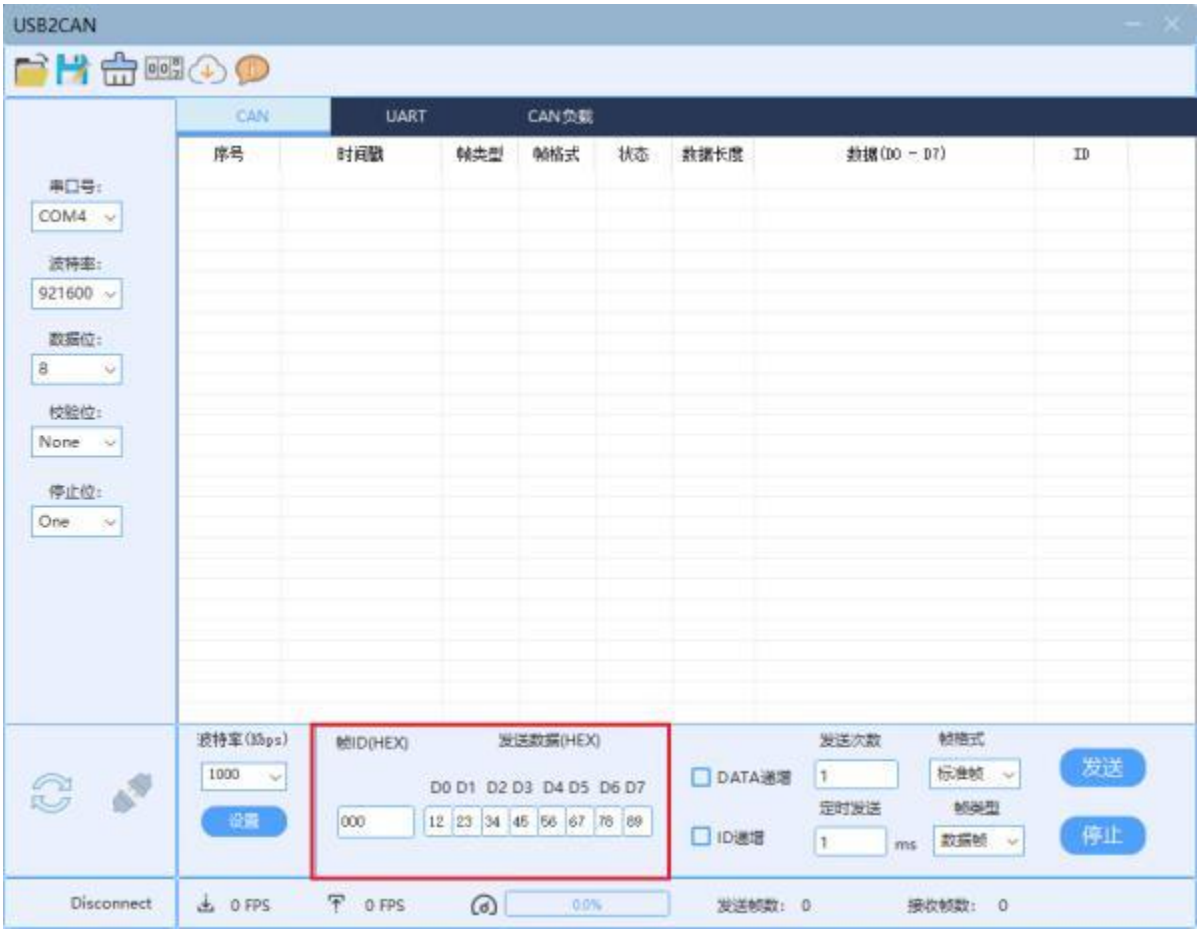


5.3 CAN通信配置

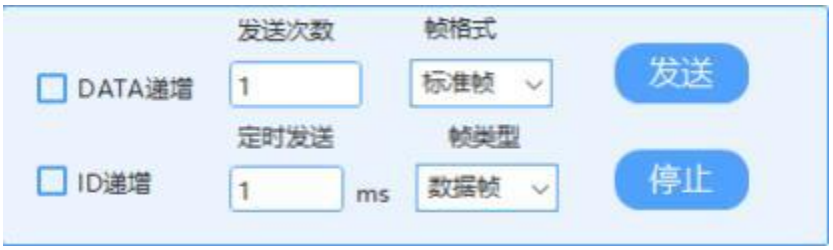
出厂电机CAN通信波特率默认1000Kbps，软件打开默认波特率也为1000Kbps，不用修改。若变更电机波特率后，选择正确的波特率后点击**设置**。



下图红框中为期望发送的数据， **为发送的ID和8字节数据**



发送设置



发送次数：即点击发送后， 一共发送多少次数据给电机

定时发送： **当发送次数大于1时**，每次发送之间间隔的时间。

DATA递增： **当发送次数大于1时**，每次发送的数据会自动+1，例如 发送的数据为 01 00 00 00 00 00 00 00，勾选DATA递增后，发送第二帧数据为 02 00 00 00 00 00 00 00，第三帧数据为 03 00 00 00 00 00 00 00 以此类推。满FF进位。

ID递增：当发送次数大于1时，每次发送的帧ID会自动+1，勾选ID递增后，发送的帧ID为001，发送的第二次帧ID则为002,以此类推满F进位。

帧格式：可选择标准帧和扩展帧，通常使用标准帧

帧类型：可选择数据帧和和远程帧，通常使用数据帧。

暂时将配置为上图即可（默认）

5.4 控制协议详解

5.4.1 电机校准

电机出厂前已经校准过，到手可以直接使用，但当电机编码器出现偏移时，可通过电机校准修复，发送指令格式如下：

❖ **电机校准**

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0x7FF	STD	CANID_L	CANID_H	0x66	0x00	xx(don' t care)			

校准完成后，电机会主动返回校准完成状态帧，其帧格式如下：

报文 ID	属性	D[0]	D[1]	D[2]	D[3]
MST_ID	STD	CANID_L	CANID_H	0x66	0x01

发送给电机校准的指令如上图所示，在电机完成校准后会返回状态帧

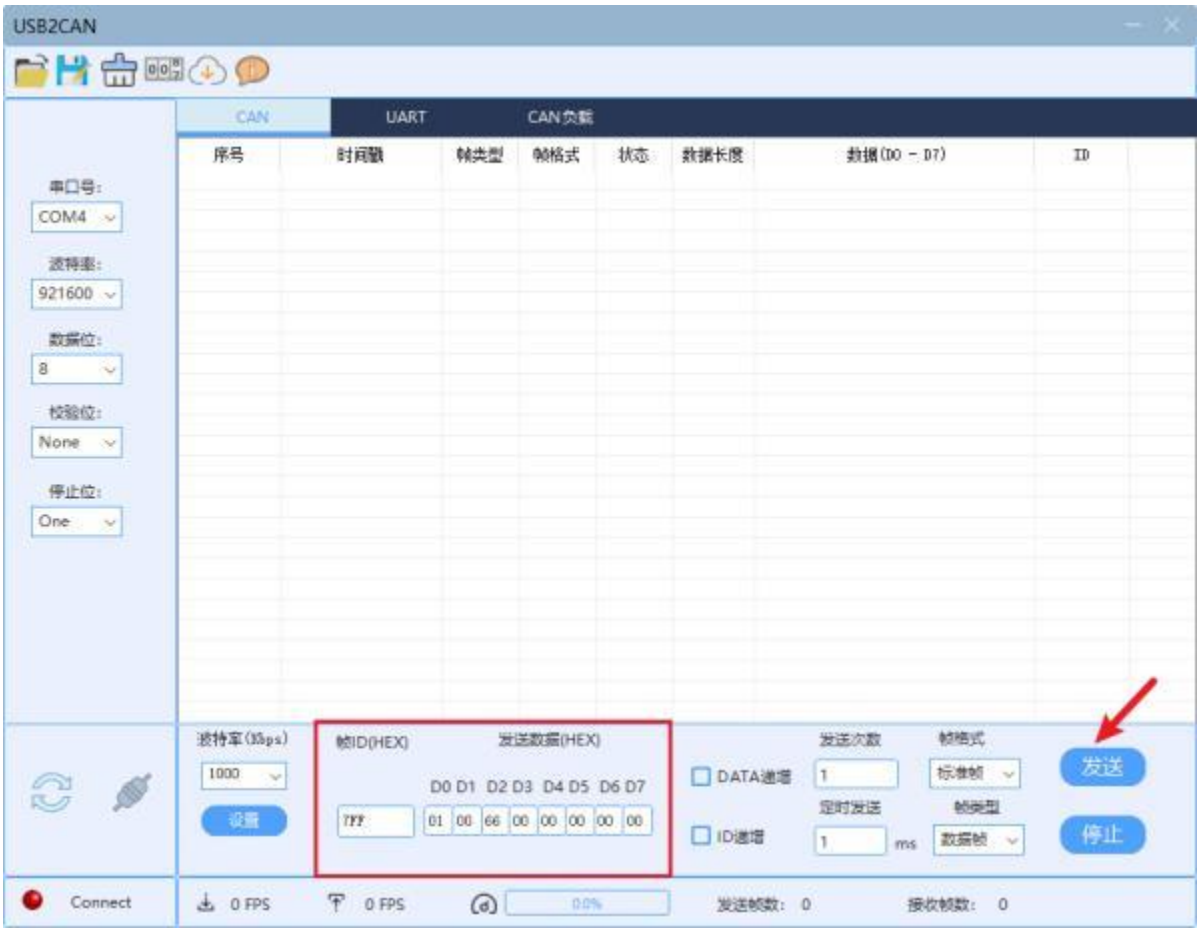
根据协议，将**帧ID修改为0x7FF**

D[0]和D[1]分别为电机CAN_ID的低位和高位，电机出厂默认ID为1,即0x0001,所以D[0] = 01 D[1] = 00

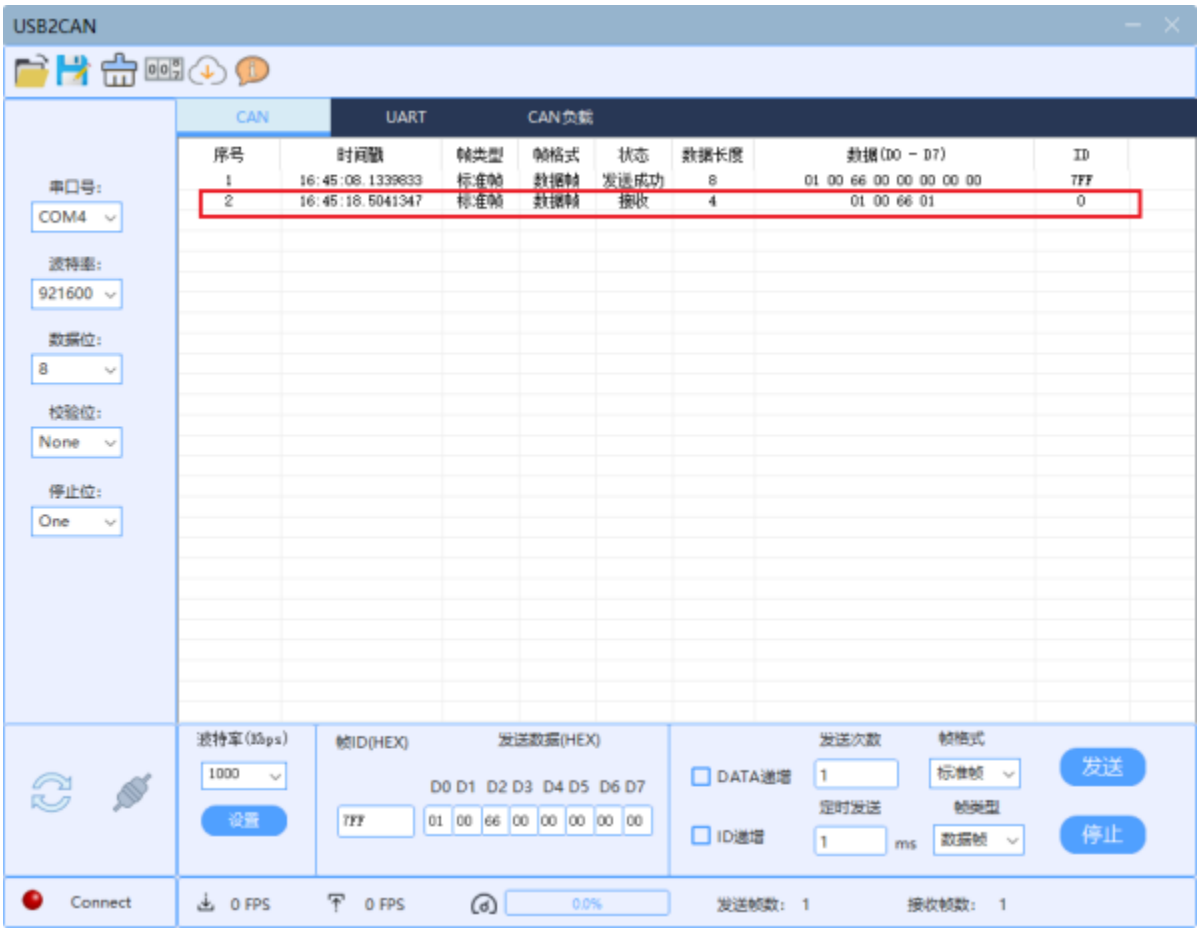
D[2] = 66 D[3] = 00 根据上图协议所示

D[4]到D[7]为任意参数即可，电机内部驱动会自动忽略这几位，可以全部设置成00

整体发送数据配置如下图所示。**在点击发送开始校准前，请固定好电机定子部分，以免电机旋转发生意外，校准期间不要触碰电机转子部分，以免影响校准精度。**



点击发送后，电机会自动校准开始旋转，校准完后，电机将返回状态帧，代表校准成功



校准完成后，校准数据会自动存储在电机驱动中。

5.4.2 标定参数

标定参数可以使电机自动获取**相电阻、相电感、磁链、粘滞系数、转动惯量**这五项关键参数。 在电机出厂前已经标定过 ，可以直接使用，若电机出现问题可尝试标定参数解决。

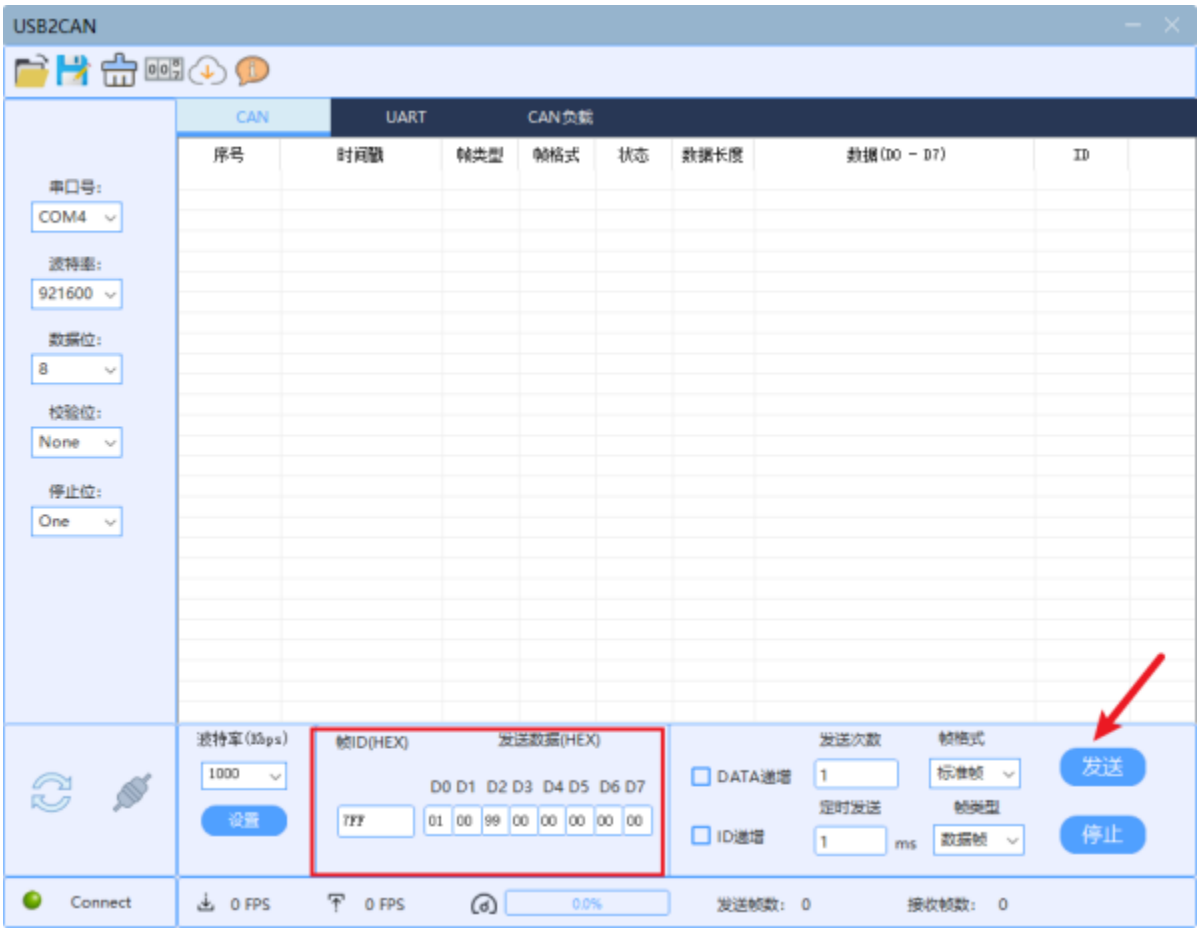
❖ **标定参数**

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0x7FF	STD	CANID_L	CANID_H	0x99	0x00	xx(don’ t care)			

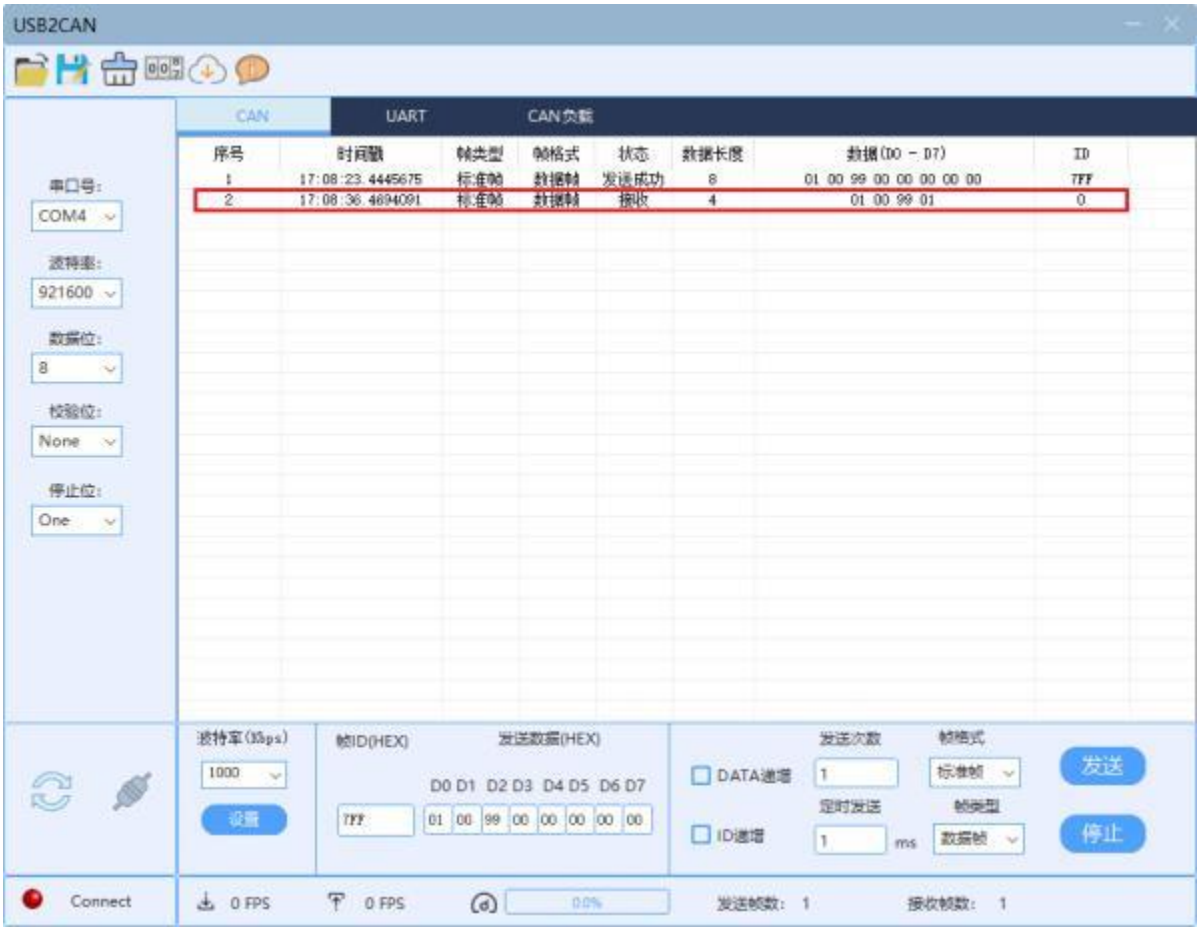
标定完成后，电机会主动返回标定完成指令，其帧格式如下：

报文 ID	属性	D[0]	D[1]	D[2]	D[3]
MST_ID	STD	CANID_L	CANID_H	0x99	01

根据上图协议发送数据为 帧ID = 0x7FF 数据 01 00 99 00 00 00 00 00 **点击发送标定参数前请固定好电机定子部分，以免电机旋转发生意外，校准期间不要触碰电机转子部分，以免影响标定精度。**



标定完成后，电机将反馈状态帧 01 00 99 01，表示标定完成，标定参数会自动储存在电机驱动中



5.4.3 读取参数

用户可发送读取参数指令读取指定参数，读取参数的指令如下

❖ 读取参数

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0x7FF	STD	CANID_L	CANID_H	0x33	RID	xx(don' t care)			

RID 为寄存器地址：

RID为我们需要读取参数的寄存器地址，选择对应的RID可读取不同的参数。以下表为例

寄存器地址	变量	描述	读写	数据类型
0	UV_Value	低压保护值	RW	float
1	KT_Value	扭矩系数	RW	float
2	OT_Value	过温保护值	RW	float
3	OC_Value	过流保护值	RW	float
4	ACC	加速度	RW	float
5	DEC	减速度	RW	float
6	MAX_SPD	最大速度	RW	float
7	MST_ID	反馈 ID	RW	uint32
8	ESC_ID	接收 ID	RW	uint32
9	TIMEOUT	超时警报时间	RW	uint32
10	CTRL_MODE	控制模式	RW	uint32
11	Damp	电机粘滞系数	RO	float
12	Inertia	电机转动惯量	RO	float

