SJM8108-08微电机使用说明书

目录

版本记录 **2**

注意事项 **3**

法律声明 **3**

售后政策 **3**

1 电机规格参数 **5**

1.1 图纸及尺寸 5

1.2 电气特性 5

1.3 机械特性 6

2 驱动器信息 **7**

2.1 外观及三维尺寸 7

2.2 接口概览 8

2.3 规格 8

2.4 接口详细定义 9

2.5 主要器件及规格 13

3 调测说明 **14**

3.1 上手指南 14

3.2 固件更新下载 32

4 通信协议及示例 **34**

4.1 CAN 协议 34

4.2 PYTHON SDK 46

4.3 ARDUINO SDK 49

4.4 ROS SDK 56

5 常见问题和异常码（待更新） **57**

5.1 常见问题（FAQ） 57

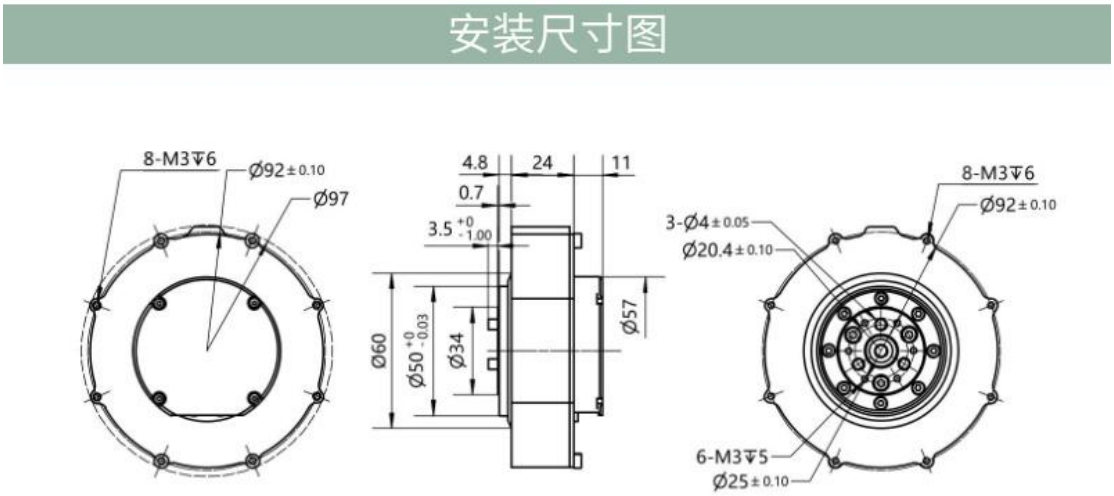
5.2 异常码 57

版本记录

版本号	日期	修订/说明
0.1	2023.12.5	支持 odrivetool 的第一版
0.2	2023.12.15	增加指令列表与指令分类
0.3	2023.12.19	增加 Python，Arduino，ROS SDK
0.4	2023.12.23	增加 USB 和 CAN 兼容性的切换描述
0.5	2023.12.29	增加上位机调测实战案例
0.6	2024.1.2	增加 CAN 协议和 Python 调测实战案例
0.7	2024.1.8	增加 CAN 接口 120R 匹配电阻开关选项
0.8	2024.2.15	增加第二编码器相关内容及用户零点设置
0.9	2024.2.23	增加可修改参数和调用接口函数的 CAN 指令
0.91	2024.3.12	增加国民下载软件的驱动下载地址
0.92	2024.3.22	增加 CAN 默认波特率的说明，以及第二编码器作用下的转子初始位置范围说明。
1.0	2024.3.26	增加电机温度保护说明
1.1	2024.5.20	增加调测上手指南
1.2	2024.5.24	修正 CAN MIT 协议的错误
1.3	2024.7.2	增加异常码表
1.4	2024.8.18	增加修改 ID 的操作说明