寄存器地址：需要读取对应参数的RID。十进制， **发送给电机时需转化为16进制发送**，例如**RID = 10，发送给电机的D[3] = 0x0A**

变量：寄存器名称。

描述：寄存器定义。

读写： RW,可读取可写入， RO，只能读取不可写入。

数据类型：读取的**参数数据类型**通过CAN通信反馈帧的**Data[4]—Data[7]**反馈，可拼接成float和uint32类型变量表示。

所有的寄存器与对应参数的表可查阅轮毂电机说明书 https://gitee.com/kit-miao/DM-H6215/tree/master/%E8%AF%B4%E6%98%8E%E4%B9%A6 或
固件更新日志链接 https://gitee.com/kit-miao/motor-firmware

以**读取当前电机低压保护值为例**，发送给电机的数据为

D[3]为读取参数的寄存地址 RID = 0

点击发送后，电机将会返回寄存器数据，帧格式如下

读取成功后，会返回该寄存器的数据，帧格式如下：

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
MST_ID	STD	CANID_L	CANID_H	0x33	RID	数据			

数据为浮点型数据，低位 D4，最高位为 D7，下同。

注意D[4]为最低位，D[7]为最高位，例如反馈的数据为 01 02 03 04 转化为32位数据就是0x04030201

发送电机读取低压保护值的指令后，电机会反馈当前参数的反馈帧

D[4]—D[7]为 00 00 70 41 转化为32位的数据为 **0x41700000**，可以通过一些网站在线转化32位浮点数或在C语言程序中使用Union联合体、指针强制转化等方式将32位数据转化位float类型变量。0x41700000转化为浮点数为15.0，当前电机的低压保护值为15.0。

5.4.4 写入参数

6215电机可通过CAN协议直接修改参数，帧格式如下，写入的参数若没有收到保存命令，将在掉电后丢失，保留修改前的参数数值

❖ **写入参数**

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0x7FF	STD	CANID_L	CANID_H	0x55	RID	数据			

例如将电机的低压保护值修改为18.0，18.0浮点数转化为32位数据为0x41900000，D[4]为最低位D[7]为最高位，所以发送给电机的数据为D[4] = 00 D[5] = 00 D[6] = 90 D[7] = 41 低压保护值的RID为0 D[3] = 00 写入参数的命令码D[2] = 55

点击发送后，电机会反回写入的参数。

报文 ID	属性	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
MST_ID	STD	CANID_L	CANID_H	0x33	RID	数据			

写寄存器数据立即生效，但无法进行存储，掉电后丢失，需要发送存储参数的命令，将修改的参数全部写入片内。

返回数据的D[4]—D[7]为 00 00 90 41 即修改的低压保护值18.0V。

CAN | UART | CAN负载

序号	时间戳	帧类型	帧格式	状态	数据长度	数据(D0 - D7)	ID
1	18:47:57.4830039	标准帧	数据帧	发送成功	8	01 00 55 00 00 00 90 41	7FF
2	18:47:57.4830039	标准帧	数据帧	接收	8	01 00 55 00 00 00 90 41	0

修改后的参数若没有收到保存命令，则在电机掉电后恢复为原来的参数

5.4.5 保存参数

在通过CAN通信直接修改电机内部参数后， **若没有收到保存该参数的指令，修改后的参数将掉电后丢失**，若需要永久修改参数，需在修改参数后再次发送保存参数的指令，掉电后该参数也不会丢失。

3.存储参数

报文 ID	属性	D[0]	D[1]	D[2]	D[3]
0x7FF	STD	CANID_L	CANID_H	0xAA	0x01

注意:

1.发送此命令成功后，所有的参数将会以当前参数全部保存，无需逐条保存。

2.保存指令不可在电机使能状态下发送，电机无法在使能情况下保存数据。

例如将电机的低压保护值修改为18.0后，将当前参数保存至电机 给电机发送保存参数指令

点击发送后 电机将会反馈帧01 00 AA 01代表保存成功，当前电机参数掉后也不会丢失

5.5 特殊参数的修改

5.5.1 CAN_ID的修改

CAN通信协议发送给电机的数据包含了电机的**CAN_ID**， D[2]和D[3]分别为CAN_ID的低位和高位例如 CAN_ID = 0x0001，则D[0] = 0x01 D [1] = 0x00。所以**修改完CAN_ID需要以新的CAN_ID发送**，例如将CAN_ID从0x0001修改为0x0002 向电机发送如下数据

当前**电机的ID**为0x0001， D[0] = 01、 D[1] = 00。**写入参数的命令码**D[2] = 0x55, **CAN_ID对应的RID**为8， D[3] = 0x08 将**CAN_ID**修改为0x00000002 = 0x0002 =0x02 = 2 **D[4]—D[7] = 02 00 00 00**

8	ESC_ID	接收 ID	RW	[0,0x7FF]	uint32

点击发送后 电机反馈帧为02 00 55 08 02 00 00 00

此时代表电机ID的D[0]和D[1]已经被修改为 02 00，对应的CAN_ID为0x02。此时若再给电机发送 01 00开头的指令，则电机不会反馈数据也不会响应指令，需将D[0]和D[1]修改为对应的ID即可。**若修改后不发送保存指令，则电机掉电后将会变为修改前的ID**

5.5.2 CAN_ID的查找

电机出厂时默认CAN_ID为1，因为6215轮毂电机无法直接通过串口读取参数，再者，通过CAN协议发送给电机读取指令时需要知道CAN_ID才能发送。此时可以通过枚举的方式获得。

使用枚举的方式给电机发送读取命令，从CAN_ID = 1开始发送，每次发送后，下次发送的CAN_ID+1， CAN_ID的范围是0-0x7FF，从CAN_ID = 1到CAN_ID = 0x7FF 给电机发送2047次读取命令，当电机CAN_ID匹配时，就会发送反馈帧， **由此便可以搜索到电机的CAN_ID**

建议将电机ID在修改时改为**0x01—0x0F**之间， 一是电机状态反馈帧数据中的CAN_ID为4bit （二进制1111）对应最大值为0x0F，如果超过0x0F，电机状态反馈帧数据中的状态位不准确（**此反馈帧是指控制电机时电机的位置、速度、力矩等信息**）。二是减少我们查找CAN_ID的次数。

例如将电机CAN_ID修改为**0x0A**后忘记该电机的ID了，通过**枚举16次（对应0x00-0x0F）**来查找该电机CAN_ID。

D[0] = 01 D[1] = 00 对应当前发送给**CAN_ID = 0x01** 的电机， **D[2] = 33**电机**读取参数的命令码**， **D[3] = 08** 为**存放CAN_ID的寄存器的地址** （任意一个RID都可以，主要看电机会不会给反馈帧） **D[4]— D[7] = 0** (任意参数都可以)

勾选DATA递增，勾选之后若发送次数大于1，则每次发送时数据自动+1 （从D[0]开始）

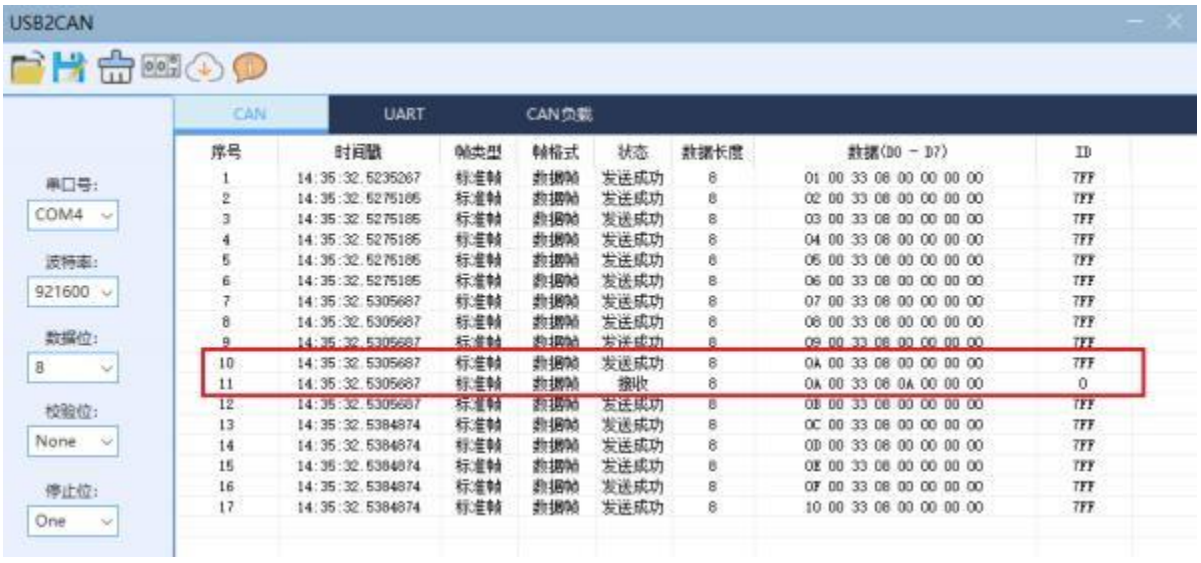
例如 01 00 33 08 00 00 00 00 **发送次数为3** 则一共发送三帧数据包

01 00 33 08 00 00 00 00、02 00 33 08 00 00 00 00、03 00 33 08 00 00 00 00

这样就会给依次给CAN_ID = 0x01、0x02、0x03电机发送读取指令。

所以我们发送次数为16， CAN_ID对应为0x01—0x0F

点击发送后



只有当发送CAN_ID为0x0A时，电机才会发送反馈帧，说明此时CAN_ID为0x0A

5.5.3 控制模式的修改

通过查阅固件更新日志 https://gitee.com/kit-miao/motor-firmware

新版固件协议手册可以知道 **控制模式的寄存器地址为（RID）10，范围为1到4。**

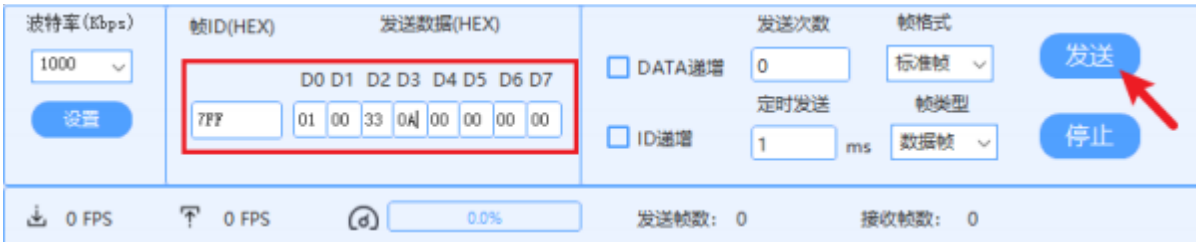
10	CTRL_MODE	控制模式	RW	[1,4]	uint32

1到4对应四种控制模式

固件版本 13 以上支持多种模式互相切换，目前支持的控制模式如下：

编码	模式
1	MIT
2	位置速度
3	速度
4	力位混控

读取当前电机控制模式，给电机发送 01 00 33 0A 00 00 00 00 注意：控制模式的RID=10， **转化为16进制为0x0A而不是0x10**，发给电机的D[3] = 0x0A



电机的反馈帧为01 00 33 0A 01 00 00 00 D[4] = 0x01 表明当前控制模式为MIT模式

若修改电机参数，例如将电机修改为位置速度模式，给点击发送 01 00 55 0A 02 00 00 00 其中D[4] = 0x02 表示位置速度模式

点击发送之后，电机的反馈帧为下图，表示修改成功了

修改电机控制模式之后，**请根据修改后的控制模式协议控制电机**，若后续不发送保存命令，掉电后电机会变为修改前的模式

5.5.4 CAN通信波特率的修改

6215 电机支持修改 CAN 通信的波特率，根据固件更新日志 https://gitee.com/kit-miao/motor-firmware

V13版本说明

35	can_br	CAN 波特率代码	RW	[0, 4]	uint32

支持特定波特率修改，目前的波特率如下：

编码	波特率
0	125K
1	200K
2	250K
3	500K
4	1M

CAN波特率代码寄存器地址（RID）为35，范围0—4，对应波特率125K、200K、250K、500K、 1M

电机出厂默认CAN波特率为1M

将电机波特率修改为500K，向电机发送指令01 00 55 23 03 00 00 00， **35的16进制表示为0x23 所以D[3] =0x23 D[4] = 0x03 将电机CAN通信波特率修改为500Kbps**

点击发送之后，电机此时波特率已经修改，无法正常返回反馈帧，若按1000Kbps（1Mbps）波特率继续发送指令，上位机会反馈发送失败的状态

将**USB转CAN波特率修改至500Kbps** 并点击设置后再次发送

指令显示发送成功并收到反馈帧，若之后不修改波特率，则以500Kbps与电机通信

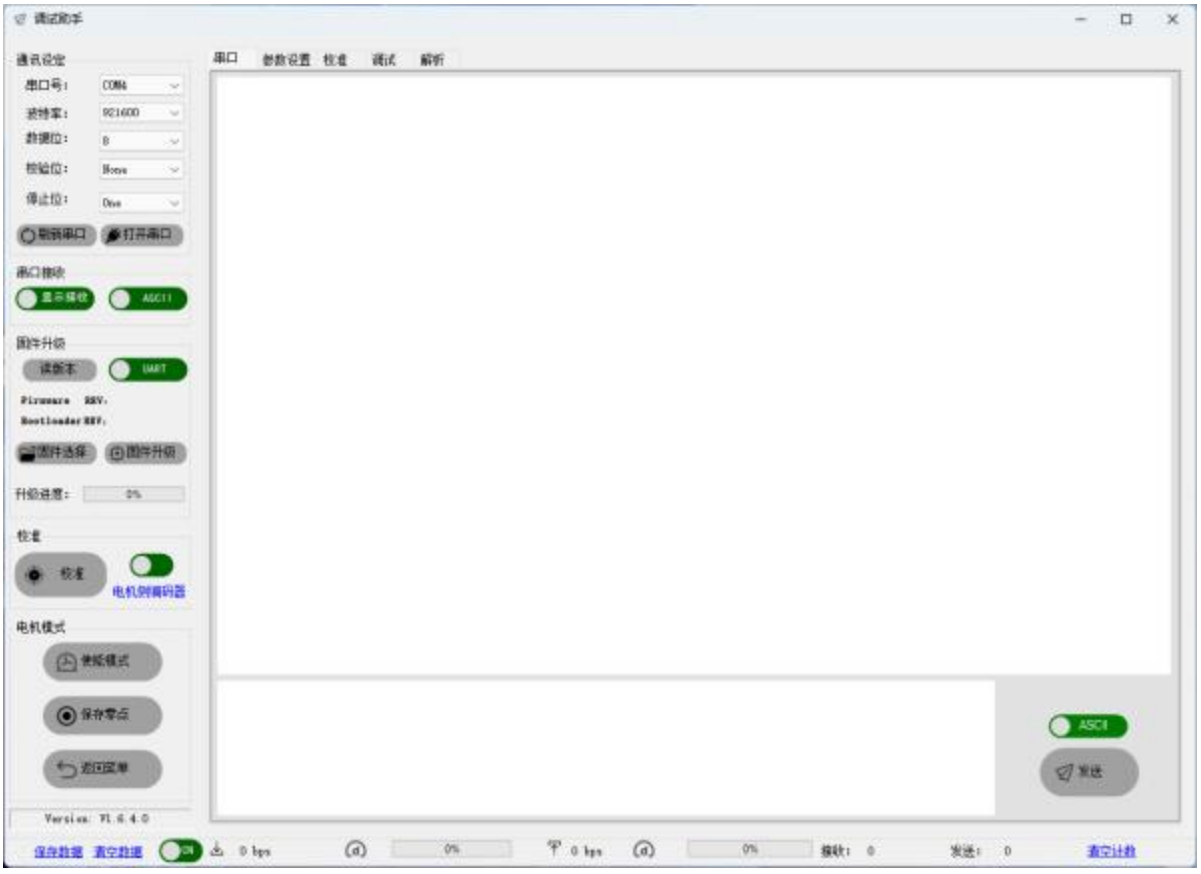
修改后若电机未收到保存参数指令，掉电后波特率将变为修改前的参数

5 上位机调试电机

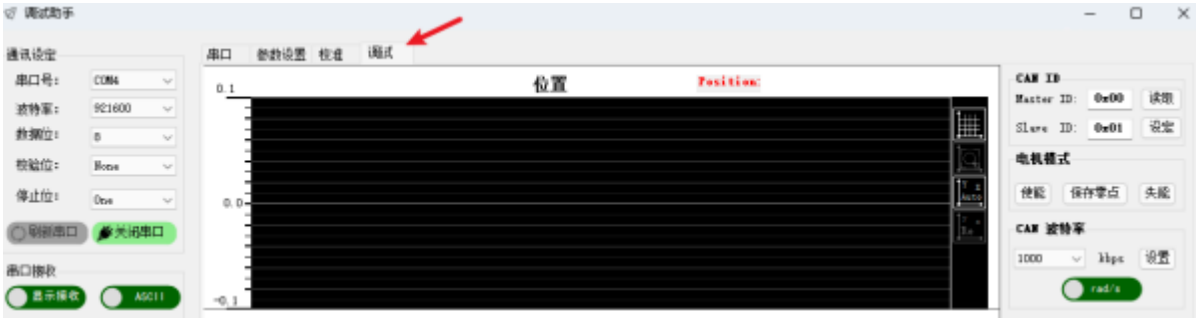
6215轮毂电机可以使用调试助手的调试功能简单调试电机，打开固件升级时下载的调试助手

接下来用上位机通过**三种模式 （MIT模式、速度位置模式、速度模式**）来调试电机。，使用每个模式时会简单介绍一下控制的原理。

调试电机时，请保证电机固定好，并准备随时切断电机供电，以免发生意外。

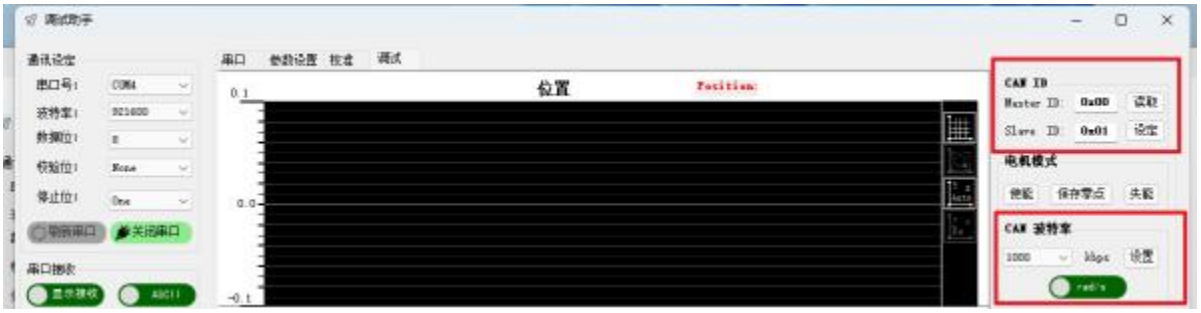


点击调试，进入调试界面。



5.1 CAN通信设置

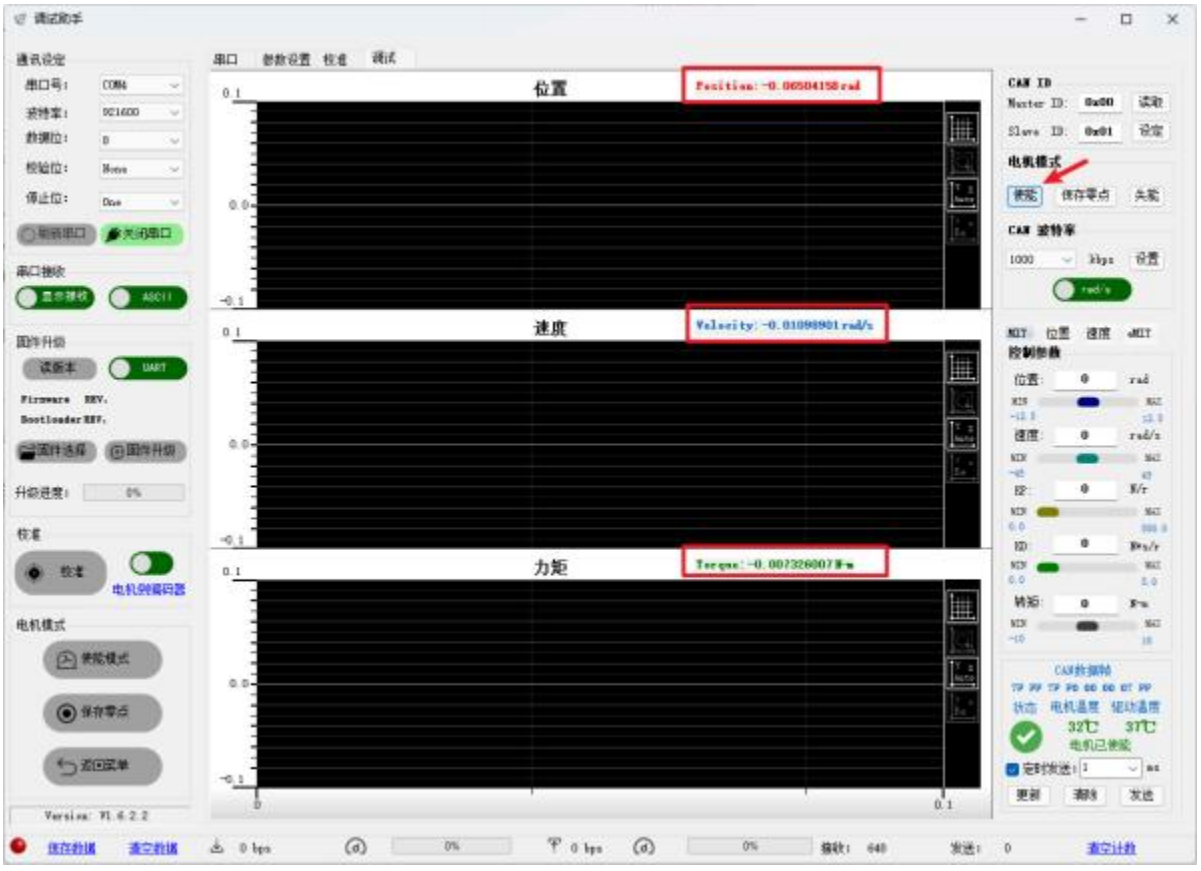
上位机通过调试模块将数据转化为CAN总线协议数据发给模块，请保证调试模块上的GH1.25 2pin接口连接电机且保证线序正确（**CAN_L接CAN_L,CAN_H接CAN_H**）



确保**Master_ID**和**Slave_ID**与通过**USB转CAN软件读取到的Master_ID和CAN_ID保持一致**，**Slave_ID**即前面的**CAN_ID**

CAN波特率保持与**电机驱动波特率一致**，默认为**1Mbps**

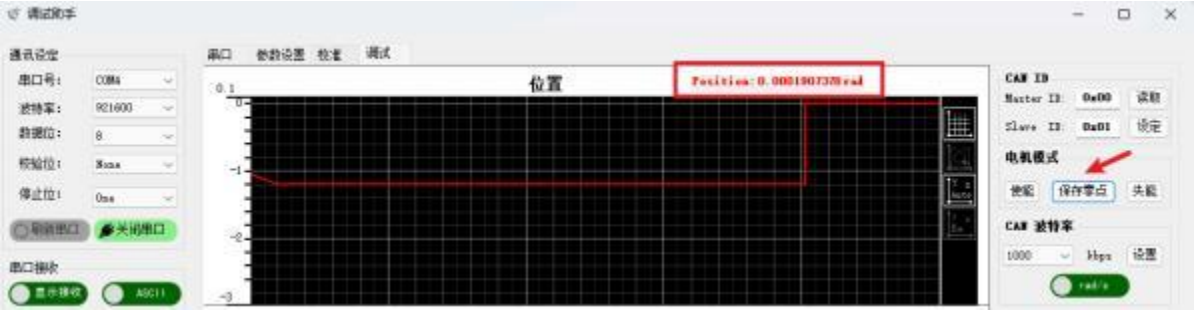
5.2 使能与失能电机



点击**使能**， 6215轮毂电机没有**LED显示灯**， **在MIT模式下**， **可以转动电机判断**，电机会从失能状态的略带阻尼的感觉变成顺滑感。**在非MIT模式下，电机会锁死在当前位置**，红框中三项本没有的参数将会显示使能电机时刻的电机数据。此时给电机发送控制指令将会响应

点击**失能**，电机的指示灯会从**绿色跳为红色**，转动电机会略带阻尼感，此时给电机发送控制指令将不会响应，当电机发送**震荡、抖动、甚至失控**时，请点击**失能**停止电机

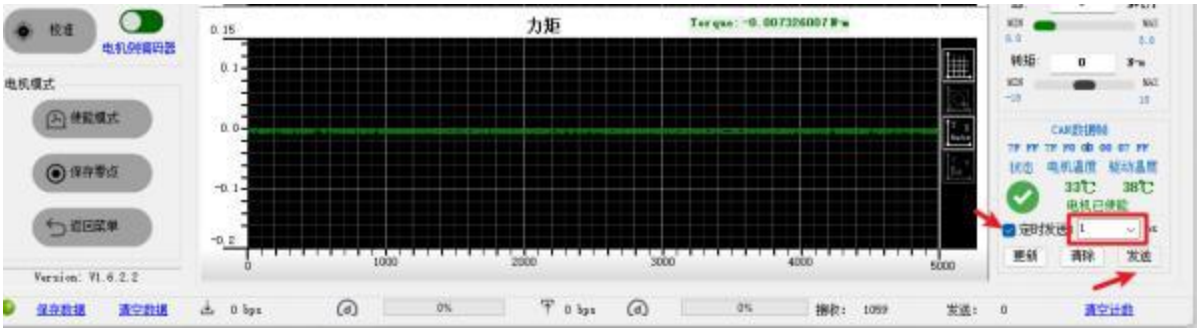
5.3 保存电机零点



H6215为霍尔增量式编码器电机，**调试界面中的"保存零点"功能无效。**

注意：电机重新上电后，系统仍将以当前实际位置作为零点。

5.4发送控制指令给电机

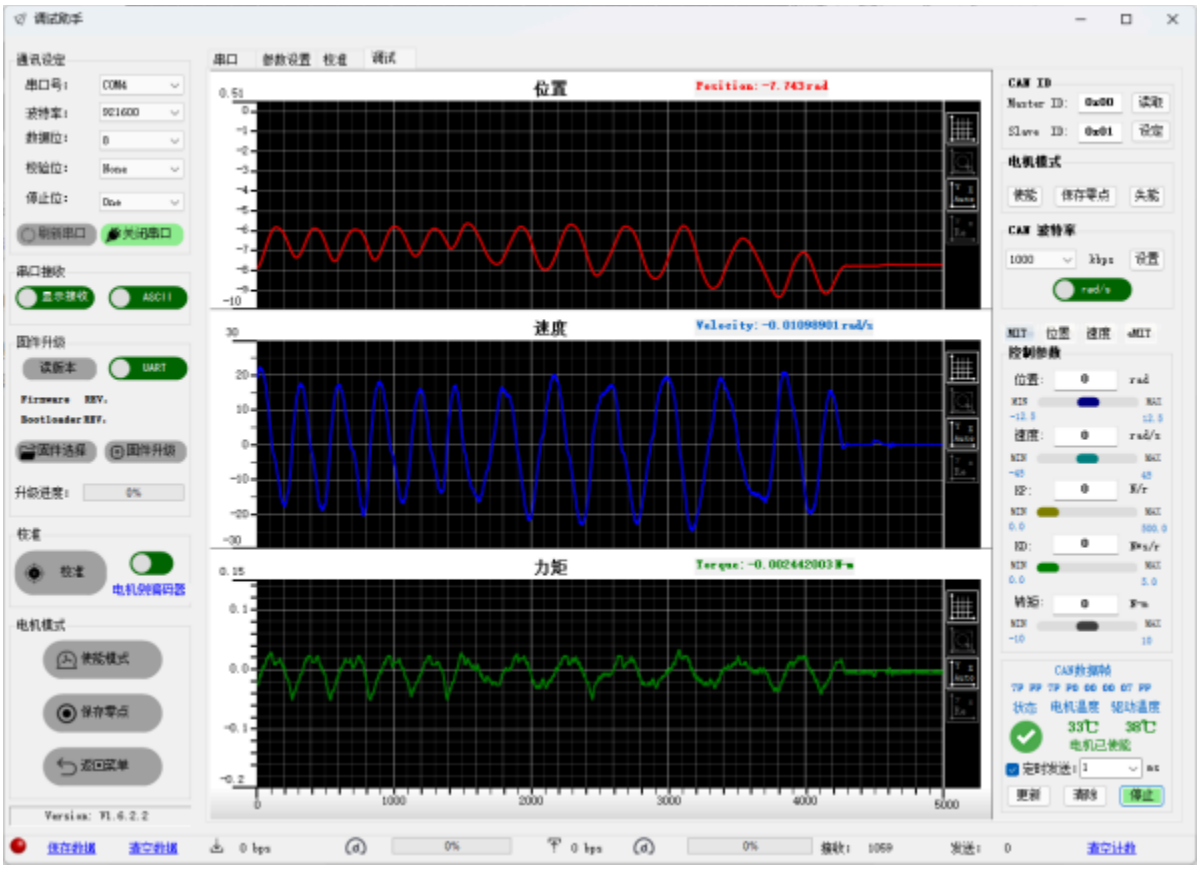


达妙电机收到一帧控制指令时，才会返回一帧数据帧，所以当不给电机发送控制指令时，电机也不会反馈数据。

右下角，勾选**定时发送**，即连续发送，时间单位（**微秒**） **ms**，设置为1代表**每1ms发送一次**，即**1秒发送1000次**控制指令给电机，电机也会以**1秒1000次反馈数据。**

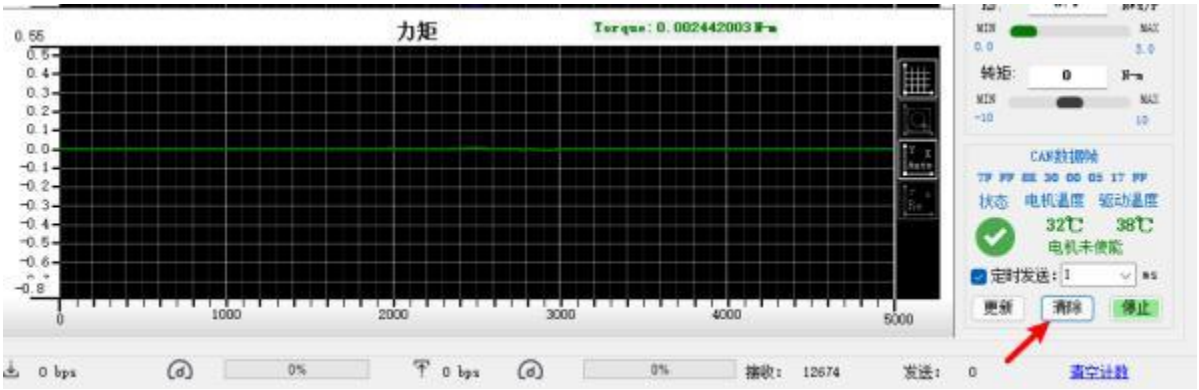
点击发送前，请保证控制参数内所有参数为0！以免电机收到控制指令时突然启动发生意外！！！

点击**发送**，转动电机，三条数据曲线将会**实时显示**



5.5 清除错误指令

清除是**清除错误**指令，是在**发生错误的情况下将错误清除**，并**强制失能电机。建议在电机发生错误时才使用此功能。**

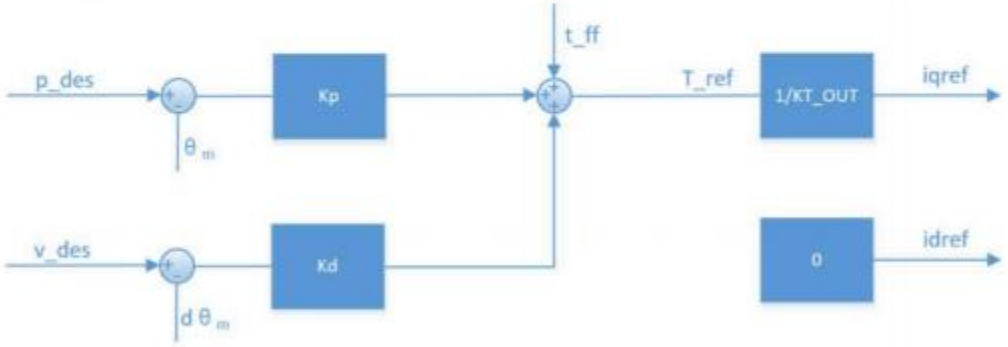


5.6 MIT模式

根据调试助手使用 说明书（达妙驱动控制协议） V1.4 (达妙驱动控制协议手册 https://gitee.com/kit-miao/damiao-document)中第三章工作模式关于 **MIT 模式**的描述

3.1 MIT 模式

MIT 模式是为了兼容原版 MIT 模式所设计，可以在实现无缝切换的同时，能够灵活设定控制范围（P_MAX,V_MAX，T_MAX），驱动器将接收到的 CAN 数据转化成控制变量进行运算得到扭矩值作为电流环的电流给定，电流环根据其调节规律最终达到给定的扭矩电流。其控制示意框图如下：



根据 MIT 模式可以衍生出多种控制模式，如 kp=0,kd 不为 0 时，给定 v_des 即可实现匀速转动;kp=0,kd=0，给定 t_ff 即可实现给定扭矩输出。

注意：对位置进行控制时，kd 不能赋 0，否则会造成电机震荡，甚至失控。

该模式下可以通过给定**期望力矩**（**T_ref**），经过~~KTOUT~~ 的放缩, 由电机内部驱动转化为期望扭矩电流**iqref**，驱动电机。

期望力矩由三个部分组成:

第一部分:

p_des: **期望位置**， **用户给定**，即期望电机转到某个位置的数值。单位**rad**， **在超过P_MAX时会发生溢出现象，即从相反的方向继续计数。**

θm :**电机实际位置**，由**电机编码器反馈**，即当前电机的位置，单位**rad**， **该参数不会超过P_MAX。**

Kp： **电机位置环的比例系数**，范围0-500。

位置期望力矩 = （p_des - θm） * Kp **即位置误差乘比例系数**

第二部分:

v_des: **期望速度**， **用户给定**，即期望电机转到某个速度的数值。单位**rad/s**， **在超过V_MAX时会发生溢出现象，即从相反的方向继续计数。**

dbm :**电机实际速度**，由**电机编码器反馈**，即当前电机的速度，单位**rad/s**， **该参数不会超过V_MAX。**

Kd： **电机速度环的微分系数**，范围0-5。

速度期望力矩 = （v_des - dbm） * Kd **即速度误差乘微分系数;**

第三部分：

t_ff: 直接期望力矩， 用户给定，即期望电机输出对应力矩的值，单位**NM， 该参数给定不能超过T_MAX。**

发送给电机的期望总力矩 = 位置期望力矩 + 速度期望力矩 +直接期望力矩

根据控制框图， **MIT模式**可以控制电机**输出指定力矩**，控制电机转动到**指定位置**，控制电机转动**指定速度**。

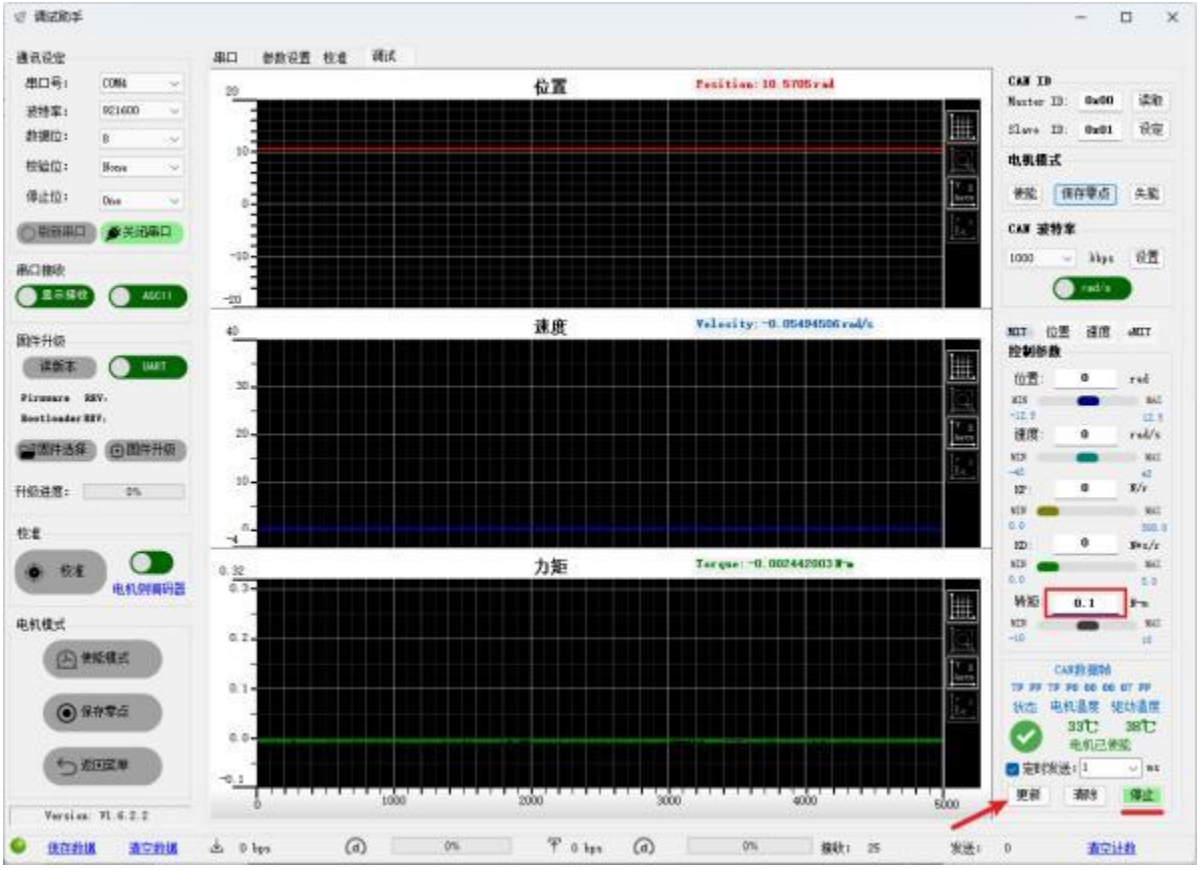
5.6.1 控制电机输出指定力矩

请注意，即使让电机输出**很小的力矩** （如0.1NM），若电机**没有负载** （**即没有一个力阻碍电机转动**），电机将会加速到很快的速度， **调试时请固定好电机，以免意外发生。**

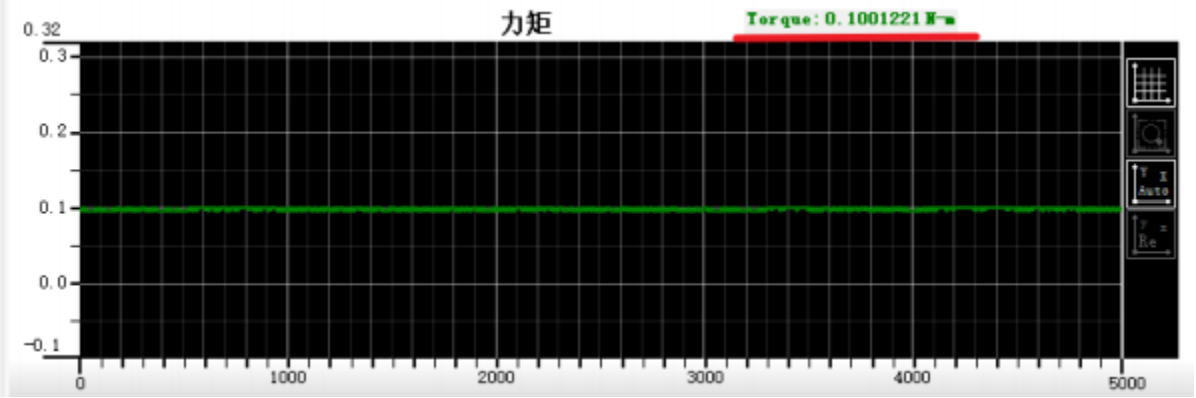
根据控制框图，想要电机输出指定力矩，直接给定**t_ff**，其他参数均为0即可。回到调试助手

请确保电机已经**使能**，并处于**发送状态**

在控制参数中将转矩的数值修改为0.1， **其余四个参数均保持为0！！！** 即让电机输出**0.1NM**的力矩。**请确保电机固定好**，点击**更新**



更新后，电机会加速到33rad/s，可增加电机负载使电机停转，可以看到力矩参数稳定在0.1NM左右。

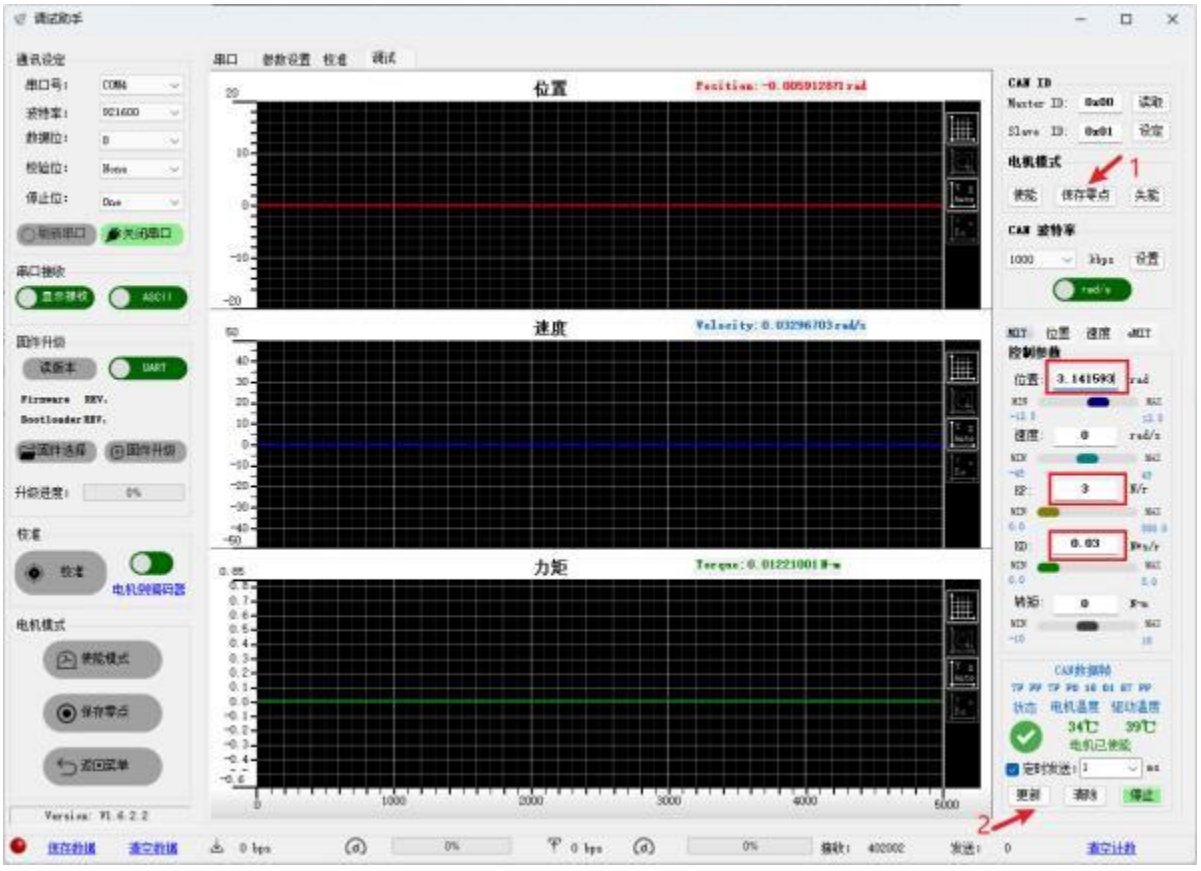


5.6.2 控制电机转到指定位置

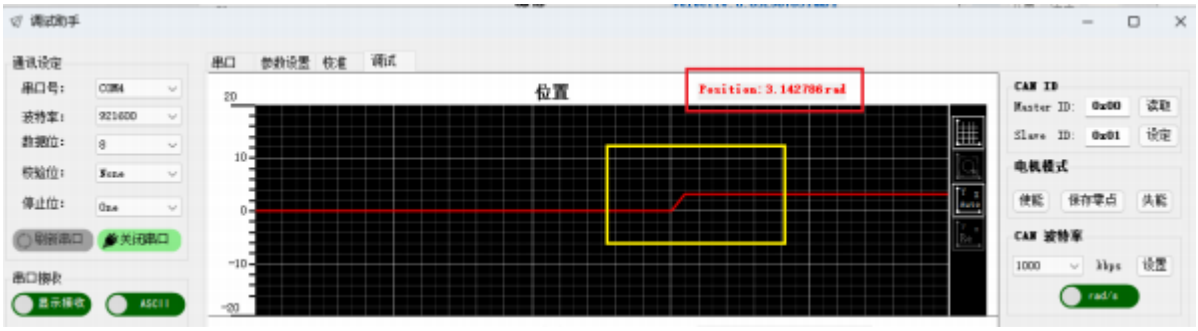
根据控制框图，控制电机转动到指定位置，期望力矩主要由电机位置误差*KP产生，但如果只给定KP， KD为0，控制电机位置时将**会产生剧烈震荡。**所以用**MIT模式**控制电机位置时， **一定要先给KD参数再给kP参数**，防止发生意外！！！