1 电机规格参数

1.1 图纸及尺寸

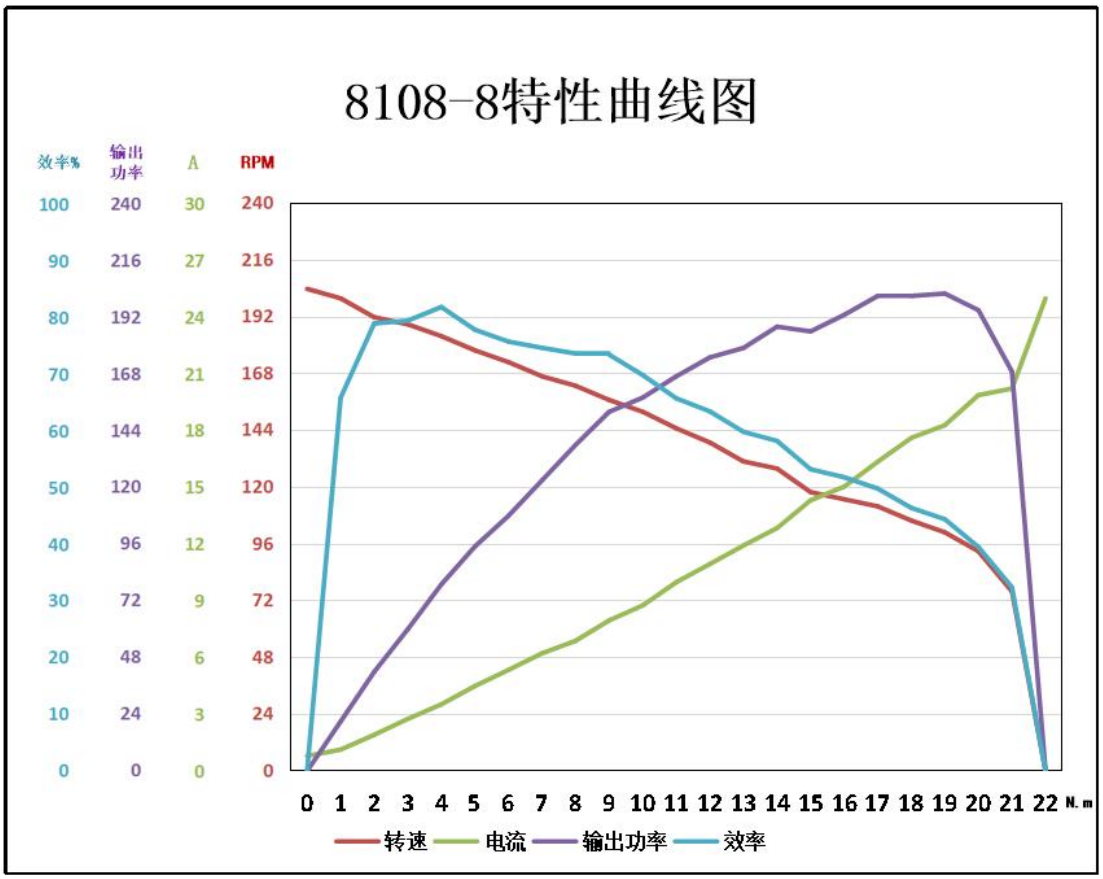


1.2 电气特性

额定转速	110rpm±10%
最大转速	320rpm±10%

额定扭矩	7.5N.m
堵转扭矩	22N.m
额定电流	7A
堵转电流	25A
空载电流	0.4A
相间电阻	0.439Ω±10%
相间电感	403μH±10%
转速常数	5.7rpm/v
扭矩常数	1.00N.M/A

特性曲线如下图：

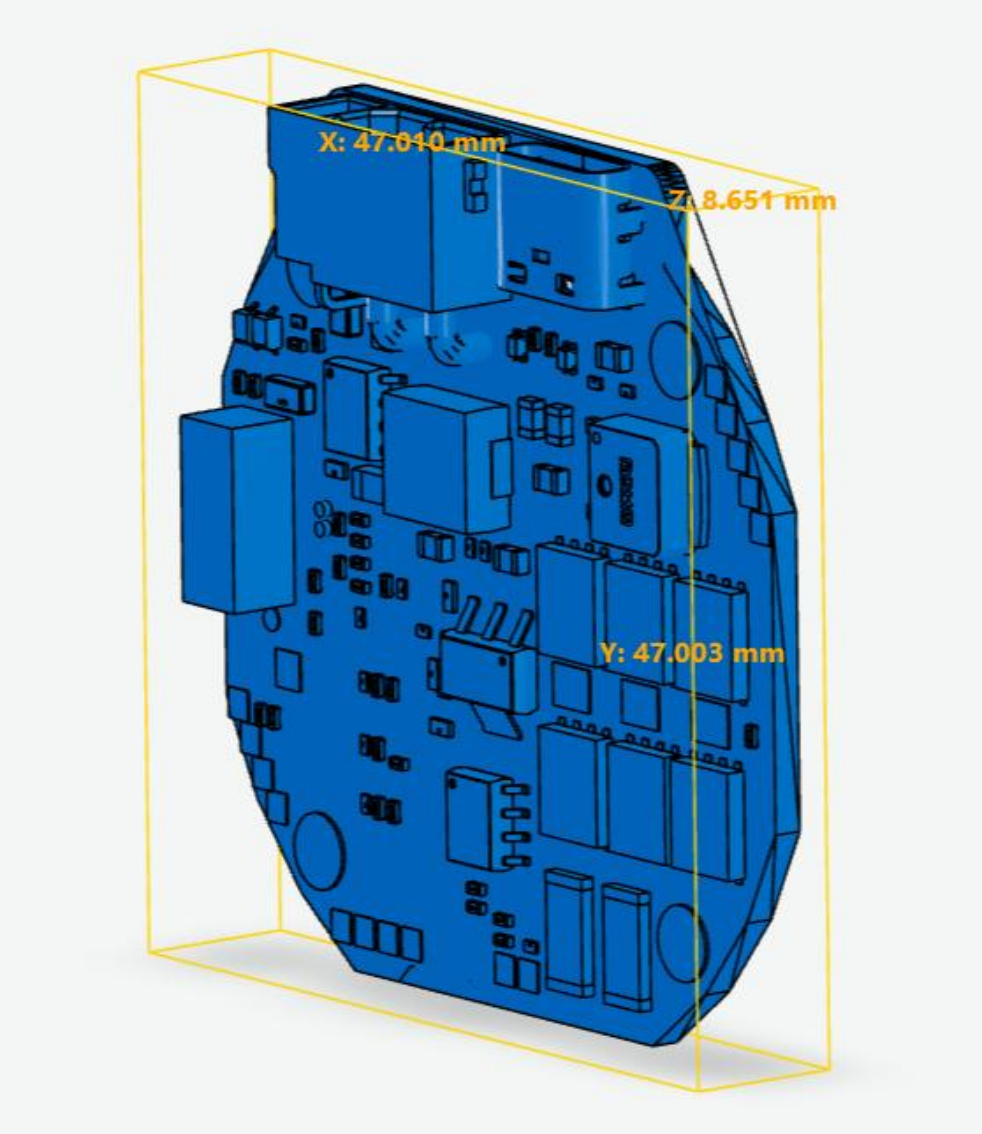


1.3 机械特性

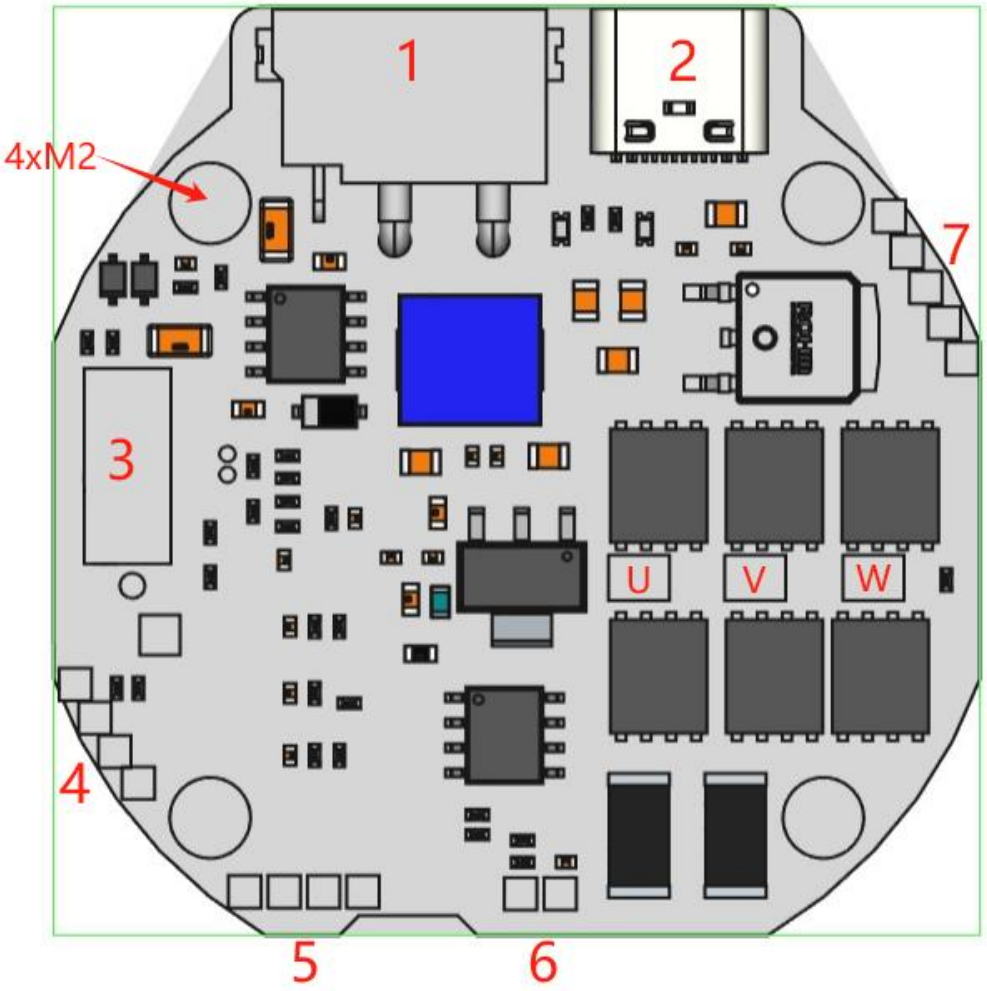
重量	396g±3
极对数	21 对
相数	3 相
驱动方式	FOC
减速比	8:1