首先保存零点，将电机的多圈计数请0，再将电机的控制参数设置为， **位置：3.141593**，速度： 0， **KP：3，KD：0.03**，转矩： 0

点击**更新。**



电机将从位置**0转到3.14**左右，转化为角度就是转了**180度**



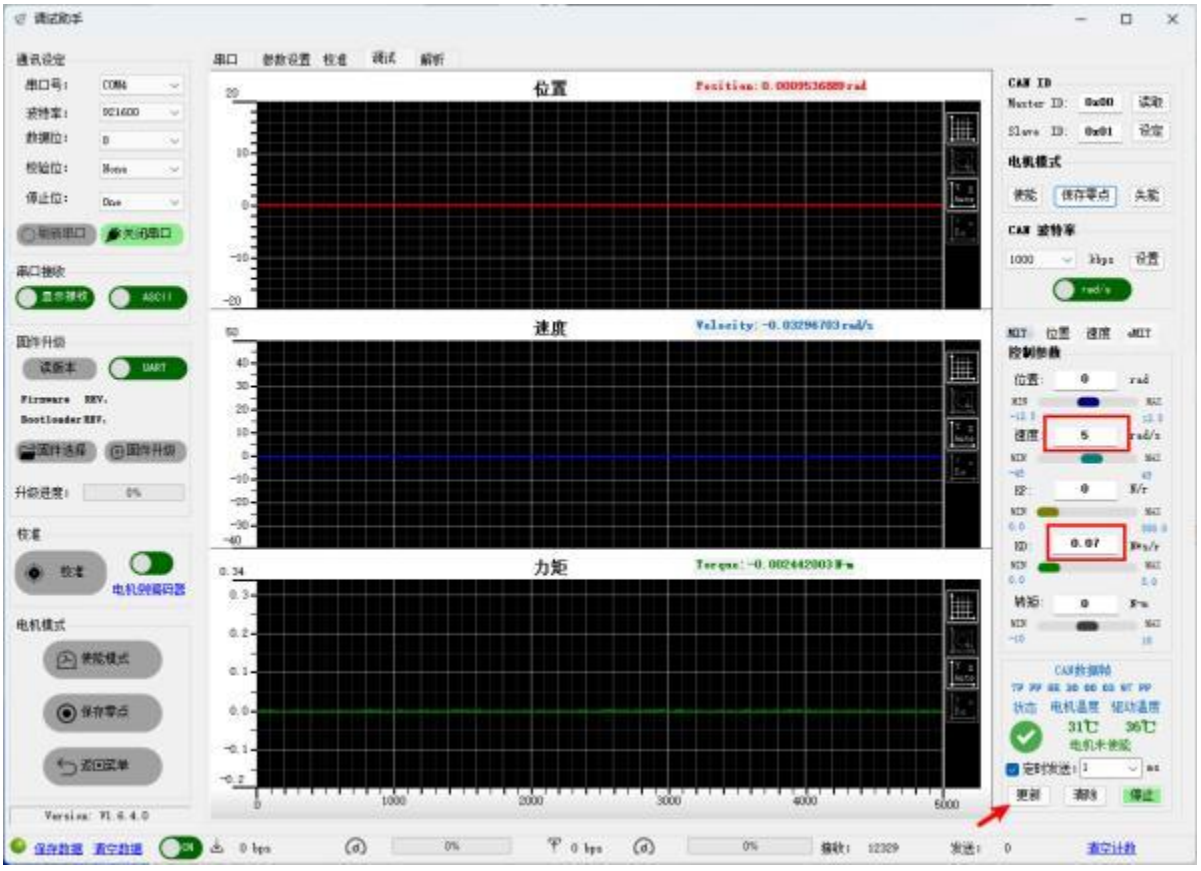
KP = 3、KD = 0.03是根据空载时电机调试出来相对合理的参数，用户请根据实际情况自行整定KP、KD参数，建议不要一下给太大，慢慢调试 。

5.6.3 控制电机转到指定速度

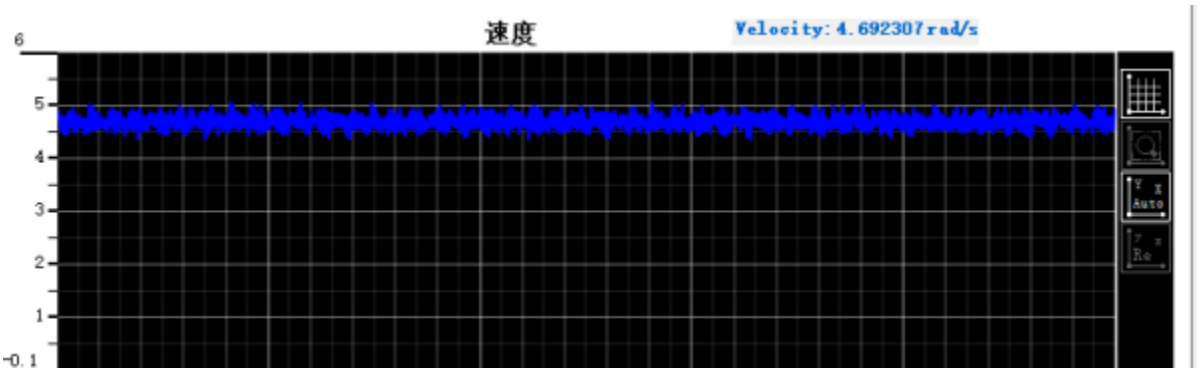
根据控制框图，控制电机转动到指定速度，期望力矩主要由电机速度误差*KD产生，只需要给定KD，再给定电机期望速度，电机就会产生力矩转到指定速度。

将电机的控制参数设置为，位置： 0， **速度：5**， KP： 0， **KD：0.07**，转矩： 0。

点击**更新**。



电机会加速到5rad/s左右。



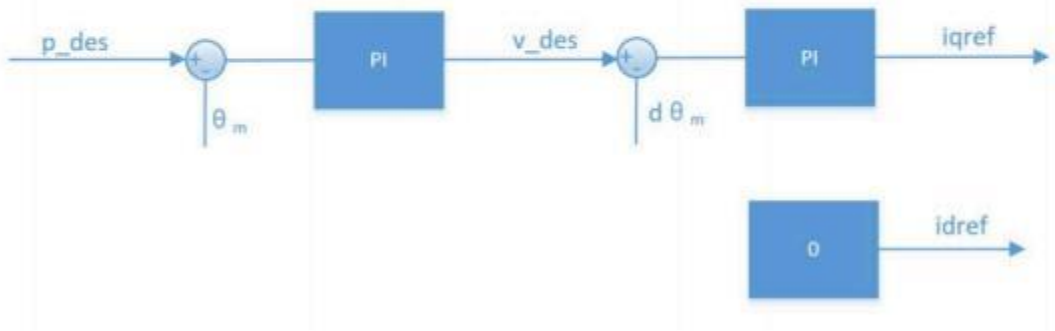
以上只是举了三个例子供用户理解，用户可根据实际应用场景使用MIT模式进行调参和控制。

5.7 位置速度模式

根据调试助手使用 说明书（达妙驱动控制协议） V1.4 ([关节电机/控制协议/调试助手使用说明书（达妙驱动控制协议） V1.4.pdf · kit/达妙科技 - 码云 - 开源中国 (gitee.com)](#))中第三章工作模式关于**位置速度模式**的描述

3.2 位置速度模式

位置串级模式是采用三环串联控制的模式，位置环作为最外环，其输出作为速度环的给定，而速度环的输出作为内环电流环的给定，用以控制实际的电流输出，其控制示意框图见下图：



p_des 为控制的目标位置，v_des 是用来限定运动过程中的最大绝对速度值。

位置串级模式如使用调试助手推荐的控制参数控制，可以达到较好的控制精度，控制过程相对柔顺，但响应时间相对较长。可配置的相关参数除 v_des 外，另有加/减速度进行设定，如控制过程中产生额外的震荡可提高加/减速度。

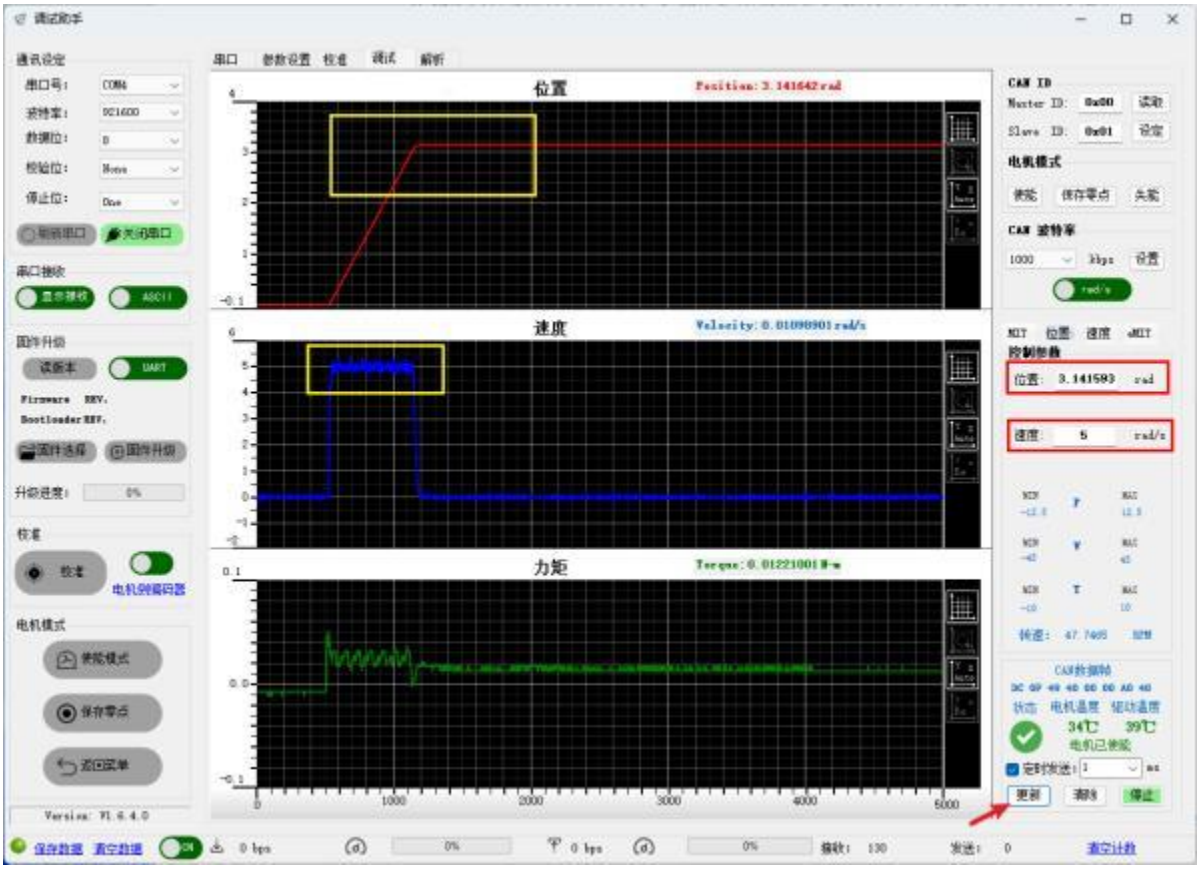
注意：p_des，v_des 单位分别为 rad 和 rad/s，数据类型为 float，阻尼因子必须设置为非 0 的正数，可参考速度模式的注意事项。

位置速度模式下，只需给定**期望位置、期望速度**，电机内部的**串级PID**就可以自动闭环，以**给定的期望速度**转到**指定位置**。

将电机修改为**位置速度模式**后，在调试模式下点击**位置**

使能电机，此时电机会**锁死**在当前位置。

此时给定位置，比如令位置等于 1，此时电机并不会转到位置1，因为速度给的是0，电机最大速度为0，也就是不会转动。**令位置等于3.141593，速度等于5**，点击**发送或更新**



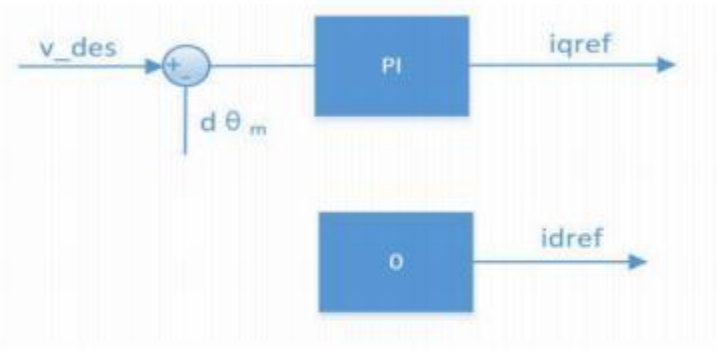
可以看到，位置由0转到3.14左右，而最大速度限制在了5rad/s，如果想让电机响应更快，可增大速度参数。

5.8 速度模式

根据调试助手使用 说明书（达妙驱动控制协议） V1.4 ([关节电机/控制协议/调试助手使用说明书（达妙驱动控制协议） V1.4.pdf · kit/达妙科技 - 码云 - 开源中国 (gitee.com)](#))中第三章工作模式关于**速度模式**的描述

3.3 速度模式

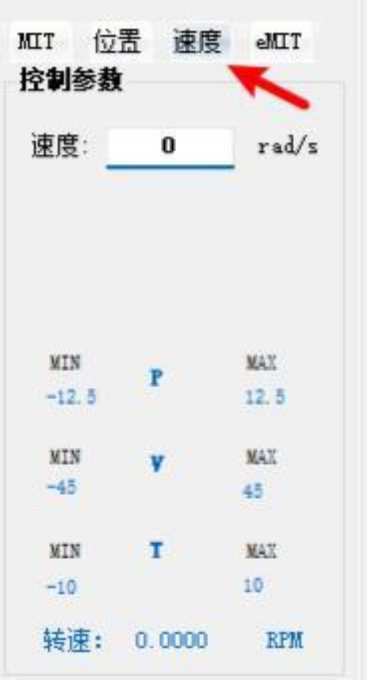
速度模式能让电机稳定运行在设定的速度，其控制示意框图如下：



注意：v_des 单位为 rad/s，数据类型为 float，如需使用调试助手自动计算参数，则需要设置阻尼因子为非 0 正数，通常情况下取值在 2.0~10.0，过小的阻尼因子会带来速度的震荡以及较大的过冲，过大的阻尼因子则会带来较长的上升时间，推荐的设定值为 4.0。

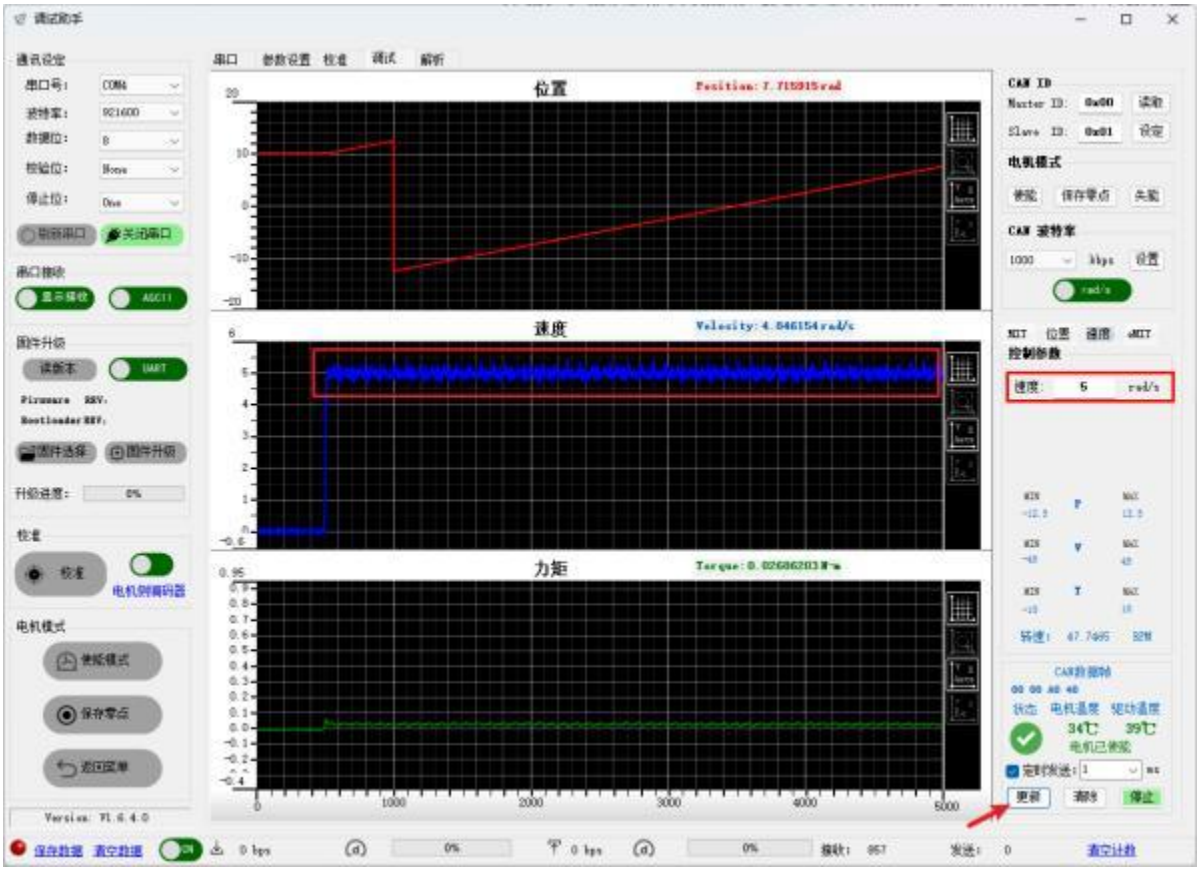
速度模式下，只需给定**期望速度**，电机内部的**PI**控制器就可以自动闭环，达到期望转速。

将电机修改为**速度模式**后，在调试模式下点击速度。



使能电机，此时电机会**锁死**在当前位置。

此时给定**速度5rad/s**，点击更新或发送，电机将会转动到5rad/s



6 STM32控制电机

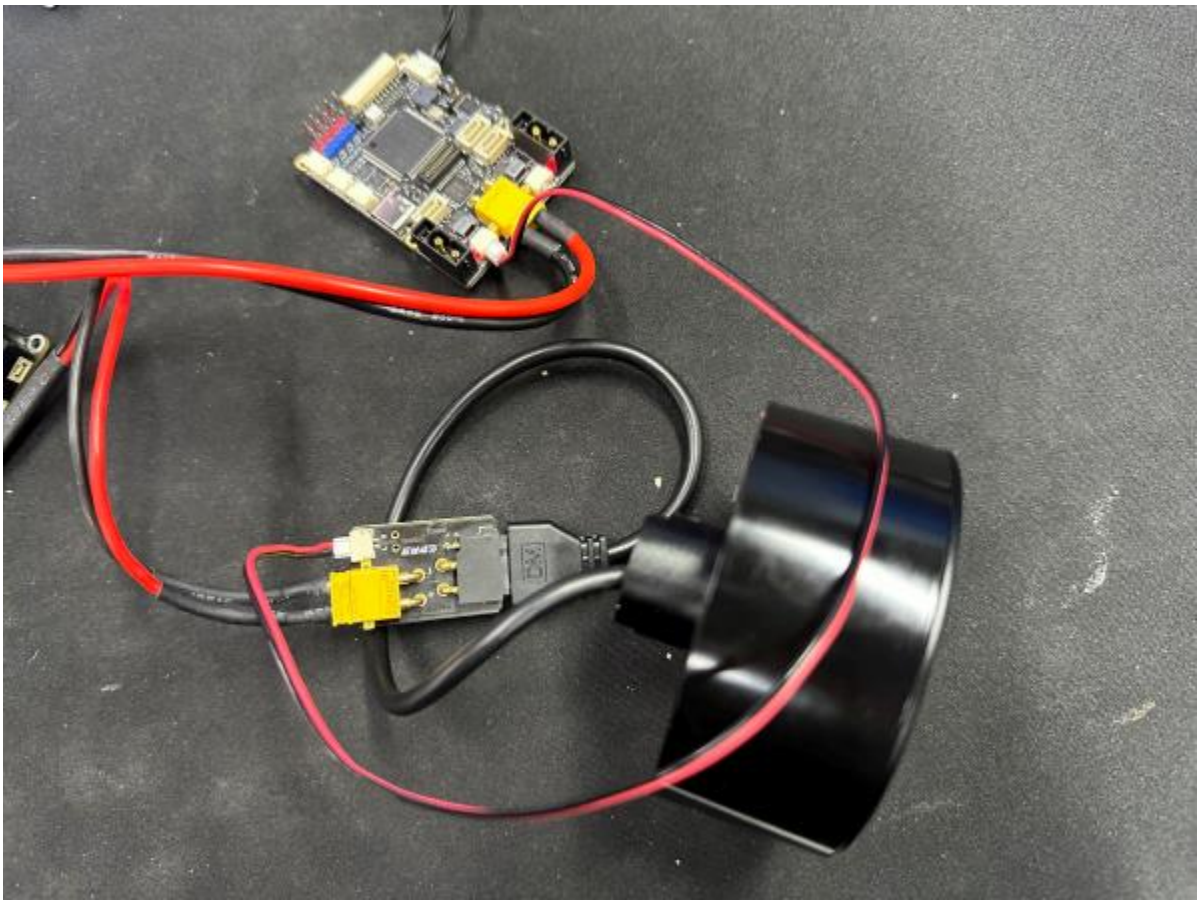
6.1 软件硬件

软件： **MDK_ARM(Keil) 5.35版本，最新版STM32CubeMX，Jlink下载器**

硬件： **达妙STM32开发板H723 DM-MC02**： [达妙STM32开发板H723 DM-MC02机器人轮足控制板机械臂板载BMI088-淘宝网 (taobao.com)]

达妙DM-H6215轮毂电机： [无刷直流电机达妙科技轮毂电机DMH6215直驱电机带驱动编码器-淘宝网 (taobao.com)]