2 驱动器信息

2.1 外观及三维尺寸



2.2 接口概览



接口序号	定义
1	15~60V 电源和 CAN 通信集成端子
2	Type-C 调试接口及上位机通信接口
3	接口扩展插槽（可扩展 RS485，EtherCAT，航模，脉冲方向，油门控制等接口/协议）
4	SWD 调试及下载接口
5	第二编码器接口（支持 I2C 和 UART）
6	电机温度接口（NTC）
7	抱闸/刹车电阻接口，12V 电源，最小/最大限位开关接口
U/V/W	三相绕组焊接孔
4xM2	安装孔

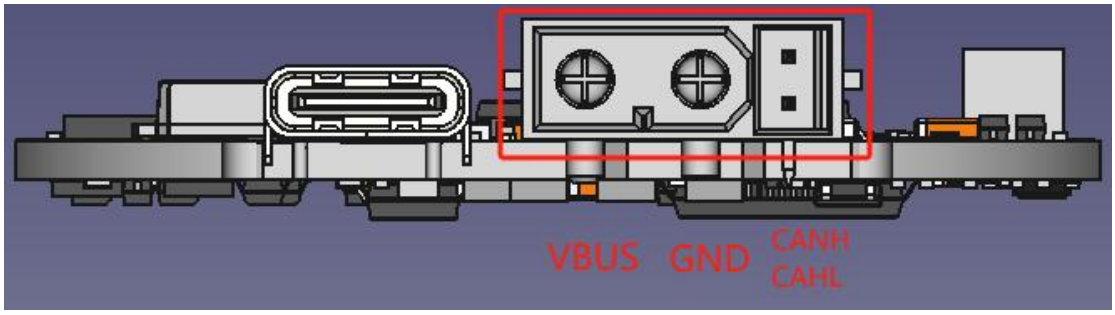
2.3 规格

额定电压	15~48V DC

最小/最大电压	12/72V DC
额定电流	6A
最大线电流	30A
最大相电流	120A
待机功耗	<10mA
CAN 总线最大波特率	1Mbps
Type-C 速率	10Mbps
编码器分辨率	16bit（单圈绝对值）
工作环境温度	-20℃至70℃
告警电机温度	90℃（可调）
告警驱动板温度	90℃（可调）

2.4 接口详细定义

2.4.1 电源及 CAN 通信端子



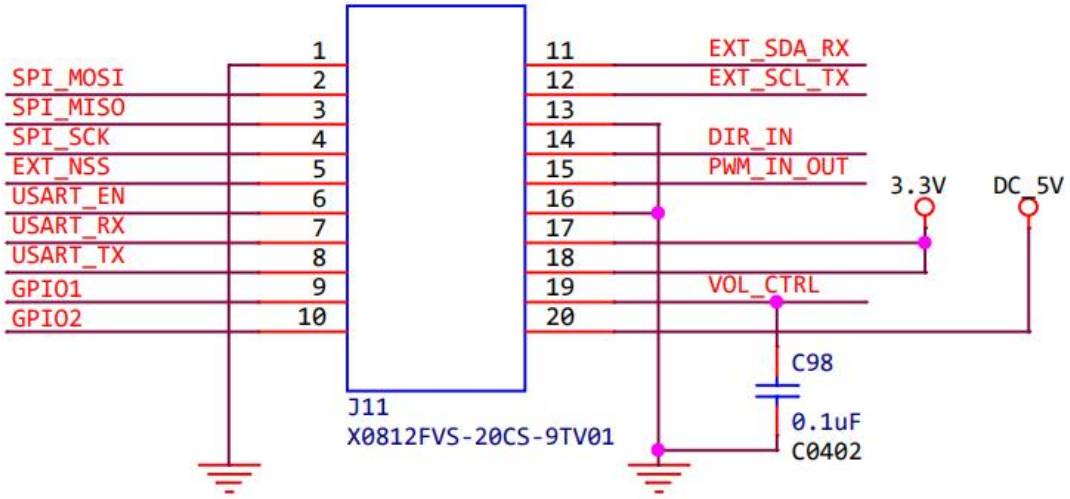
板载端子型号 XT30PB(2+2)-M，线端型号 XT30(2+2)-F，品牌厂家艾迈斯（AMASS）。