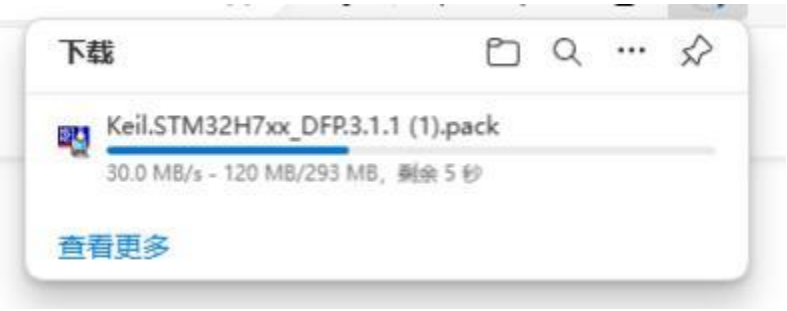
使用**24V**电压给**达妙MC-02和6215电机进行供电**，使用**GH1.25连接线-2pin**连接开发板上的**CAN1**和转接板**GH1.25-2pin口**，**注意线序（CAN_L接CAN_L、CAN_H接CAN_H)**



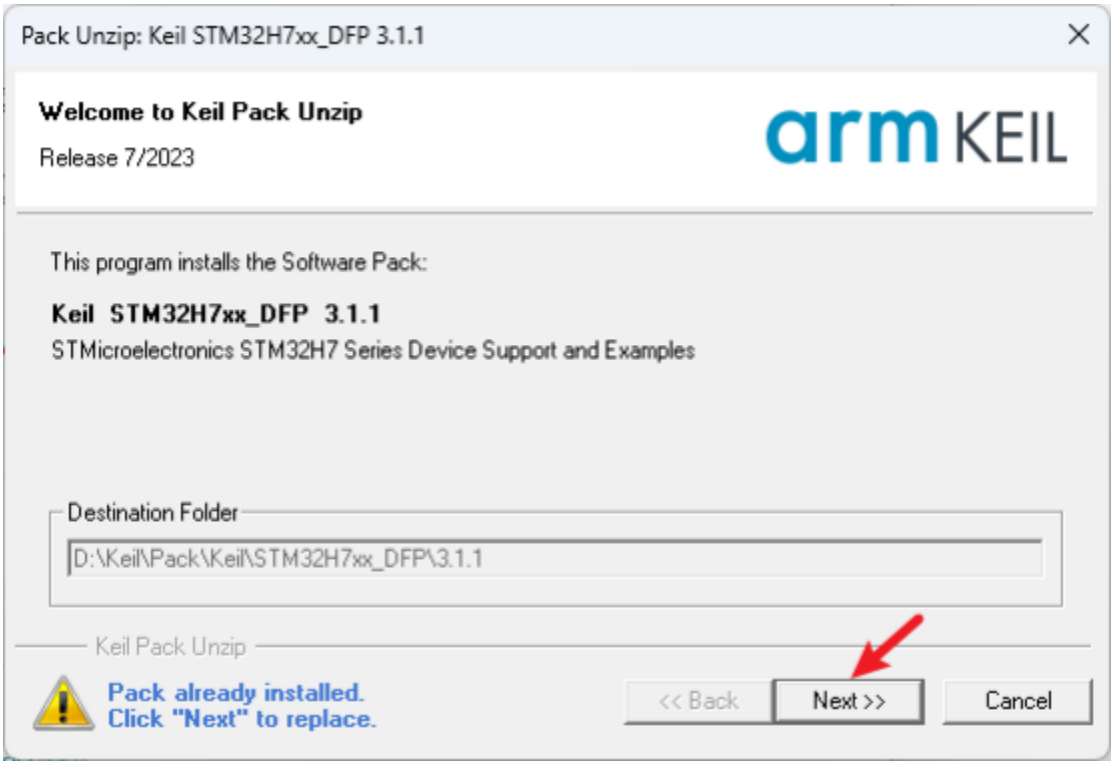
6.2下载STM32H7系列芯片包

点击Keil的官方STM32H7芯片包下载地址[Arm Keil | Keil STM32H7xx_DFP]