2.4.2 Type-C 调试接口

Type-C 采用标准的数据线规格，常用 PC 或手机 Type-C 数据线均兼容。

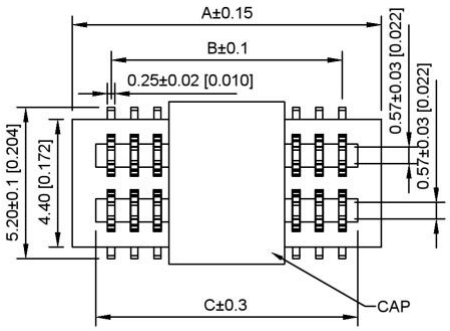
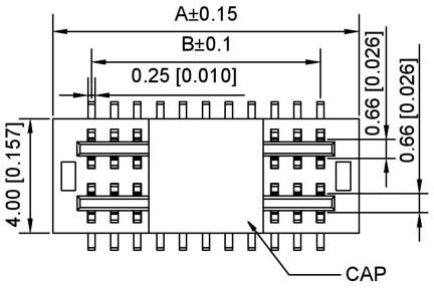
2.4.3 接口扩展插槽

此插槽采用下述设计方式，提供丰富的板间扩展接口，可由第三方开发任意的扩展板:



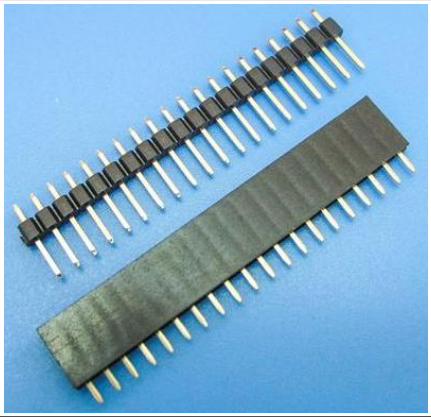
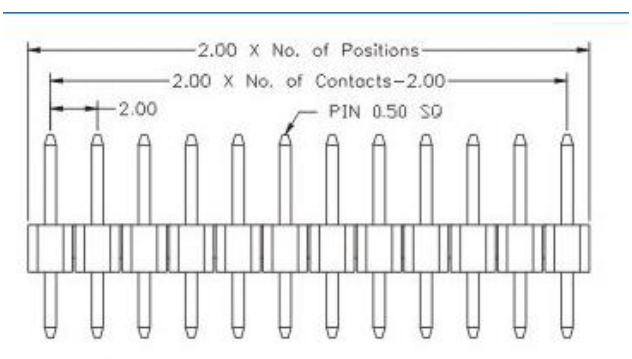
第三方可通过 SPI，USART，I2C，PWM，ADC，GPIO 等方式与驱动器进行交互，实现各种扩展功能。

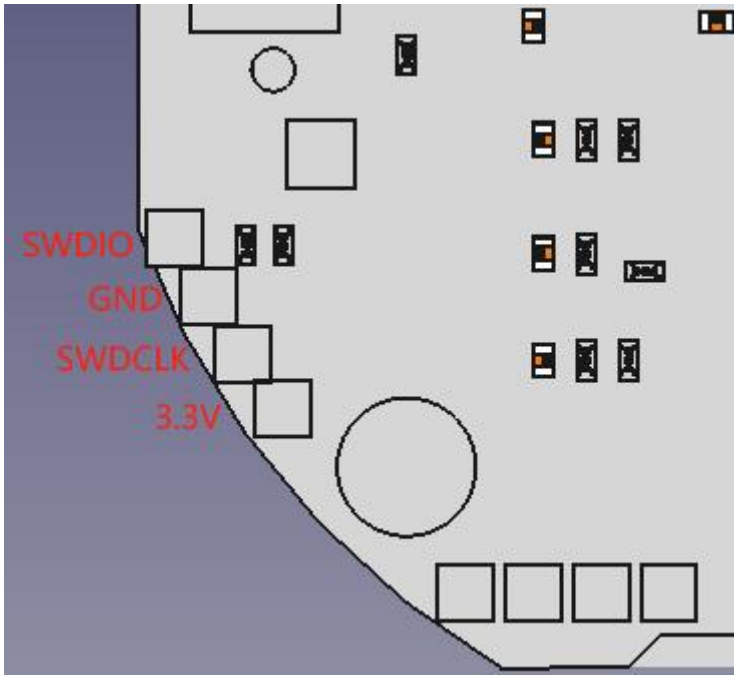
板载插槽型号是 X0812FVS-20CS-9TV01（母座），扩展板插槽型号是 X0812WVS-20AS-9TV01（公座），品牌厂家是星坤。