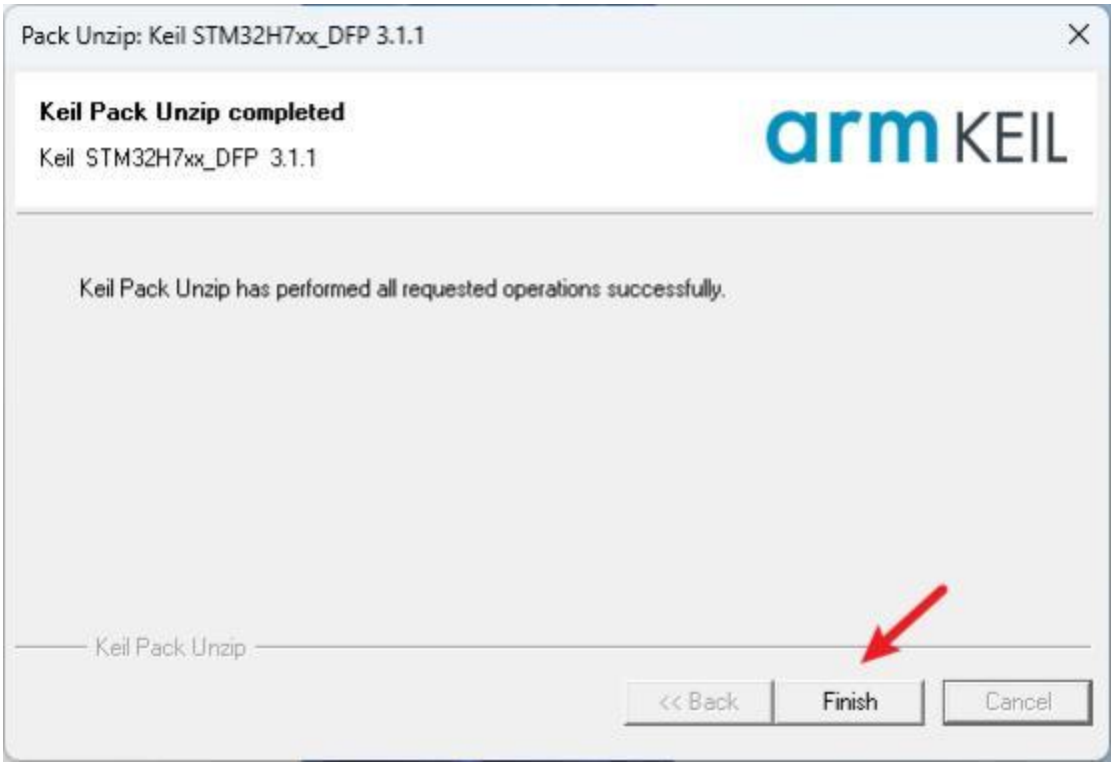
点击下载。等待芯片包下载完成



下载完成后，打开芯片包



点击Next，安装芯片包

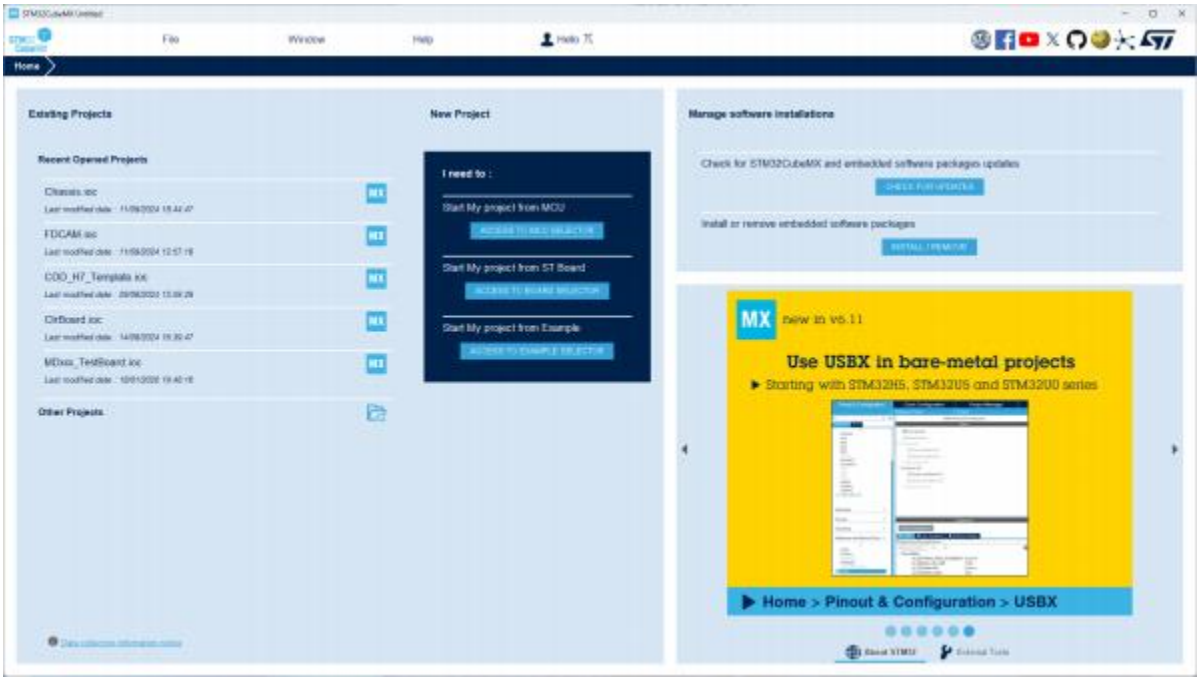


下载完成后 点击Finish，安装完成

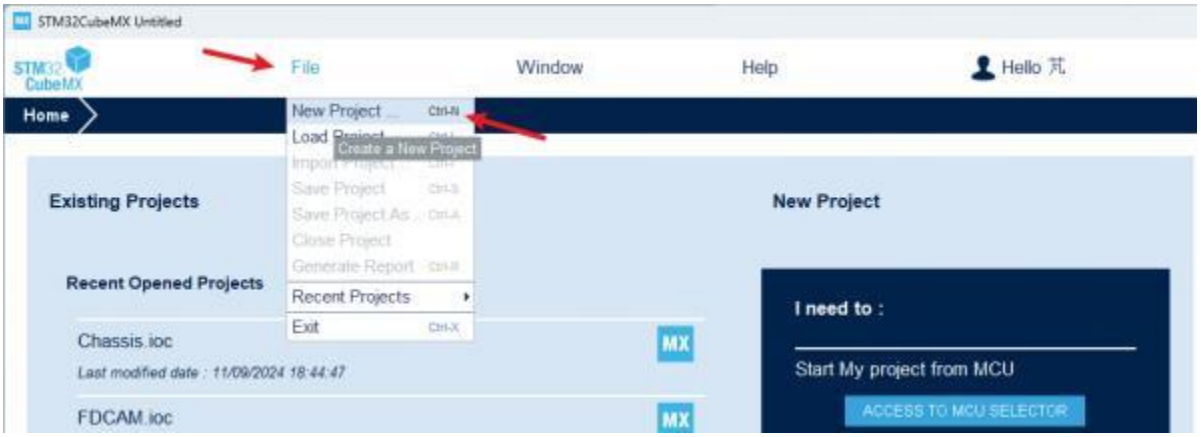
6.3 STM32CubeMX配置

4.3.1 新建工程

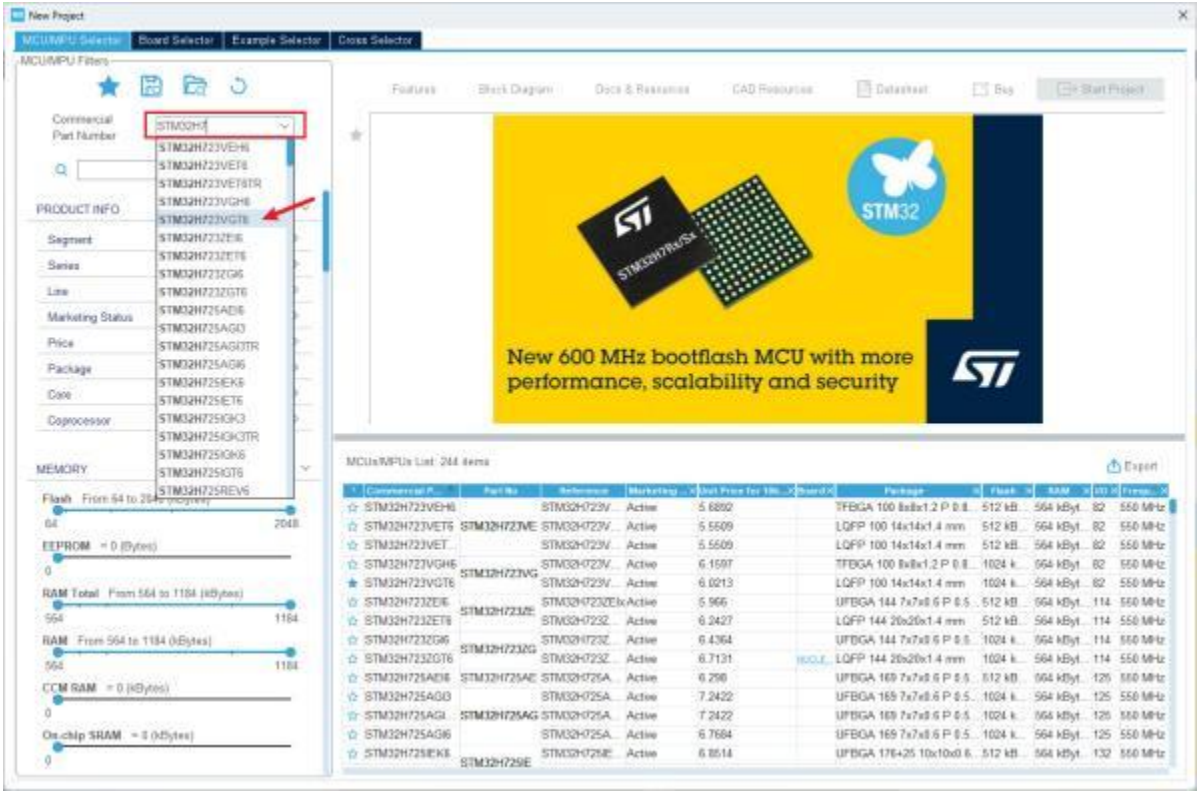
打开最新版本的STM32CubeMX



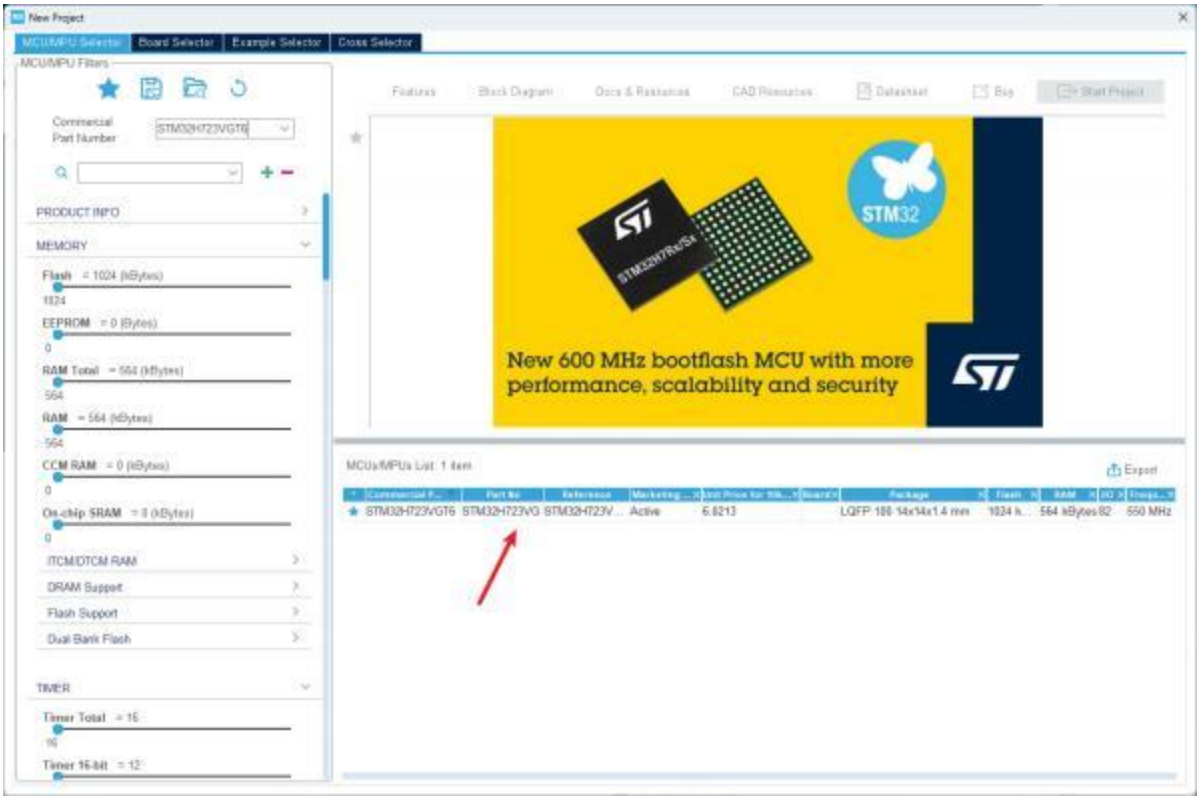
点击File->New project,新建工程



芯片选择界面，在搜索栏里搜索STM32H7，点击STM32H7VGT6

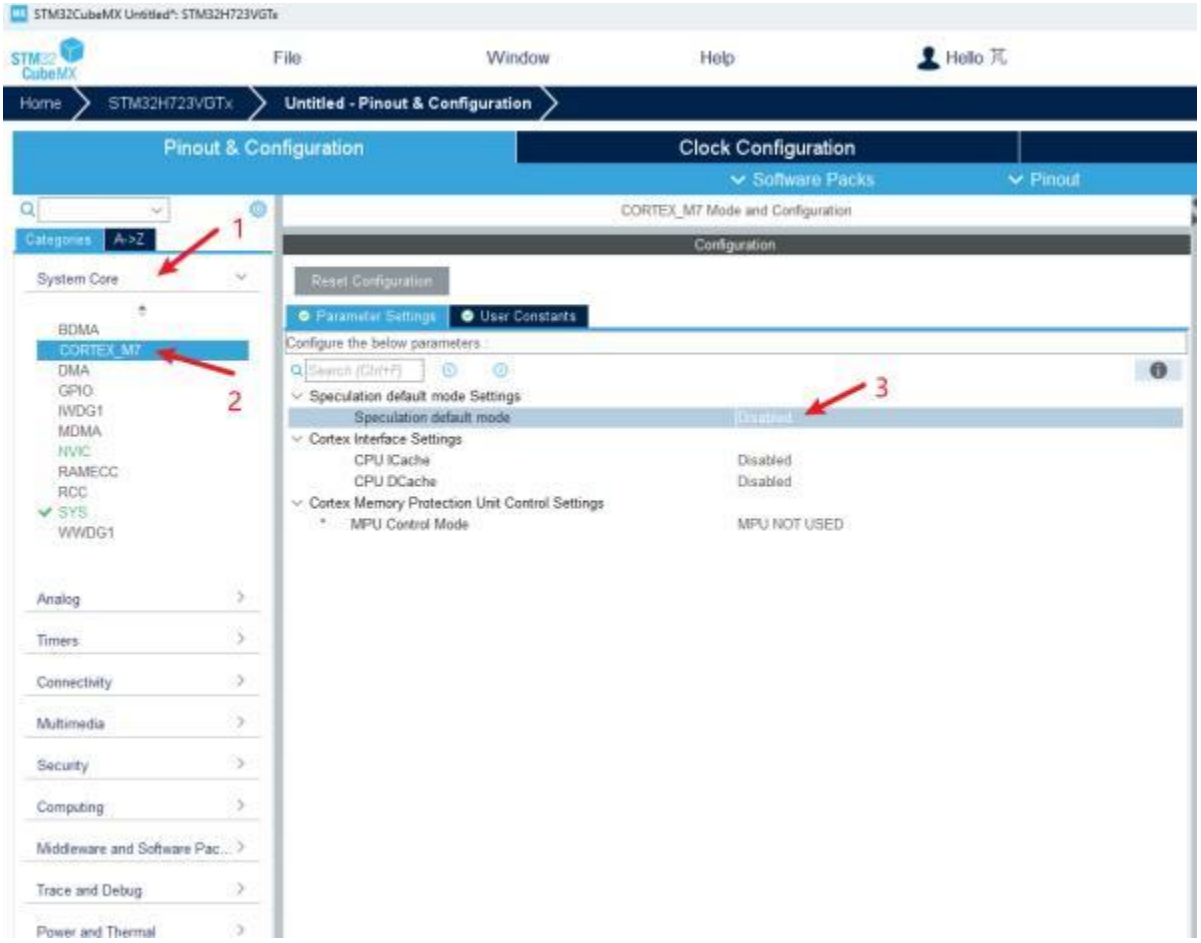


双击箭头所指的地方，打开STM32H7VGT6的工程

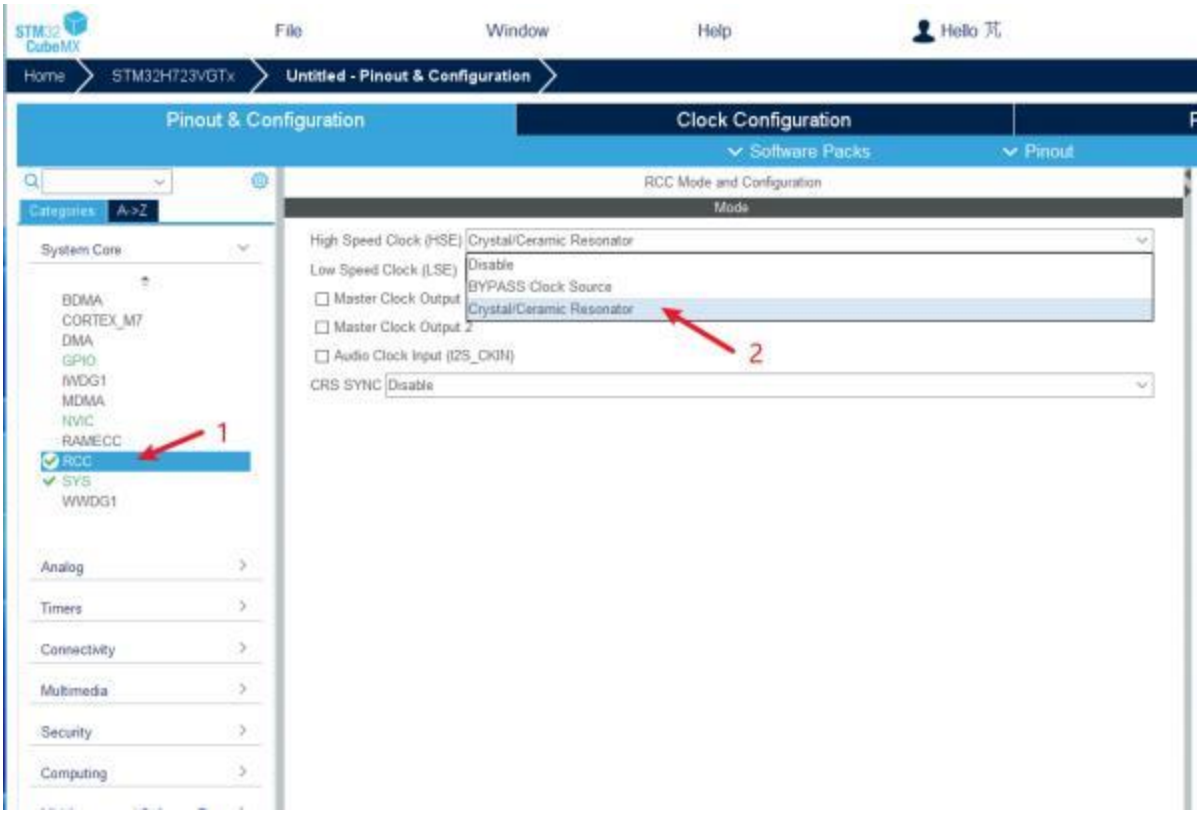


6.3.2 基础配置

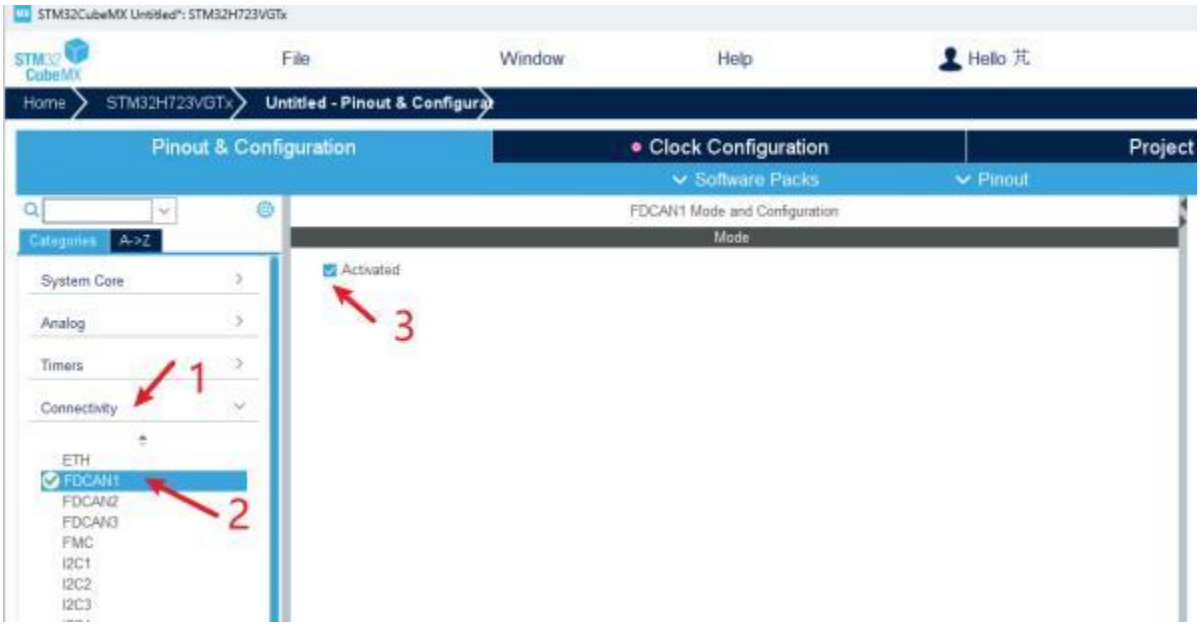
首先点击左侧的**System Core**，再点击**CORTEX_M7**,将 Speculation default mode 修改为Disabled



点击**RCC**，将**High Speed Clock（HSE）**修改为第三个，**Crystal/Ceramic Resontor**



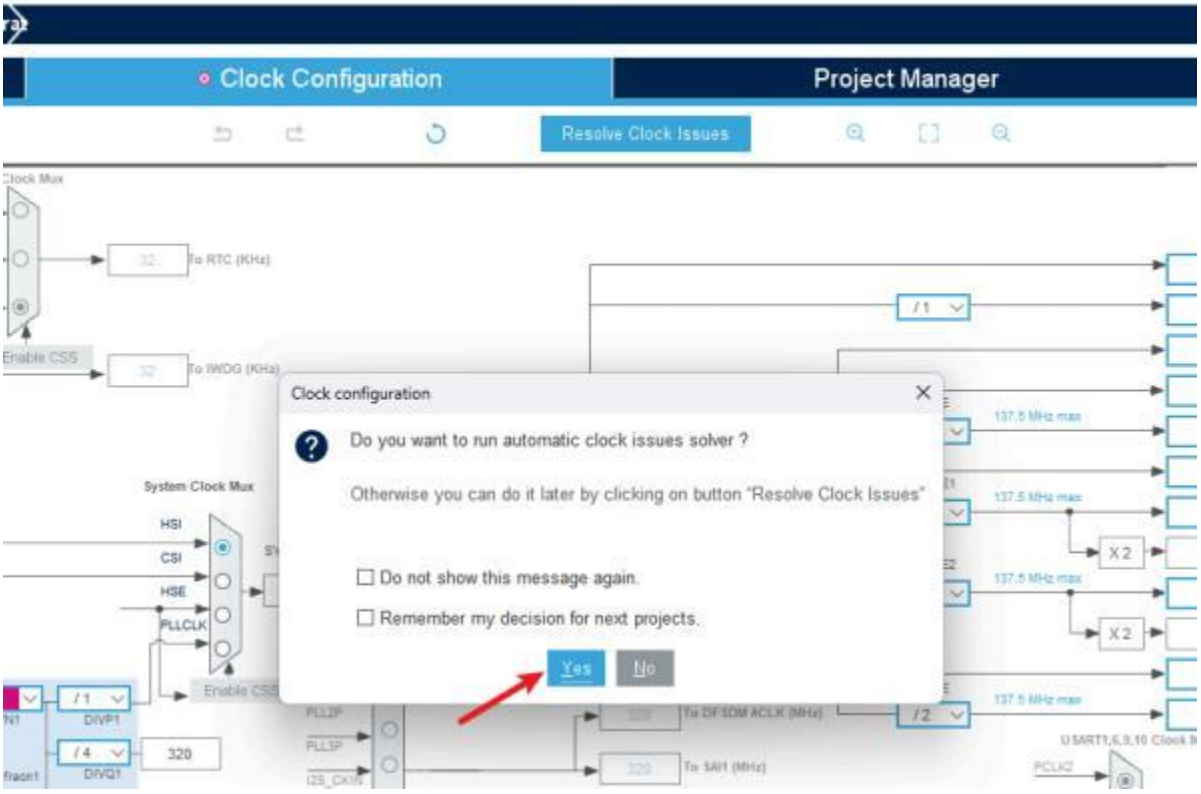
点击**Connectivity**，点击**FDCAN1**，选择**Activated**



6.3.3 时钟配置

1.点击 **Clock Configuration**

若弹出弹窗。点击**yes**

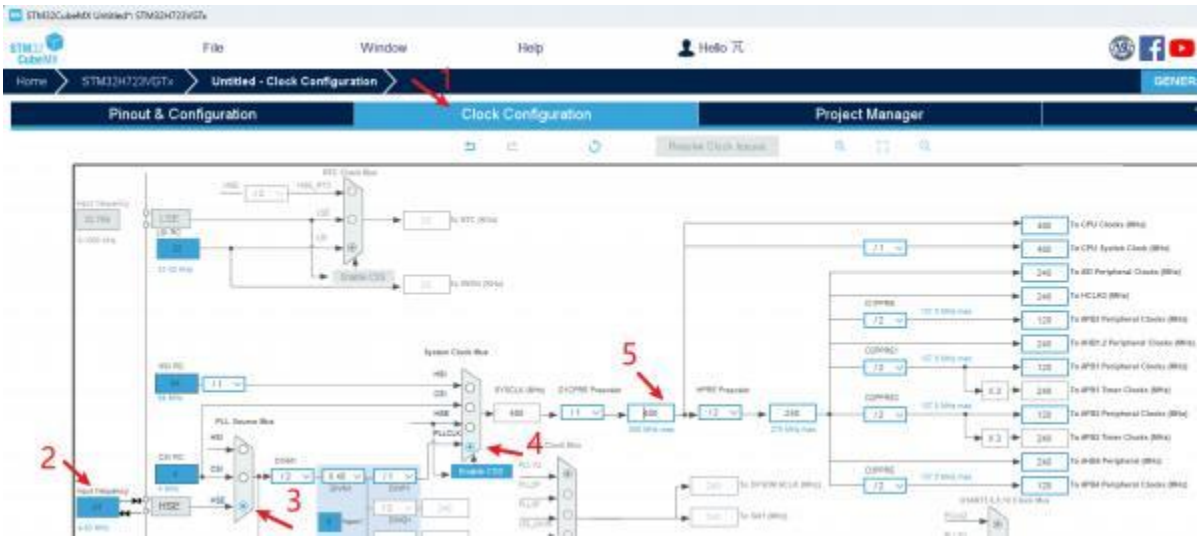


2.将**Input frequency**改为**24**

3.选择**HSE**

4.选择**PLLCLK**

5.修改成**480MHz**

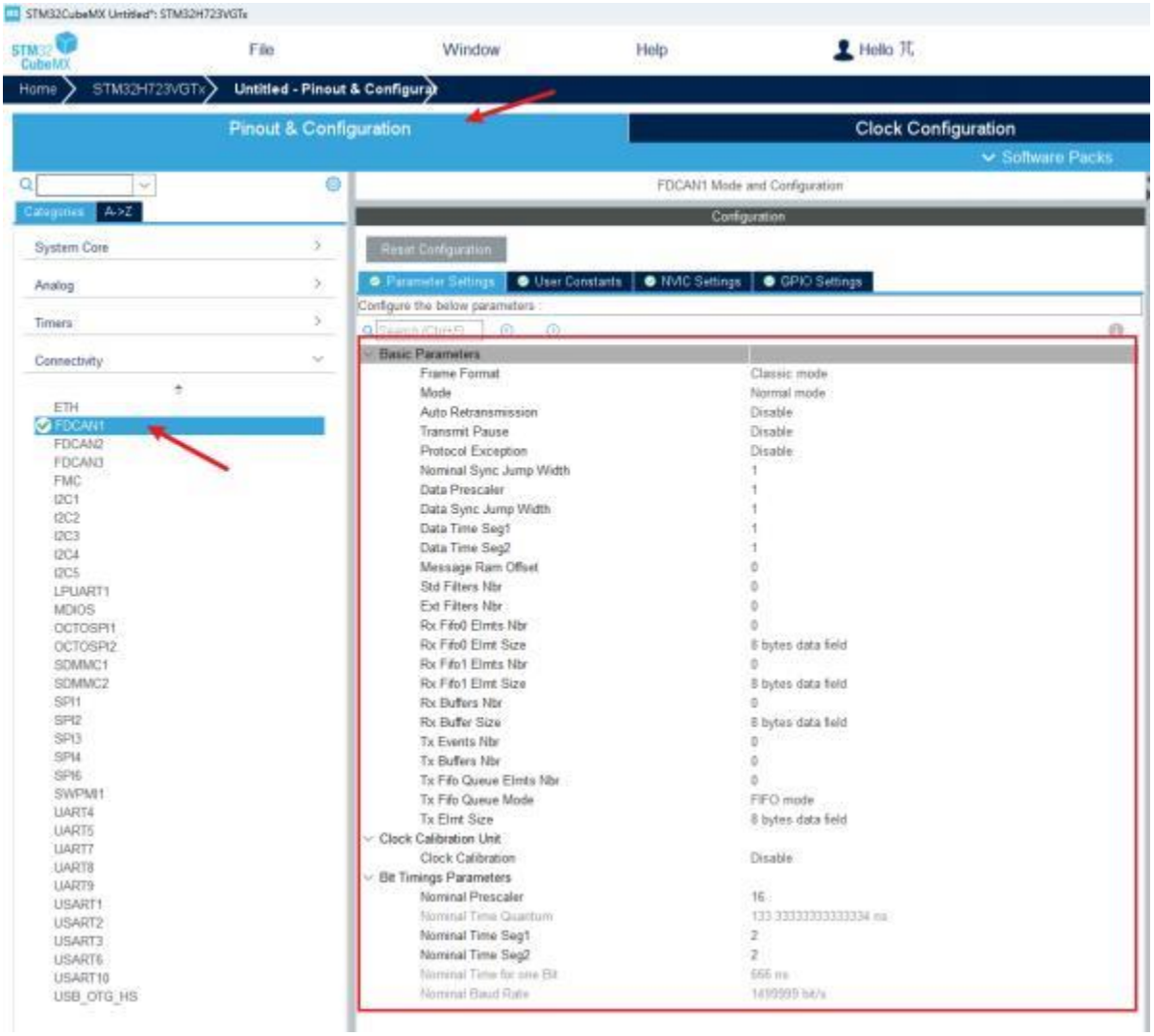


等待软件自动配置完

6.3.4 CAN通信配置

DM_6220电机 采用的经典CAN2.0协议通信，而STM32H7上的外设为FDCAN，为经典CAN的升级版，但可兼容经典CAN模式，接下来将配置成经典CAN模式使用

回到**Pinout&Configuration**， **FDCAN1界面**接下来将配置红框中的部分



接下来只说明需要修改的参数 没有说明的保持默认！！！

Nominal Sync Jump Width: 10

Data Prescaler: 3

Data Sync Jump Width: 10

Data Time Seg1: 29

Data Time Seg2: 10

Std Filters Nbr: 1

Rx Fifo0 Elmts Nbr: 4

Rx Fifo0 Elmt Size: 8 bytes data field

Tx Fifo Queue Elmts Nbr: 4

Tx Fifo Queue Mode: FIFO mode

Tx Elmt Size: 8 bytes data field

Nominal Prescaler: 3

Nominal Time Seg1： 29

Nominal Time Seg2: 10

最后配置成下图所示，可以对照参考一下

Parameter Settings | User Constants | NVIC Settings | GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters	
Frame Format	Classic mode
Mode	Normal mode
Auto Retransmission	Disable
Transmit Pause	Disable
Protocol Exception	Disable
Nominal Sync Jump Width	10
Data Prescaler	3
Data Sync Jump Width	10
Data Time Seg1	29
Data Time Seg2	10
Message Ram Offset	0
Std Filters Nbr	1
Ext Filters Nbr	0
Rx Fifo0 Elmts Nbr	4
Rx Fifo0 Elmt Size	8 bytes data field
Rx Fifo1 Elmts Nbr	0
Rx Fifo1 Elmt Size	8 bytes data field
Rx Buffers Nbr	0
Rx Buffer Size	8 bytes data field
Tx Events Nbr	0
Tx Buffers Nbr	0
Tx Fifo Queue Elmts Nbr	4
Tx Fifo Queue Mode	FIFO mode
Tx Elmt Size	8 bytes data field
Clock Calibration Unit	
Clock Calibration	Disable
Bit Timings Parameters	
Nominal Prescaler	3
Nominal Time Quantum	25.0 ns
Nominal Time Seg1	29
Nominal Time Seg2	10
Nominal Time for one Bit	1000 ns
Nominal Baud Rate	1000000 bit/s

重点看下最后一行 Nominal Baud Rate 是不是为1000000，这是跟电机通信的波特率

点击**NVIC Settings**， **FDCAN1 interrupt 0 点击Enabled**，打开**CAN中断**

Reset Configuration

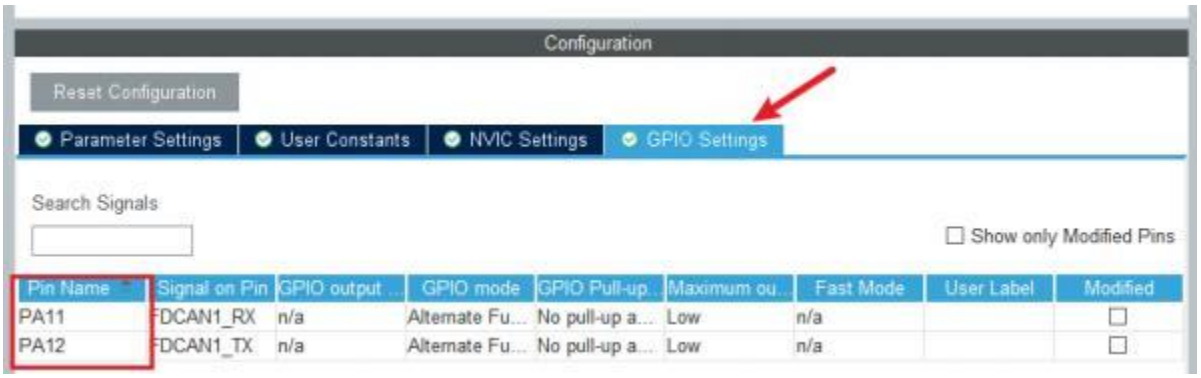
Parameter Settings | User Constants | NVIC Settings | GPIO Settings

1

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
FDCAN1 interrupt 0	☑	0	0
FDCAN1 interrupt 1	☐	0	0
FDCAN calibration unit interrupt	☐	0	0