母座 X0812FVS-20CS-9TV01	公座 X0812WVS-20AS-9TV01
	

2.4.4 SWD 调试接口

间隔 2mm 的插针孔，用户可焊接 2mm 直插单排针，如下图：

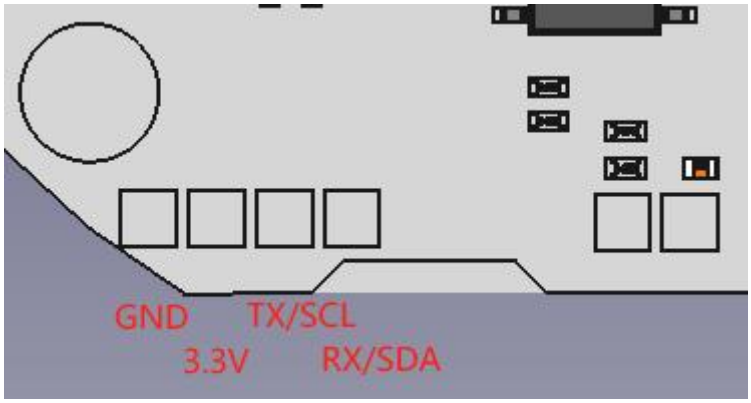
	



2.4.5 第二编码器接口

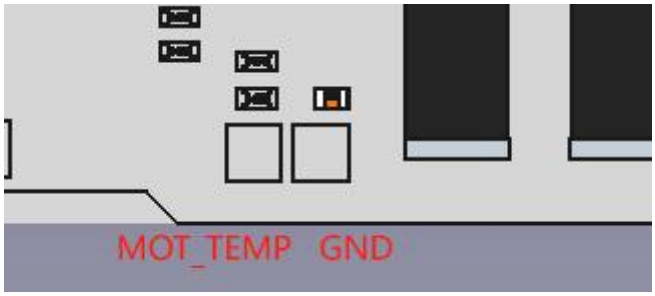
间隔 2mm 的插针孔，用户可焊接 2mm 直插单排针，请参见 2.4.4。

此接口可通过 USART（TX/RX）或 I2C（SCL/SDA）与第二编码器进行通信。



2.4.6 电机温度接口

电机内置 10K 的 NTC 电阻，两条引线焊接到 MOT_TEMP 和 GND，没有线序。

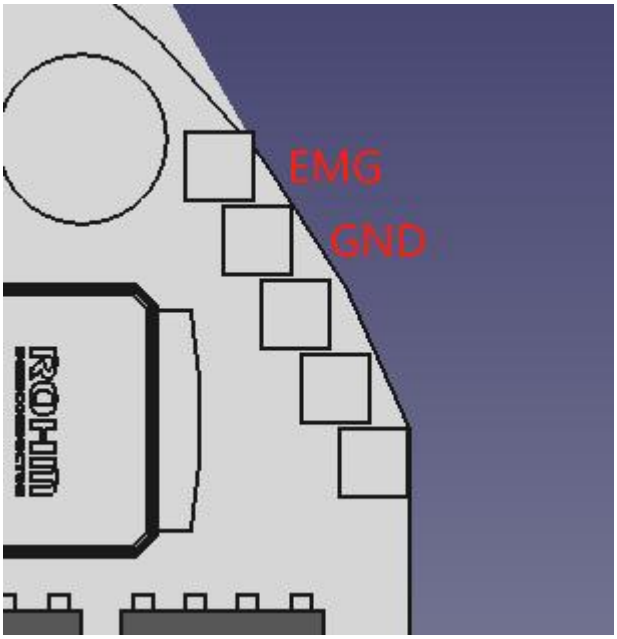


2.4.7 抱闸/刹车电阻接口

图示 5 针接口中上方两个焊孔为抱闸/刹车电阻接口，间隔 2mm 的插针孔，用户可焊接 2mm 直插单排针，请参见 2.4.4。

当为抱闸接口时，驱动器在加电时持续向此接口输出一个电流，以使抱闸呈打开状态，电机得以正常运转，如驱动器断电，则此电流停止，抱闸锁住，电机将在断电位置锁死。

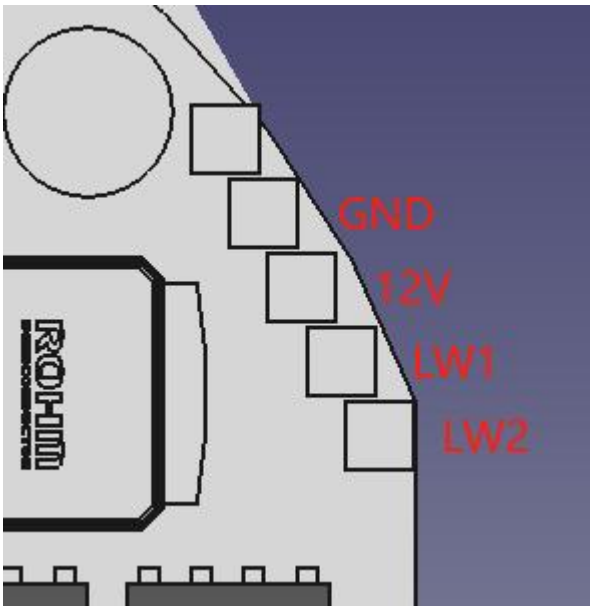
当为刹车电阻接口时，可外接一个刹车电阻（或称之为泄放电阻），在反电动势高于门限电压时，通过此刹车电阻泄放电流，防止无法紧急刹车，或反电动势损坏驱动器。



2.4.8 限位开关接口

驱动器提供两个限位开关接口，且为外部限位开关提供 12V 电源，间隔 2mm 的插针孔，用户可焊接 2mm 直插单排针，请参见 2.4.4。

其中 LW1 是最小位置限位开关，LW2 是最大位置限位开关。可外接两线式开关或三线式 NPN 开关。



2.5 主要器件及规格

序号	器件	型号/规格	数量
1	MCU	N32G455REL7	1
2	驱动芯片	FD6288Q	1
3	磁编码器芯片	MA600，16bit 绝对值	1
4	MOSFET	JMSH1004NG，100V/120A	6

3 调测说明

3.1 上手指南

3.1.1 准备工作

要让电机工作起来，你需要：

✓ 电源

请参见第 1 章对于电源电压的要求，推荐用稳压电源或电池。经常用户会疑惑的问题是，我该怎么选择电源呢？下面是一些简单的建议，仅作参考：

选择电源的几个点：

◆ 电流需求

一般来说，至少要大于 5A。具体的数值取决于系统的功率需求和电压。

◆ 电压需求

电压需求取决于两个因素：电机的 Kv 和系统所需的最大转速 RPMmax，所需电源电压最大值可参考公式：

$$V_{max} = \frac{RPM_{max}}{K_v} \times 1.25$$

其中 1.25 是一个经验系数，给系统一个安全的电压门限。

◆ 功率需求

对于功率，简单的说，取决于最高转速时的最大电流值 Imax，可参考公式：

$$P_{max} = I_{max} \times \frac{RPM_{max}}{K_v} \times 1.25$$

✓ 电源+通信接口线缆

8108-8 带 2+2 电源通信插座，请联系售后推荐合适的 2+2 线缆。请参考 2.4.1，务必接对电源正负极，否则会有烧毁驱动器的危险，因为驱动器没有防反接的能力。同时，请注意 2 个通信线的定义顺序，如 CANH/CANL 的顺序，接错会导致通信不正常。

警告

✧ 请务必避免用手触碰通信总线，避免静电打坏驱动器的接口芯片，特别是干燥区域及干燥季节！

✧ 请务必不要带电拔插电源端子！

✧ 避免使用空开作为电源的开关，有击毁驱动器上电源芯片的风险！

✧ 电源电压不要超过 72V！

✓ Type-C 数据线

在调测的早期，强烈建议用 USB Type-C 数据线来测试电机。可以使用最常用的手机 Type-C 数据线即可，不可使用仅充电的 Type-C 线。

请注意，Type-C 数据线并不能给驱动器供电，更不能驱动电机转动！

✓ 加电

请先关闭电源，连接电源线缆，再打开电源，务必不要带电拔插，或用空开控制单正极或负极线缆进行开关，这样会导致过大的开机电流烧毁驱动器。

在加电后，可随时插拔 USB Type-C 数据线。

3.1.2 用 odrivetool 开始

驱动器兼容 odrive（https://github.com/odriverobotics/odrive.git），所以使用 odrivetool 作为上位机进行调测。

按照下述步骤安装 odrivetool：

➢ **Windows**

1. 安装 python

进入 python 官方网站 https://www.python.org 下载最新 python 安装程序并按提示安装。请不要下载第三方网站或微软 Microsoft Store 商店中的 python 版本。