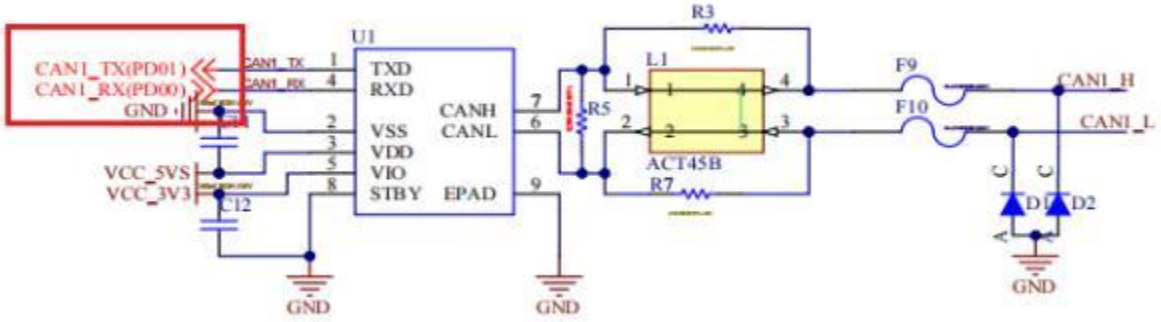
2

点击**GPIO Settings**

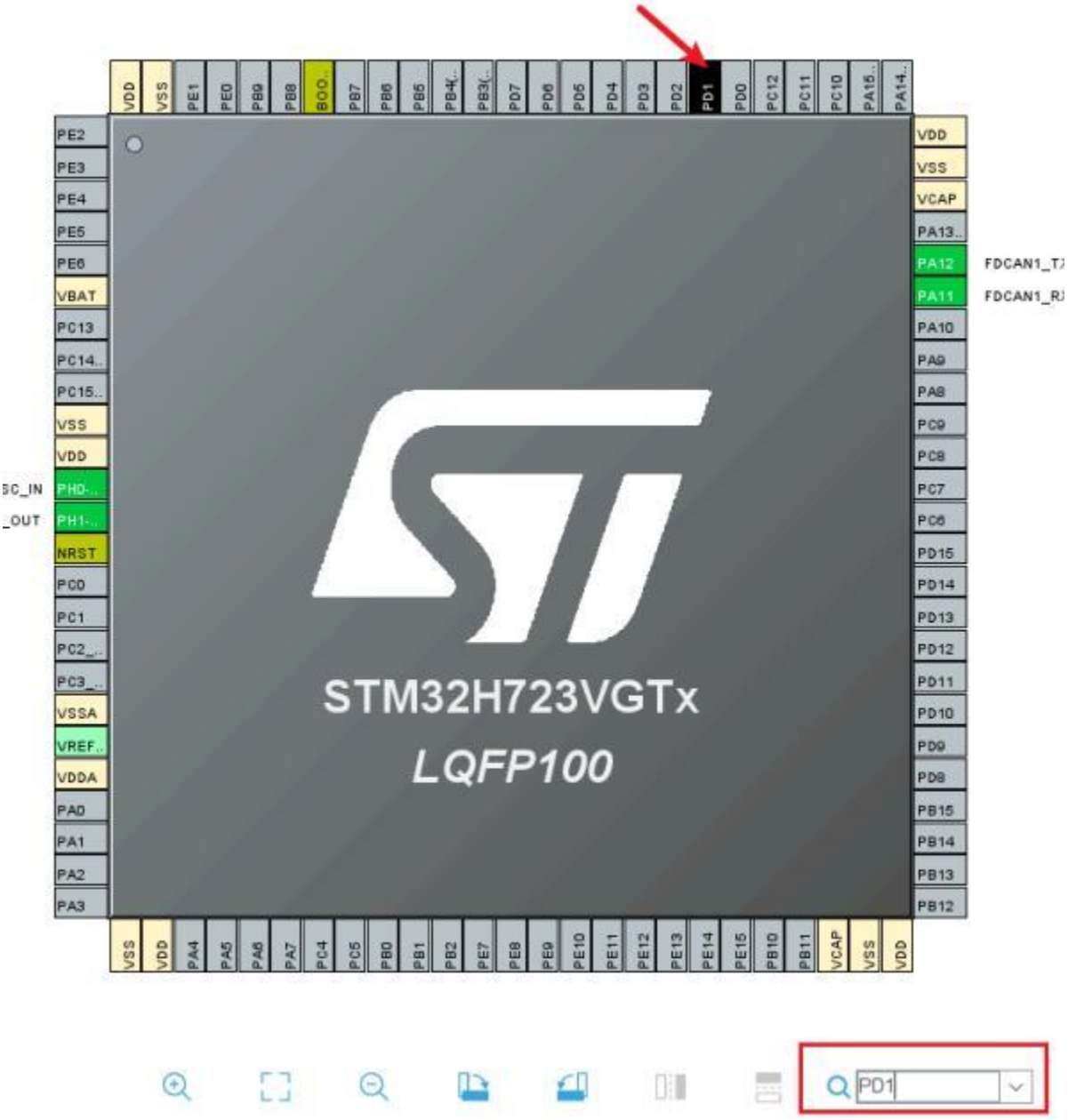


软件自动打开的是PA11、 PA12引脚，翻阅达妙MC_02使用说明书关于FDCAN1引脚连接

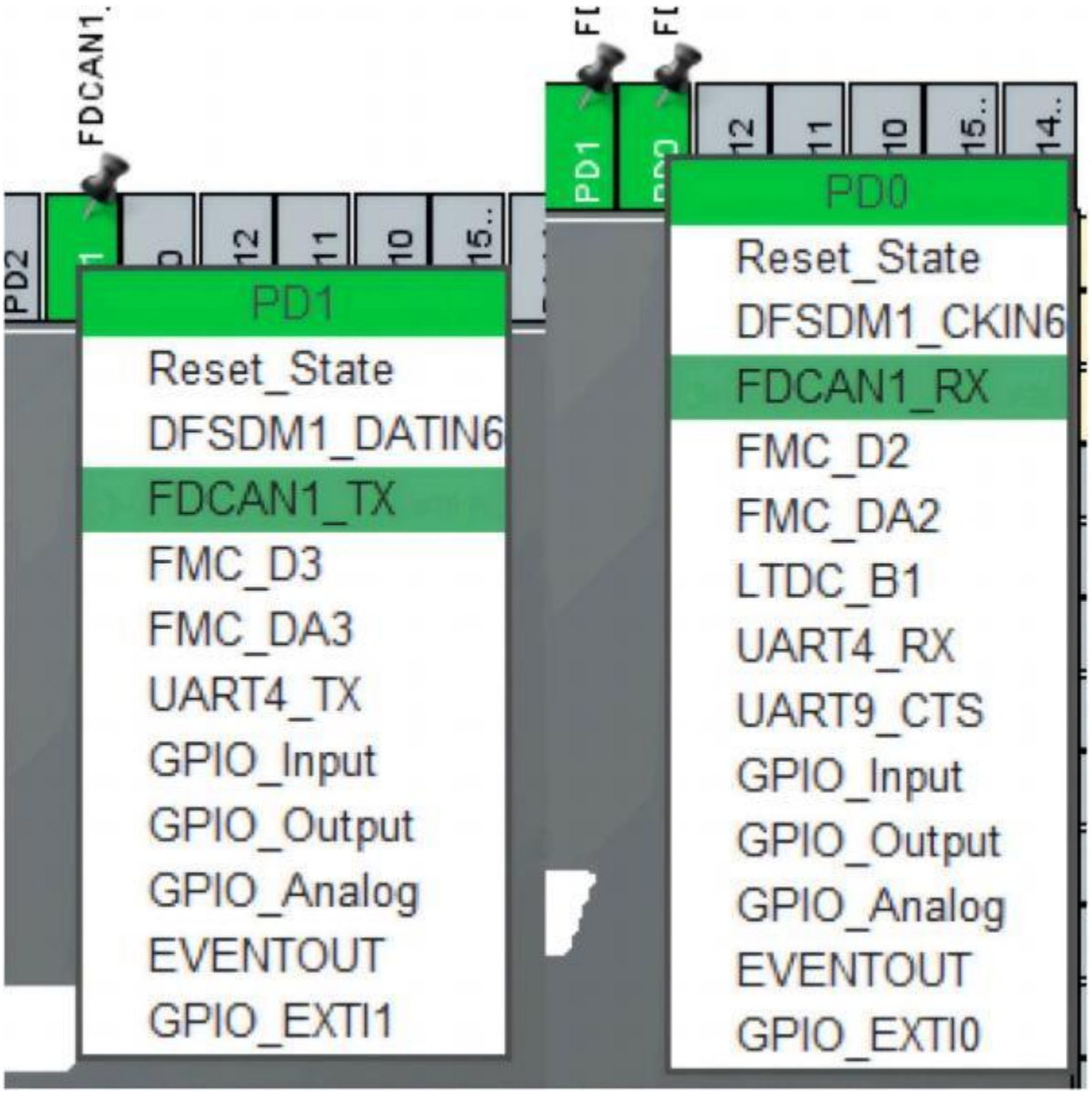
可以看到FDCAN1的RX、TX连接的是**PD01**和**PD02**引脚



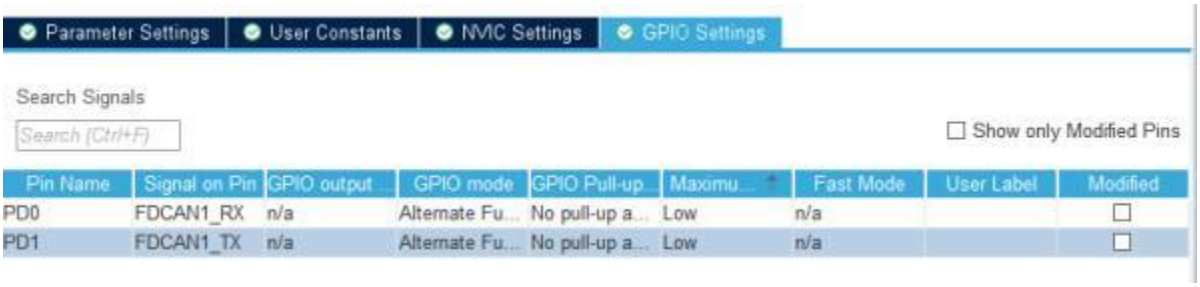
回到芯片界面，在右下角搜索PD1，可以看到PD1在闪烁



将**PD1**和**PD0**改为**FDCAN1_TX**、**FDCAN1_RX**



FDCAN1的GPIO Setting也变为了PD0,PD1



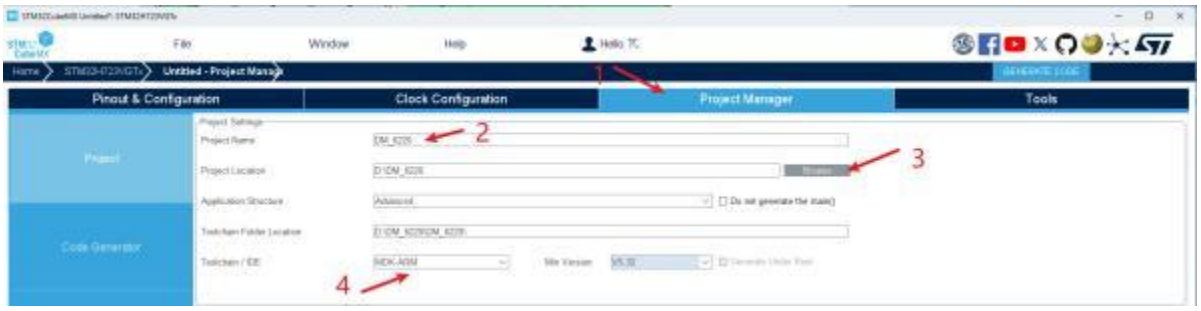
6.3.5 生成MDK工程

1.点击上方的 **Project Manager**

2.输入想要创建的**工程名字**

3.选择生成工程**存放的位置**

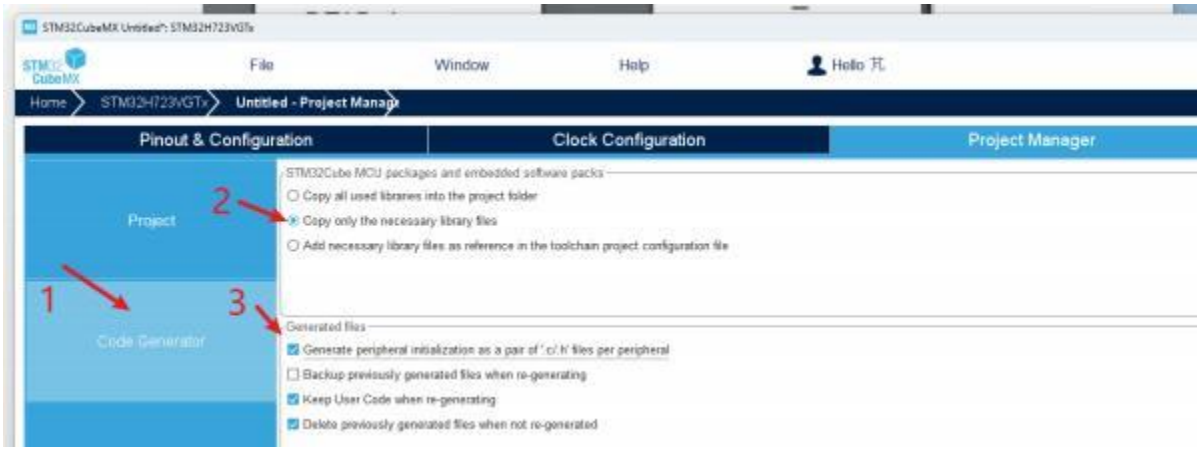
4.选择IDE为**MDK_ARM**



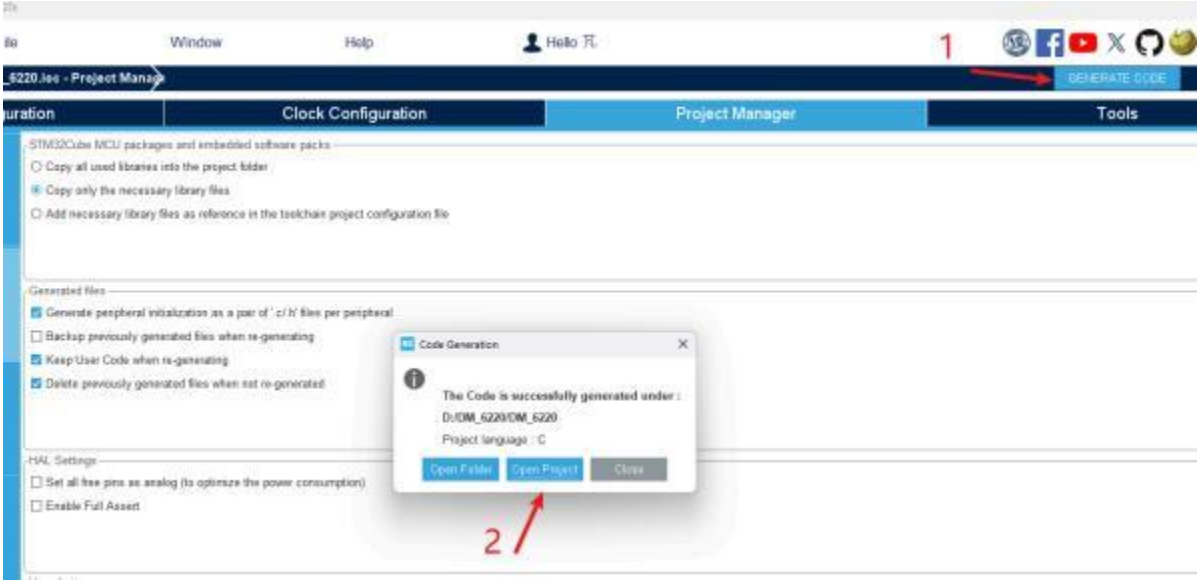
1.点击**Code Generator**

2.勾选 **Copy only necessary library files**

3.勾选 **Generate peripheral initialization as a pair of'c/.h'files per peripheral**



最后点击生成 **GENERATE CODE**生成代码，等待软件自动加载，弹出弹窗后点击 **Open Project**



7 达妙电机控制协议详解与代码

电机完成**校准、标定以及参数设置**后，即可以使用，控制使用 **CAN 标准帧**格式，固定波特率为 **1Mbps**，按功能可分为**接收帧**和**反馈帧**，接收帧为**接收到的控制数据**，用于**实现对电机的命令控制**； **反馈帧**为电机向上层控制器发送**电机的状态数据。**根据电机选定的**不同模式**，其**接收帧帧格式定义以及帧 ID 各不相同**， **但各种模式下的反馈帧是相同的**

7.1 反馈帧

反馈帧 ID 由调试助手设置（Master ID），默认为 0，主要反馈电机的位置，速度和扭矩信息，其帧格式定义为：

反馈报文	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
MST_ID	ID|ERR<<4	POS[15:8]	POS[7:0]	VEL[11:4]	VEL[3:0]|T[11:8]	T[7:0]	T_MOS	T_Rotor

7.1.1 电机ID与ERR状态

反馈帧数据的第一位 **Data[0] = ID | ERR <<4**

ID 表示电机的**CAN ID**，也就是上位机设置的**CAN_ID** ,取 **CAN_ID** 的低 8 位，所以CAN_ID设置超过0x0F （二进制1111）反馈帧将显示不完整

ERR 表示**状态和故障**，其含义如下：

0——失能；

1——使能；

8——超压；

9——欠压；

A——过电流；

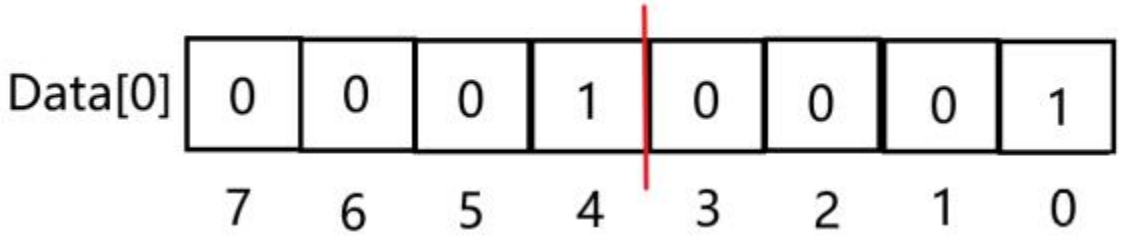
B——MOS 过温；

C——电机线圈过温；

D——通讯丢失；

E——过载；

例如 Data[0] = 0x11 二进制为00010001



0-3位表示电机的ID为0x01 （二进制0001）， 4-7位表示电机状态与故障代码0x01(二进制0001)表示**已经使能**。

所以获取电机ID需要将**Data[0]&0x0F**,即去掉高四位，且保留低四位为1的位。

获取电机ERR直接将**Data[0]前移动四位**，保留高四位。

```
DM_Motor->ID  = Data[0] & 0x0F;
DM_Motor->Data.State = Data[0]>>4;
```

7.1.2 电机位置

D[1]	D[2]
POS[15:8]	POS[7:0]

电机位置由反馈帧数据的**Data[1]和Data[2]**组成。其中D[2]表示位置反馈值无符号16位整型变量的0-7位， D[1]表示8-15位。所以**位置反馈值**为

```
DM_Motor->Data.P_int = (uint16_t)(((Data[1]) <<8) | (Data[2]));
```

将Data[1]整体位移8位到16位的高8位，再与Data[2]进行按位或运算，即保留各自的位组成16位无符号整型变量。

例如Data[1] = 0x7F,Data[2] = 0xFF 则P_int = 0x7FFF

再通过如下函数将 P_int映射成位置的浮点数。

```
static float uint_to_float(int X_int, float X_min, float X_max,
int Bits){
    float span = X_max - X_min;
    float offset = X_min;
    return ((float)X_int)*span/((float)((1<<Bits)-1)) + offset;
}
```

X_int为需要转化的整型数据，以0x7FFF为例 **X_int = 0x7FFF**

X_min、X_max为数据映射的最小最大范围，即4.6.2上位机**控制幅值**中的**P_MAX**,默认为12.5，如在上位机修改了**P_MAX、V_MAX、T_MAX**,转化函数中的**X_min、X_max**也需要修改！！！ **否则数据将会进行等比例放缩**

X_min= -12.5 ; X_max = 12.5

Bits为需要转化数据的位数，为16位 **Bits = 16**

计算过程可参考如下

1. 计算范围跨度：

$$\text{span} = X_max - X_min = 12.5 - (-12.5) = 25$$

2. 计算位数的最大值：

$$(1 << Bits) - 1 = (1 << 16) - 1 = 65536 - 1 = 65535$$

3. 计算无符号整数的浮点映射：

$$\frac{X_int \times \text{span}}{65535} = \frac{32767 \times 25}{65535} = \frac{819175}{65535} \approx 12.499809262$$

4. 加上偏移量 X_min：

$$\text{float_value} = 12.499809262 + (-12.5) = -0.000190737$$

最终结果为：

$$\text{float_value} \approx -0.00019$$

所以0x7FFF表示位置为0左右

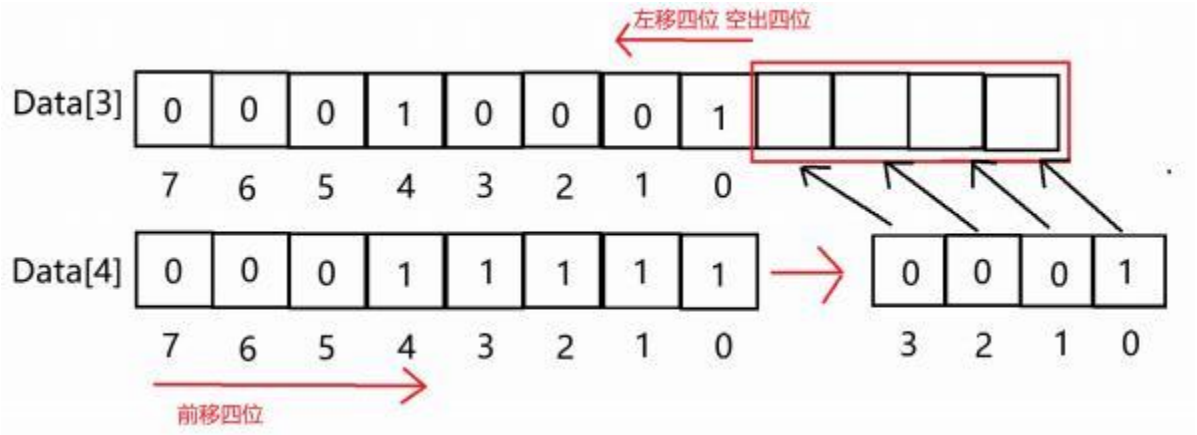
7.1.3 电机速度

D[3]	D[4]
VEL[11:4]	VEL[3:0]|T[11:8]

电机**12位无符号整型变量**的**速度数据**的**后8位**为整个Data[3]， **前4位**由**Data[4]后四位**组成，通过按位或运算组成**12位**数据，但需要用uint16_t来表示。

```
DM_Motor->Data.V_int = (uint16_t)((Data[3]) <<4) | ((Data[4])>>4);
```

移位的逻辑可参考下图



最后通过**uint_to_float**函数转化为浮点数变量，注意映射范围与上位机的**V_MAX**对应， Bits为**12位。**

7.1.4 电机力矩

D[4]	D[5]
VEL[3:0]|T[11:8]	T[7:0]

电机**12位无符号整型变量**的**力矩数据**的**后4位**为**Data[4]的后4位**， **前8位**由**整个Data[4]**组成，通过按位或运算组成**12位**数据，但需要用uint16_t来表示。

```
DM_Motor->Data.T_int = (uint16_t)((Data[4]&0xF) <<8) | ((uint16_t)(Data[5]));
```

最后通过**uint_to_float**函数转化为浮点数变量，注意映射范围与上位机的**T_MAX**对应， Bits为**12位。**

7.1.5 电机温度

D[6]	D[7]
T_MOS	T_Rotor

电机的**T_MOS为Data[6],T_Rotor为Data[7]。**

T_MOS 表示驱动上 MOS 的平均温度，单位℃

T_Rotor 表示电机内部线圈的平均温度，单位℃

```
DM_Motor->Data.Temperature_MOS   = (float)(Data[6]);
DM_Motor->Data.Temperature_Rotor = (float)(Data[7]);
```

7.1.6 电机反馈帧解析函数

整体的电机解析函数为

```
void DM_Motor_Info_Update(uint8_t *Data,DM_Motor_Info_Typedef
*DM_Motor)
{
DM_Motor->ID  = Data[0] & 0x0F;
DM_Motor->Data.State = Data[0]>>4;
DM_Motor->Data.P_int = (uint16_t)(((Data[1]) <<8) | (Data[2]));
DM_Motor->Data.V_int = (uint16_t)((Data[3]) <<4) | ((Data[4])>>4);
DM_Motor->Data.T_int = (uint16_t)((Data[4]&0xF) <<8) | ((uint16_t)
(Data[5]));
DM_Motor->Data.Position=uint_to_float(DM_Motor->Data.P_int,-
P_MAX,P_MAX,16);
DM_Motor->Data.Velocity=uint_to_float(DM_Motor->Data.V_int,-
V_MAX,V_MAX,12);
DM_Motor->Data.Torque=  uint_to_float(DM_Motor->Data.T_int,-
T_MAX,T_MAX,12);
DM_Motor->Data.Temperature_MOS   = (float)(Data[6]);
DM_Motor->Data.Temperature_Rotor = (float)(Data[7]);
}
```

其中**P_MAX**、**V_MAX**、**T_MAX**为宏定义

```
#define P_MAX 12.5f
#define V_MAX 45.f
#define T_MAX 10.f
```

DM_Motor_Info_Typedef *DM_Motor为电机结构体，包含电机ID，电机Master_ID信息，数据结构体

```
typedef struct
{
 uint16_t ID;
  Motor_CANFrameInfo_typedef CANFrameInfo;
  DM_Motor_Data_Typedef Data;
}DM_Motor_Info_Typedef;
```

7.1.7 CAN中断接收电机反馈帧

例程通过CAN的FIFO接收中断回调函数**HAL_FDCAN_RxFifo0Callback**接收电机反馈数据

```
void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan,
uint32_t RxFifo0ITs)
{
HAL_FDCAN_GetRxMessage(hfdcan, FDCAN_RX_FIFO0,
&FDCAN1_RxFrame.Header, FDCAN1_RxFrame.Data);
FDCAN1_RxFifo0RxHandler(&FDCAN1_RxFrame.Header.Identifier,FDCAN1_R
xFrame.Data);

}
```

FDCAN1_RxFrame为CAN接收的结构体

```
typedef struct {
    FDCAN_HandleTypeDef *hcan;
    FDCAN_RxHeaderTypeDef Header;
    uint8_t              Data[8];
} FDCAN_RxFrame_TypeDef;
```

通过**HAL_FDCAN_GetRxMessage**HAL库函数接收反馈信息

最后通过自定义的**FDCAN1_RxFifo0RxHandler**数据处理函数解析反馈数据

```
static void FDCAN1_RxFifo0RxHandler(uint32_t *Master_ID,uint8_t
Data[8])
{

  DM_Motor_Info_Update(Data,&DM_6220_Motor);

}
```

DM_6220_Motor为

```
DM_Motor_Info_Typedef DM_6220_Motor = { //6220电机结构体
  .CANFrameInfo = {
        .CAN_ID = 0x01,
      .Master_ID = 0x00,
     },
};
```