2. 安装 visual c++生成工具

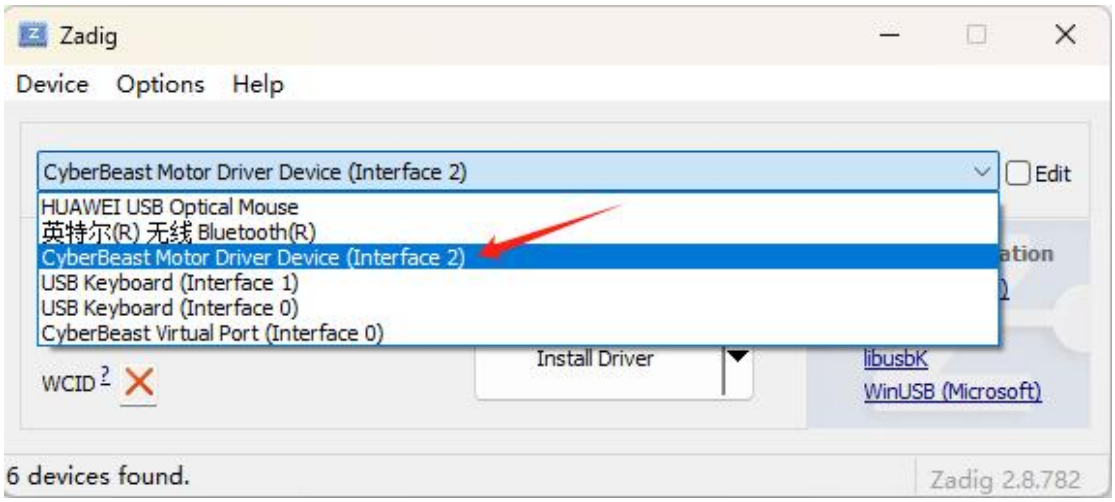
安装 visual c++生成工具 https://visualstudio.microsoft.com/visual-cpp-build-tools/ ，安装过程中勾选“使用 C++的桌面开发”，如下图所示。

3. 安装 odrivetool

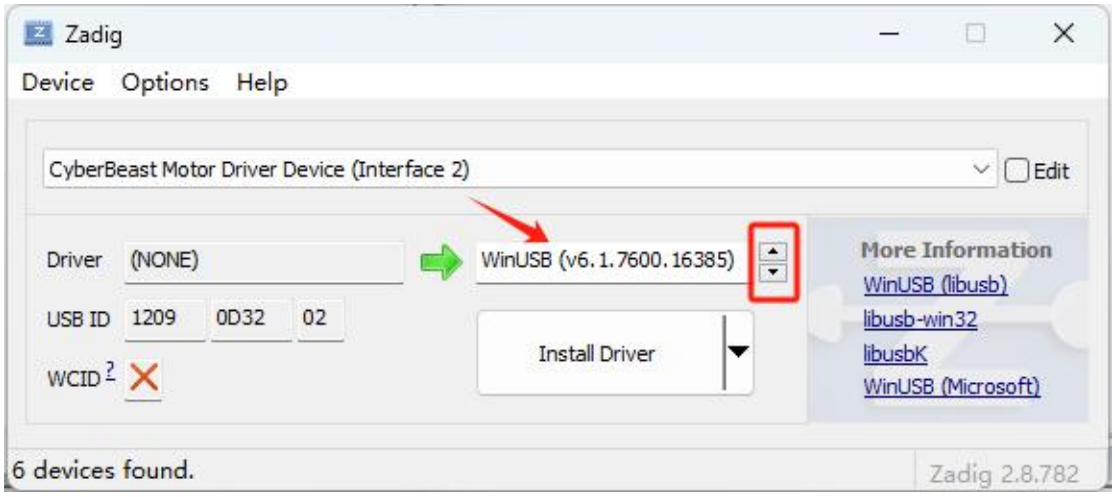
使用管理员运行 Windows PowerShell，在其中运行 pip install odrive 并回车安装。如果中途出错请重试。如果再三出错，请重启电脑再重试。

4. 安装 USB 驱动

进入网站 https://zadig.akeo.ie 并下载 USB 驱动工具 Zadig，用 Type-C 数据线将驱动器接入电脑，此时驱动器电源灯亮，打开 Zadig，通过下拉框选择“CyberBeast Motor Driver Device（Interface 2）”：



通过点击上下键选择不同的 USB 驱动，请为此接口选择“WinUSB”驱动版本，并点击“Install Driver”为这个接口安装驱动：



- **WSL（Windows Subsystem for Linux）**

1) 安装 python/usb/odrivetool

如果用户安装了 WSL2，可进入 WSL2 的命令行，按照下面的步骤来安装（假设用户 WSL 安装的是 Ubuntu）：

```
sudo apt install python3 python3-pip
sudo apt install libusb-1.0-0
sudo pip install odrive numpy matplotlib
```

第一行指令安装 python，第二行指令安装 usb 驱动，第三行指令安装 odrivetool 上位机。

2) 连接驱动器到 WSL

用 type-C 数据线插入 Windows，默认情况下 Windows 会加载此 USB 端口的驱动，而 WSL 并不会加载。需要将此 USB 端口加载到 WSL 中，请参照微软文档（https://learn.microsoft.com/zh-cn/windows/wsl/connect-usb）操作。

➢ **Ubuntu**

Ubuntu 下的安装过程跟 WSL 下非常类似，请参见上一小节。

3.1.3 合理配置电机参数

警告

建议认真仔细阅读这一节，这是成功运转的关键，且避免把电机烧毁！

按照上一节内容成功安装 odrivetool 后，电机加电且连接 USB Type-C 数据线后，在 shell（Windows PowerShell 或 Linux Terminal）中运行 odrivetool（键入 odrivetool 并回车），下图显示 Windows 下连接成功（绿色字体显示连接信息）：

指令示例：odrv0.axis0.controller.input_vel，odrv0 代表当前连接的电机，默认第一个连接的电机叫 odrv0，第二个叫 odrv1，以此类推；axis0 代表驱动器连接的第一个电机，目前的版本中只支持连接一个电机。这个指令的意思是查询驱动器当前速度控制目标值。

操作提示

- ◆ 经常使用 TAB 键，会有指令提示，类似于 linux 下的指令提示，可用上下左右键选择指令；
- ◆ 上下键会显示历史指令
- ◆ 在指令输入时，会提示历史类似指令，使用右键会直接补全

➢ **设置关键门限（limits）**

```
odrv0.axis0.motor.config.current_lim = 30
```

- 电流门限

上述指令将电流门限设置为 30A。请注意，此电流门限是指 Q 轴电流，而不是电源电流。这个门限值直接限制了输出扭矩。对于 8108-8，请不要将此门限设置超过 50A!

其他影响电流门限的因素
◆ 电机温度 如果使能电机温度保护（odrv0.axis0.motor.motor_thermistor.config.enabled=1），则电机的当前温度也会影响 Q 轴电流。
◆ 驱动板温度 如果使能驱动板温度保护（odrv0.axis0.motor.fet_thermistor.config.enabled=1），则驱动板的当前温度也会影响 Q 轴电流。
上述两个温度的影响非常类似，可以用下述公式表示： $$I'_{lim} = \frac{T - T_l}{T_u - T_l} \times I_{lim}$$ 其中I'_{lim}是最终生效的电流门限，I_{lim}是配置的电流门限，T是当前温度（电机温度或驱动板温度），T_l是设置的温度下限（motor_thermistor.config.temp_limit_lower 和 fet_thermistor.config.temp_limit_lower），T_u是设置的温度上限（motor_thermistor.temp_limit_upper 和 fet_thermistor.config.temp_limit_upper）。

```
odrv0.axis0.controller.config.vel_limit = 30
```

- 速度门限

系统全局速度限制，上述指令将其限制为 30turn/s（转/秒）。请注意，默认情况下纯力矩模式下此速度限制不起作用，但可以打开下述开关来使能：

```
odrv0.axis0.controller.config.enable_torque_mode_vel_limit = 1
```

- 校准电流

此电流默认是 5A，默认情况下不用修改，但如果用户的电源电流较小，可将此值 减小，否

```
odrv0.axis0.motor.config.calibration_current = 2
```

则在校准时会出现低压告警。

➢ **设置关键硬件参数**

- **最大放电/充电电流**

放电电流是指电源供给驱动器和电机的正向电流，充电电流是指反向流入电源的电流。这两个值跟电源有关，请设置为一个合适的值，避免电源无法放电导致电压被拉低，或者被反电动势

```
odrv0.config.dc_max_negative_current
odrv0.config.dc_max_positive_current
```

损坏。但请注意，如果将这两个值设置为一个绝对值较小的值，则很容易产生告警。

- **极对数**

极对数是电机转子中磁极数量除以 2。用户必须正确设置这个值，才能校准成功，否则将会

```
odrv0.axis0.motor.config.pole_pairs
```

有校准告警。

- **扭矩常数**

扭矩常数是电机产生的扭矩除以 Q 轴电流，它跟电机 Kv 值的关系是：$Torque\_Constant = 8.27/K_v$。

```
odrv0.axis0.motor.config.torque_constant
```

扭矩常数是否正确并不影响电机的运行，但会影响用户做力矩控制时输入的值的单位换算，如果用户想用单位 A 而不是 Nm 来进行力矩控制，只需将这个值设为 1 即可。

- **温度传感**

驱动板和电机内部均带有 NTC 温度传感器，如果使能的话，驱动器会根据温度来控制输出

```
# 电机温度保护
odrv0.axis0.motor.motor_thermistor.config.enabled = 1
odrv0.axis0.motor.motor_thermistor.config.temp_limit_lower = 20
odrv0.axis0.motor.motor_thermistor.config.temp_limit_upper = 100
# 驱动板温度保护
odrv0.axis0.motor.fet_thermistor.config.enabled = 1
odrv0.axis0.motor.fet_thermistor.config.temp_limit_lower = 20
odrv0.axis0.motor.fet_thermistor.config.temp_limit_upper = 100
# 获取温度
odrv0.axis0.motor.motor_thermistor.temperature #电机温度
odrv0.axis0.motor.fet_thermistor.temperature #驱动板温度
```

电流（扭矩），从而对驱动器和电机进行保护。

- **PID 调整**

```
odrv0.axis0.controller.config.pos_gain=20.0
odrv0.axis0.controller.config.vel_gain=0.16
odrv0.axis0.controller.config.vel_integrator_gain=0.32
```

下述过程可以为用户调节 PID 参数提供一个参考：

1. 设置 PID 初始值
2. 将 vel_integrator_gain 调为 0

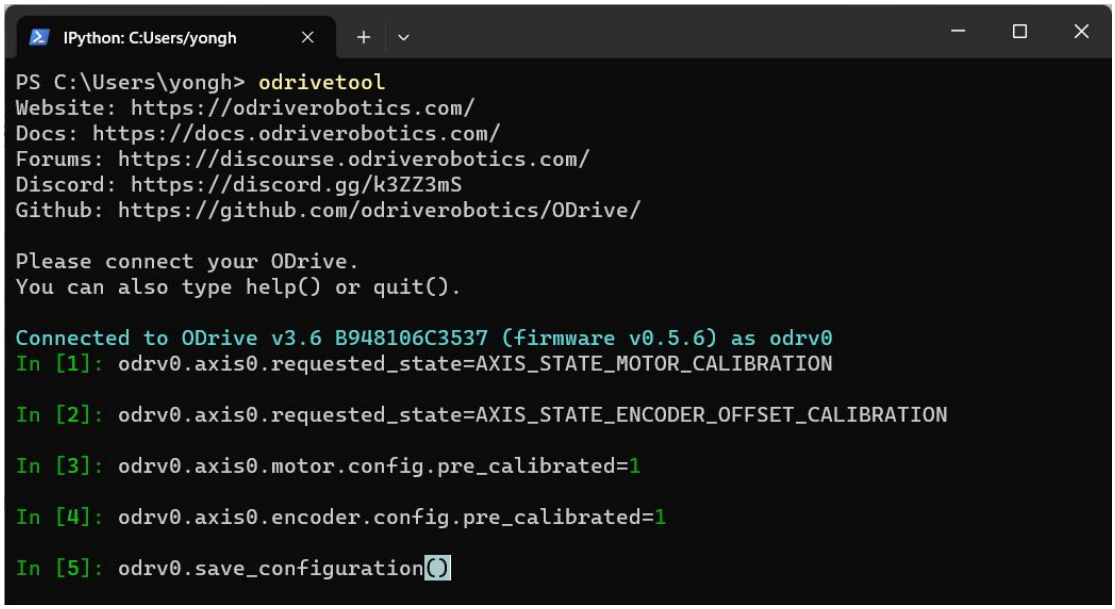
```
odrv0.axis0.controller.config.vel_integrator_gain=0
```