7.2 控制帧

7.2.1 使能、失能、保存零点、清除错误

电机需要发送**使能命令**后，电机**LED由红变绿**之后才可以进行控制， **无论使用哪种模式，使能电机的命令都是一样的**

D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFC

失能电机的命令也一样 最后一个字节改为**0xFD**

“失能”帧属于控制帧，帧 ID 如前所述，数据段定义如下：

D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFD

保存零点，发送该指令后， **电机会将当前位置设置为0点 最后一个字节为0xFC**

“保存位置零点”帧属于控制帧，帧 ID 如前所述，数据段定义如下：

D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFE

清除错误，电机进入错误保护后，如（**过温、过流、过压**）等，修正错误后继续使用，发送清除错误指令**清除错误**

电机出现过热等错误时，发送“清除”命令可以清除错误。“清除”帧属于控制帧，帧 ID 如前所述，数据段定义如下：

D[0]	D[1]	D[3]	D[4]	D[5]	D[6]	D[7]
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFB

函数如下

```
void DM_Motor_Command(FDCAN_TxFrame_TypeDef *TxFrame,uint16_t
CAN_ID,uint8_t CMD){
      TxFrame->Header.Identifier = CAN_ID;
      TxFrame->Data[0] = 0xFF;
      TxFrame->Data[1] = 0xFF;
      TxFrame->Data[2] = 0xFF;
      TxFrame->Data[3] = 0xFF;
      TxFrame->Data[4] = 0xFF;
      TxFrame->Data[5] = 0xFF;
      TxFrame->Data[6] = 0xFF;
      switch(CMD){
        case Motor_Enable  :
```

```
            TxFrame->Data[7] = 0xFC;
        break;
        case Motor_Disable :
            TxFrame->Data[7] = 0xFD;
        break;
        case Motor_Save_Zero_Position :
            TxFrame->Data[7] = 0xFE;
        break;
        case Motor_Clear_Error :
            TxFrame->Data[7] = 0xFB;
            break;
        default:
        break;
    }
   HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame->Header,TxFrame->Data);
}
```

FDCAN_TxFrame_TypeDef *TxFrame为CAN发送的结构体 包含了**CAN的句柄、头结构体和8个字节的数据**

```
typedef struct {
    FDCAN_HandleTypeDef *hcan;
    FDCAN_TxHeaderTypeDef Header;
    uint8_t              Data[8];
}FDCAN_TxFrame_TypeDef;
```

uint16_t CAN_ID为发送给电机的**CAN_ID**，与上位机的**CAN_ID**对应

uint8_t CMD为选择发给电机的命令，通过CMD的不同，修改Data[8]的值。通过枚举的宏定义表示

```
typedef enum{
  Motor_Enable,
  Motor_Disable,
  Motor_Save_Zero_Position,
  Motor_Clear_Error
  DM_Motor_CMD_Type_Num,
}DM_Motor_CMD_e;
```

如要使**CAN_ID**为**0x01**的6220电机的**使能**

DM_Motor_Command(&FDCAN1TxFrame,0x01,Motor_Enable);

其中**FDCAN1TxFrame**为

```
FDCAN_TxFrame_TypeDef FDCAN1TxFrame = {
  .hcan = &hfdcan1,
  .Header.IdType = FDCAN_STANDARD_ID,
  .Header.TxFrameType = FDCAN_DATA_FRAME,
  .Header.DataLength = 8,
  .Header.ErrorStateIndicator =  FDCAN_ESI_ACTIVE,
  .Header.BitRateSwitch = FDCAN_BRS_OFF,
  .Header.FDFormat =  FDCAN_CLASSIC_CAN,
  .Header.TxEventFifoControl =  FDCAN_NO_TX_EVENTS,
  .Header.MessageMarker = 0,
};
```

最后通过**HAL_FDCAN_AddMessageToTxFifoQ**库函数发送控制帧。

7.2.2 MIT模式

控制报文	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
ID	p_des [15:8]	p_des [7:0]	v_des [11:4]	v_des[3:0]‖ Kp[11:8]	Kp [7:0]	Kd [11:4]	Kd[3:0]‖ t_ff[11:8]	t_ff[7:0]

MIT模式的控制帧与反馈帧类似。给定**位置、速度、KP、KD、力矩**五个参数的期望，**通过将浮点数映射成整型数据发送给电机**

帧 ID 等于设定的 **CAN ID** 值，**P_des**：位置给定，**V_des**：速度给定，**Kp**：位置比例系数，**Kd**：位置微分系数，**T_ff**：转矩给定值，各参数**符合上一节的映射关系**，其中 **p_des,v_des,t_ff 的范围可由调试助手进行设定**，Kp 的范围为**[0,500]**，Kd 的范围为**[0,5]**。

7.2.2.1 帧 ID

帧ID 即要发送给目标电机的**CAN_ID**,HAL库中为 **FDCAN_TxHeaderTypeDef**结构体中的**Identifier**。

发送时让它等于电机结构体保存的发送ID

```
TxFrame->Header.Identifier = DM_Motor->CANFrameInfo.CAN_ID;
```

7.2.2.2 Data[0]和Data[1]

D[0]	D[1]
p_des [15:8]	p_des [7:0]

发给电机**Data[0]、Data[1]** 由**期望位置的无符号16位整型变量**的**高八位和低八位**组成

给定的期望为**浮点类型变量**，范围不可超过正负**P_MAX**。通过如下函数转换

```
static int float_to_uint(float x_float, float x_min, float x_max,
int bits){
    float span = x_max - x_min;
    float offset = x_min;
    return (int) ((x_float-offset)*((float)((1<<bits)-1))/span);
}
```

X_float为**需要转化的浮点数据**，以**3.141593**为例 **X_float = 3.141593**

X_min、X_max为数据映射的最小最大范围，即4.6.2上位机**控制幅值**中的**P_MAX**,默认为12.5，如在上位机修改了**P_MAX、V_MAX、T_MAX**,转化函数中的**X_min、X_max**也需要修改！！！ **否则数据将会进行等比例放缩**

X_min= -12.5 ; X_max = 12.5

Bits为需要转化数据的位数，为16位 **Bits = 16**

计算过程参考如下

1. 计算范围跨度：

$$\text{span} = X_max - X_min = 12.5 - (-12.5) = 25$$

2. 计算位数的最大值：

$$(1 << Bits) - 1 = (1 << 16) - 1 = 65536 - 1 = 65535$$

3. 计算浮点数相对于最小值的距离：

$$X_float - X_min = 3.141593 - (-12.5) = 3.141593 + 12.5 = 15.641593$$

4. 将浮点数映射到无符号整数范围：

$$\text{uint_value} = \left(\frac{15.641593 \times 65535}{25}\right)$$

5. 计算结果：

$$\text{uint_value} = \frac{1025055.473455}{25} = 41002.218938$$

最终的整数结果为：

$$\text{uint_value} \approx 41002$$

最后算得的**十进制数**为**41002**转化为**16进制**则为 **0xA02A**

Data[0] = 0xA0、Data[1] = 0x2A

```
Postion_Tmp  =  float_to_uint(Postion,-P_MAX,P_MAX,16)  ;
TxFrame->Data[0] = (uint8_t)(Postion_Tmp>>8);
TxFrame->Data[1] = (uint8_t)(Postion_Tmp);
```

5.2.2.3 Data[2]与Data[3]

D[2]	D[3]
v_des [11:4]	v_des[3:0]| Kp[11:8]

发给电机的**Data[2]**为**期望位置的无符号12位整型变量的高8位。**

Data[3]为**期望位置的无符号12位整型变量的低4位**和**KP 位置的比例系数**的**无符号12位整型变量的高4位**组成。

```
Velocity_Tmp =  float_to_uint(Velocity,-V_MAX,V_MAX,12);
KP_Tmp = float_to_uint(KP,0,500,12);
TxFrame->Data[2] = (uint8_t)(Velocity_Tmp>>4);
TxFrame->Data[3] = (uint8_t)((Velocity_Tmp&0x0F)<<4) |
(KP_Tmp>>8);
```

转化函数中**V_MAX**与**上位机**中设置的**V_MAX**需相同， **KP的映射范围固定为0-500**

7.2.2.4 Data[4]和Data[5]

D[4]	D[5]
Kp [7:0]	Kd [11:4]

发给电机的**Data[4]**为**KP 位置的比例系数**的**无符号12位整型变量**的**低8位。 Data[5]**为**KD位置微分系数**的**无符号12位整型变量**的**高8位**

```
KP_Tmp = float_to_uint(KP,0,500,12);
KD_Tmp = float_to_uint(KD,0,5,12);
TxFrame->Data[4] = (uint8_t)(KP_Tmp);
TxFrame->Data[5] = (uint8_t)(KD_Tmp>>4);
```

KD的映射范围固定为0-5

7.2.2.5 Data[6]和Data[7]

D[6]	D[7]
Kd[3:0]| t_ff[11:8]	t_ff[7:0]

发给电机的**Data[6]**由**KD 位置的微分系数**的**无符号12位整型变量**的**低4位**和**期望力矩**的**无符号12位整型变量**的**高4位**组成。 **Data[7]**为**期望力矩**的**无符号12位整型变量**的**低8位**

```
KD_Tmp = float_to_uint(KD,0,5,12);
Torque_Tmp = float_to_uint(Torque,-T_MAX,T_MAX,12);
TxFrame->Data[6] = (uint8_t)((KD_Tmp&0x0F)<<4) | (Torque_Tmp>>8);
TxFrame->Data[7] = (uint8_t)(Torque_Tmp);
```

转化函数中**T_MAX**与**上位机**中设置的**T_MAX**需相同

7.2.3 位置速度模式

控制报文	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0x100+ID	p_des				v_des			

帧 ID 为设定的 **CAN ID** 值加上 **0x100** 的偏移， **P_des**：位置给定， **浮点型**，低位在前，高位在后， **V_des**：速度给定， **浮点型**，低位在前，高位在后

此处发送命令的 **CAN ID** 是 ***0x100+ID。***

速度给定是梯形加速度运行下最高速度的，即为匀速段的速度值

7.2.3.1 帧ID

位置速度模式下，发送给电机的帧ID需要加上**0x100的偏移**，例如电机的**CAN_ID为0x01**，则通过**CAN总线发送的ID为0x101**

```
TxFrame->Header.Identifier = DM_Motor->CANFrameInfo.CAN_ID +
0x100;
```

7.2.3.2 Data[0]—Data[3]

D[0]	D[1]	D[2]	D[3]
p_des			

发给电机的**Data[0]到Data[3]**,由直接给定的**32位浮点数**期望**位置**拆成**4个无符号8位整型数据**， **低位在前，高位在后**

```
uint8_t *Postion_Tmp
Postion_Tmp = (uint8_t *)&Postion;
TxFrame->Data[0] = *(Postion_Tmp);
TxFrame->Data[1] = *(Postion_Tmp + 1);
TxFrame->Data[2] = *(Postion_Tmp + 2);
TxFrame->Data[3] = *(Postion_Tmp + 3);
```

建立一个**uint8_t * 类型的指针**，通过**取地址**取**期望位置**的**前8位（一字节）**地址， **通过+1、+2、+3**， **获取32位浮点数的第2、3、4字节的地址**，最后将**变量地址转化为整型数据赋值给数组**

7.2.3.3 Data[4]—Data[7]

D[4]	D[5]	D[6]	D[7]
v_des			

发给电机的**Data[4]到Data[7]**,由直接给定的**32位浮点数**期望**速度**拆成**4个无符号8位整型数据**， **低位在前，高位在后**

```
uint8_t *Velocity_Tmp
Velocity_Tmp = (uint8_t *)&Velocity;
TxFrame->Data[4] = *(Velocity_Tmp);
TxFrame->Data[5] = *(Velocity_Tmp + 1);
TxFrame->Data[6] = *(Velocity_Tmp + 2);
TxFrame->Data[7] = *(Velocity_Tmp + 3);
```

建立一个**uint8_t * 类型的指针**，通过**取地址**取**期望位置**的**前8位（一字节）**地址， **通过+1、+2、+3**， **获取32位浮点数的第2、3、4字节的地址**，最后将**变量地址转化为整型数据赋值给数组**

7.2.4 速度模式

控制报文	D[0]	D[1]	D[2]	D[3]
0x200+ID	v_des			

帧 ID 为设定的 CAN ID 值加上 0x200 的偏移， V_des：速度给定，浮点型，低位在前，高位在后，此处发送命令的 CAN ID 是 *0x200+ID*

7.2.4.1 帧ID

位置速度模式下，发送给电机的帧ID需要加上**0x200的偏移**，例如电机的**CAN_ID为0x01**，则通过**CAN总线发送的ID为0x201**

```
TxFrame->Header.Identifier = DM_Motor->CANFrameInfo.CAN_ID +
0x200;
```

7.2.4.2 Data[0]—Data[7]

发给电机的**Data[0]到Data[4]**,由直接给定的**32位浮点数**期望**速度**拆成**4个无符号8位整型数据**， **低位在前，高位在后**， **Data[4]—Data[7]都为0**

```
uint8_t *Velocity_Tmp
Velocity_Tmp = (uint8_t *)&Velocity;
TxFrame->Data[0] = *(Velocity_Tmp);
TxFrame->Data[1] = *(Velocity_Tmp + 1);
TxFrame->Data[2] = *(Velocity_Tmp + 2);
TxFrame->Data[3] = *(Velocity_Tmp + 3);
TxFrame->Data[4] = 0;
TxFrame->Data[5] = 0;
TxFrame->Data[6] = 0;
TxFrame->Data[7] = 0;
```

建立一个**uint8_t * 类型的指针**，通过**取地址**取**期望速度的**前8位（一字节） **地址，**通过+1、+2、+3，获取32位浮点数的第2、3、4字节的地址，**最后将**变量地址转化为**整型数据赋值给数组**

7.2.5 控制帧发送函数

```
void DM_Motor_CAN_TxMessage(FDCAN_TxFrame_TypeDef
*TxFrame,DM_Motor_Info_Typedef *DM_Motor,uint8_t Mode,float
Postion, float Velocity, float KP, float KD, float Torque){

     if(Mode > Velocity_Mode || Mode < MIT_Mode ) Mode = MIT_Mode;

     if(Mode == MIT_Mode) {

             static uint16_t
Postion_Tmp,Velocity_Tmp,Torque_Tmp,KP_Tmp,KD_Tmp;

             Postion_Tmp  =  float_to_uint(Postion,-
P_MAX,P_MAX,16) ;
```

```
            Velocity_Tmp =  float_to_uint(Velocity,-
V_MAX,V_MAX,12);
            Torque_Tmp = float_to_uint(Torque,-T_MAX,T_MAX,12);
            KP_Tmp = float_to_uint(KP,0,500,12);
            KD_Tmp = float_to_uint(KD,0,5,12);

            TxFrame->Header.Identifier = DM_Motor-
>CANFrameInfo.CAN_ID;
            TxFrame->Data[0] = (uint8_t)(Postion_Tmp>>8);
            TxFrame->Data[1] = (uint8_t)(Postion_Tmp);
            TxFrame->Data[2] = (uint8_t)(Velocity_Tmp>>4);
            TxFrame->Data[3] = (uint8_t)((Velocity_Tmp&0x0F)<<4)
| (KP_Tmp>>8);
            TxFrame->Data[4] = (uint8_t)(KP_Tmp);
            TxFrame->Data[5] = (uint8_t)(KD_Tmp>>4);
            TxFrame->Data[6] = (uint8_t)((KD_Tmp&0x0F)<<4) |
(Torque_Tmp>>8);
            TxFrame->Data[7] = (uint8_t)(Torque_Tmp);

            HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame-
>Header,TxFrame->Data);

     }else if(Mode == Position_Velocity_Mode){

            KP = 0; KD = 0; Torque = 0;
            uint8_t *Postion_Tmp,*Velocity_Tmp;
            Postion_Tmp = (uint8_t *)&Postion;
            Velocity_Tmp = (uint8_t *)&Velocity;

            TxFrame->Header.Identifier = DM_Motor-
>CANFrameInfo.CAN_ID + 0x100;
            TxFrame->Data[0] = *(Postion_Tmp);
            TxFrame->Data[1] = *(Postion_Tmp + 1);
            TxFrame->Data[2] = *(Postion_Tmp + 2);
            TxFrame->Data[3] = *(Postion_Tmp + 3);
            TxFrame->Data[4] = *(Velocity_Tmp);
            TxFrame->Data[5] = *(Velocity_Tmp + 1);
            TxFrame->Data[6] = *(Velocity_Tmp + 2);
            TxFrame->Data[7] = *(Velocity_Tmp + 3);

            HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame-
>Header,TxFrame->Data);

     }else if(Mode == Velocity_Mode){
```

```
            Postion = 0;KP = 0; KD = 0; Torque = 0;
            uint8_t *Velocity_Tmp;
            Velocity_Tmp = (uint8_t *)&Velocity;

            TxFrame->Header.Identifier = DM_Motor->CANFrameInfo.CAN_ID + 0x200;
            TxFrame->Data[0] = *(Velocity_Tmp);
            TxFrame->Data[1] = *(Velocity_Tmp + 1);
            TxFrame->Data[2] = *(Velocity_Tmp + 2);
            TxFrame->Data[3] = *(Velocity_Tmp + 3);
            TxFrame->Data[4] = 0;
            TxFrame->Data[5] = 0;
            TxFrame->Data[6] = 0;
            TxFrame->Data[7] = 0;

            HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame->Header,TxFrame->Data);

      }

}
```

void DM_Motor_CAN_TxMessage(FDCAN_TxFrame_TypeDef *TxFrame,DM_Motor_Info_Typedef *DM_Motor,uint8_t Mode,float Postion, float Velocity, float KP, float KD, float Torque)

FDCAN_TxFrame_TypeDef *TxFrame 为 **CAN发送结构体，包含了CAN句柄、CAN头结构体、8字节数据**

DM_Motor_Info_Typedef *DM_Motor 为 **达妙电机结构体， 包含了电机的各项数据**

uint8_t Mode为电机使用的控制模式，通过枚举表示

```
typedef enum{
  MIT_Mode,
  Position_Velocity_Mode,
  Velocity_Mode,
  DM_Motor_Mode_Type_Num,
}DM_Motor_Mode_e;
```

float Postion,float Velocity, float KP, float KD, float Torque为发送给电机的**期望数据， 在非MIT模式下某些用不上的数据会清0**

代码示例:

MIT模式下:

控制电机输出0.2NM的力矩:

DM_Motor_CAN_TxMessage(&FDCAN1TxFrame,&DM_6220_Motor,MIT_Mode,0, 0, 0, 0, 0.2f) ;

控制电机转到到3.141593的位置:

DM_Motor_CAN_TxMessage(&FDCAN1TxFrame,&DM_6220_Motor,MIT_Mode,3.141593f, 0, 3.f, 0.03f, 0) ;

控制电机以5rad/s的转速旋转:

DM_Motor_CAN_TxMessage(&FDCAN1TxFrame,&DM_6220_Motor,MIT_Mode, 0, 5.f, 0, 0.07f, 0) ;

位置速度模式下

控制电机以5rad/s的最大速度转动到3.141593的位置:

DM_Motor_CAN_TxMessage(&FDCAN1TxFrame,&DM_6220_Motor,Position_Velocity_Mode, 3.141593f, 5.f, 0, 0, 0) ;

速度模式下

控制电机以5rad/s的转速旋转:

DM_Motor_CAN_TxMessage(&FDCAN1TxFrame,&DM_6220_Motor,Velocity_Mode, 0, 5.f, 0, 0, 0) ;

7.3 读取、写入、保存电机参数函数

7.3.1 读取电机参数命令

```
void DM_Motor_Read_Param(FDCAN_TxFrame_TypeDef
*TxFrame,DM_Motor_Info_Typedef *DM_Motor,uint8_t RID){

     TxFrame->Header.Identifier = 0x7FF;
     TxFrame->Data[0] = (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID);
     TxFrame->Data[1] = (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID >>
8);
     TxFrame->Data[2] = 0x33;
     TxFrame->Data[3] = RID;
     TxFrame->Data[4] = 0;
     TxFrame->Data[5] = 0;
     TxFrame->Data[6] = 0;
     TxFrame->Data[7] = 0;
```

```
 HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame->Header,TxFrame->Data);
}
```

FDCAN_TxFrame_TypeDef *TxFrame：使用CAN发送的结构体

DM_Motor_Info_Typedef *DM_Motor：发送给哪个CAN_ID的电机

uint8_t RID：读取参数的寄存器地址

例如，读取电机的低压保护值

```
DM_Motor_Read_Param(&FDCAN1TxFrame,&DM_6215_Motor,0);
```

电机反馈值将在FDCAN_RxFrame_TypeDef 结构体中的Data[4]—Data[7]显示

7.3.2 写入电机参数指令

```
void DM_Motor_Write_Param(FDCAN_TxFrame_TypeDef *TxFrame,DM_Motor_Info_Typedef *DM_Motor,uint8_t RID,uint8_t *Write_Param){

     TxFrame->Header.Identifier = 0x7FF;
     TxFrame->Data[0] =  (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID);
     TxFrame->Data[1] =  (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID >> 8);
     TxFrame->Data[2] = 0x55;
     TxFrame->Data[3] = RID;
     TxFrame->Data[4] = Write_Param[0];
     TxFrame->Data[5] = Write_Param[1];
     TxFrame->Data[6] = Write_Param[2];
     TxFrame->Data[7] = Write_Param[3];

  HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame->Header,TxFrame->Data);
}
```

该函数比读取函数多一个uint8_t *Write_Param，为变量的地址。将需要写入的float类型或者uint32类型的变量转化为uint8_t类型长度为4的数组，传入到发给电机CAN通信的Data[4]—Data[7]中

例如，使用union联合体将float转化为4个uint8_t类型

```
static uint8_t float32TObit8(uint8_t i,float change_info)
{
    union{
    float float32;
    uint8_t  byte[4];
    }u32val;
   u32val.float32 = change_info;
   return u32val.byte[i];
}
```

注意Data[4]为最低位、 Data[7]为最高位

7.3.3 保存电机参数指令

```
void DM_Motor_Save_Param(FDCAN_TxFrame_TypeDef
*TxFrame,DM_Motor_Info_Typedef *DM_Motor){

     TxFrame->Header.Identifier = 0x7FF;
     TxFrame->Data[0] = (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID);
     TxFrame->Data[1] = (uint8_t)(DM_Motor->CANFrameInfo.CAN_ID >>
8);
     TxFrame->Data[2] = 0xAA;
     TxFrame->Data[3] = 1;
     TxFrame->Data[4] = 0;
     TxFrame->Data[5] = 0;
     TxFrame->Data[6] = 0;
     TxFrame->Data[7] = 0;

     HAL_FDCAN_AddMessageToTxFifoQ(TxFrame->hcan,&TxFrame-
>Header,TxFrame->Data);

}
```

发送该指令给电机后，电机将保存当前参数的值，掉电不会丢失

8 开源例程

DM_H6215轮毂电机的例程已上传至gitee[DM_6215: 达妙科技DM_H6215轮毂电机控制例程 (gitee.com)]

代码与第7章中保持一致。