3. 调节 vel_gain 方法：
    1) 用速度控制模式转动电机，如果转动不平稳，有抖动或震动，减小 vel_gain，直到转动平稳
    2) 接着，每次把 vel_gain 增大 30%左右，直到出现明显的抖动
    3) 此时，将 vel_gain 减小 50%左右，即可稳定
4. 调节 pos_gain 方法：
    1) 用位置模式尝试转动电机，如果转动不平稳，有拉动或震动，减小 pos_gain，直到转动平稳
    2) 接着，每次把 pos_gain 增大 30%左右，直到位置控制出现明显的过调（即每次位置控制电机会超出目标位置，然后振荡回到目标位置）
    3) 然后，逐渐减小 pos_gain，直到过调现象消失
5. 在上述 4 步调整过后，可将 vel_integrator_gain 设置为 0.5*bandwidth*vel_gain，其中 bandwidth 是系统控制带宽。何为控制带宽？比如，从用户设置目标位置，到电机真正到达目标位置的时间为 10ms，则控制带宽就是 100Hz，那么 vel_integrator_gain=0.5*100*vel_gain。

在上述调参过程中，建议使用 3.1.8 中的图形化手段实时查看调参效果，避免肉眼感知的误差。

3.1.4 上电校准

用户第一次使用微电机时，需要对电机以及编码器进行校准。在校准之前，请固定好电机，或用手握紧，输出轴空载，校准过程如下：




```
odrv0.axis0.requested_state = AXIS_STATE_MOTOR_CALIBRATION
dump_errors(odrv0)
odrv0.axis0.requested_state = AXIS_STATE_ENCODER_OFFSET_CALIBRATION
dump_errors(odrv0)
odrv0.axis0.motor.config.pre_calibrated = 1
odrv0.axis0.encoder.config.pre_calibrated = 1
odrv0.save_configuration()
```

分步解释如下：

- 第一步：电机参数自识别

测量电机的相电阻和相电感，会听到尖利的“嘀”一声。相电阻和相电感的测量结果可通过下

```
odrv0.axis0.motor.config.phase_resistance
odrv0.axis0.motor.config.phase_inductance
```

述指令查看：

- 第二步：查看错误码

查看第一步过后的系统错误码，如果出现任何的红色错误码，则需要重启电机，然后重试，或者报告售后。

- 第三步：编码器校准

对编码器进行校准，包括编码器的安装角度与电机机械角度的校准，以及编码器自身的校准。在这个校准过程中，电机会缓慢正转一个角度，再反转一个角度。如果只正转后就停止，则说明有错误，请通过第四步查看错误码。

- 第四步：查看错误码

在第三步编码器校准后，查看系统的错误码。通常出现的错误是 ERROR_CPR_POLEPAIRS_MISMATCH，表示编码器的 CPR 设置错误，或者电机的极对数设置错误，

```
odrv0.axis0.encoder.config.cpr
odrv0.axis0.motor.config.pole_pairs
```

请通过下述指令查看/设置：

- 第五步：写入电机校准成功标志
- 第六步：写入编码器校准成功标志

```
odrv0.axis0.encoder.config.pre_calibrated = 1
odrv0.axis0.motor.config.pre_calibrated = 1
odrv0.save_configuration()
```

- 第七步：存储校准结果并重启

3.1.5 修改 ID

驱动器 ID 是总线唯一的，范围是 0~63，默认的 ID 为 0。如果用户有多台电机串联在总线上，则需要给每台电机设置不同的 ID。用户可以通过 USB 或总线方式修改 ID：

- USB 修改 ID

```
odrv0.axis0.config.can.node_id = xxx
```

用户通过 odrivetool 连接上驱动器后，可通过下述指令修改 ID：

- 总线修改 ID

请参见 4.1.2 中 Set_Axis_Node_ID 指令消息。

3.1.6 存储和备份参数

在任何参数修改过后，请务必存储，否则所做更改将在断电或重启后失效。存储参数后驱动

```
odrv0.save_configuration()
```

器将重启。

参数备份：

```
odrivetool backup-config "d:/test.json"
```

其中“d:\test.json”是用户可自由修改的保存路径和文件名。

```
odrivetool restore-config "d:/test.json"
```

参数恢复的指令为：

3.1.7 四种控制模式

经历上述准备工作和参数配置过后，就可以尝试控制电机进行不同模式的转动。8108-8 支持位置控制，速度控制，力矩控制，及运动控制模式。

在位置控制模式中，支持滤波位置控制（Filtered Position Control），梯形曲线位置控制（Trajectory Control），周期位置控制（Circular Position Control）；

在速度控制模式中，支持直接速度控制（Velocity Control），斜坡速度控制（Ramped Velocity Control）；

在力矩控制模式中，支持直接力矩控制（Torque Control），斜坡力矩控制（Ramped Torque Control）。

运动控制模式是综合位置，速度和力矩的控制模式，通常用于需要瞬时爆发力强的场景，如机器人膝关节。行业内也有用户将其称之为 MIT 控制模式，此称呼来源于 MIT 开源机械狗，因为它采用了这种运动控制模式来控制电机。

在后续对每一种控制模式的详细描述中，此文档中用 USB 控制指令来做示例，但同样也可用通信协议（如 CAN）做同样的控制，逻辑是一致的。

如何让电机转起来？

◆ 启动电机（进入闭环控制状态）

在所有后续的控制操作中，要让电机转动，都需要让电机进入闭环控制状态，指令如下：

odrv0.axis0.requested_state = 8

◆ 停止电机（进入空闲状态）

用户需要让电机停止运行，或希望修改参数及保存参数，都需要先让电机进入空闲状态，指令如下：

odrv0.axis0.requested_state = 1

➢ **滤波位置控制（Filtered Position Control）**

如果用户希望自己生成位置曲线，然后以一定频率来发送位置控制指令，则建议使用滤波位置控制，因为这种模式将这些指令平滑的衔接到一起执行。如果这种情况下采用梯形曲线位置控制，则有可能会让电机转动产生顿挫感或颗粒感。

在这种模式下需要根据发送指令的频率来调整滤波带宽，一个比较好的经验是，将带宽设置

```
odrv0.axis0.controller.config.input_filter_bandwidth = 25
```

为指令频率（单位 Hz）的一半，如以 50Hz 频率发送指令，则：

使能滤波位置控制：

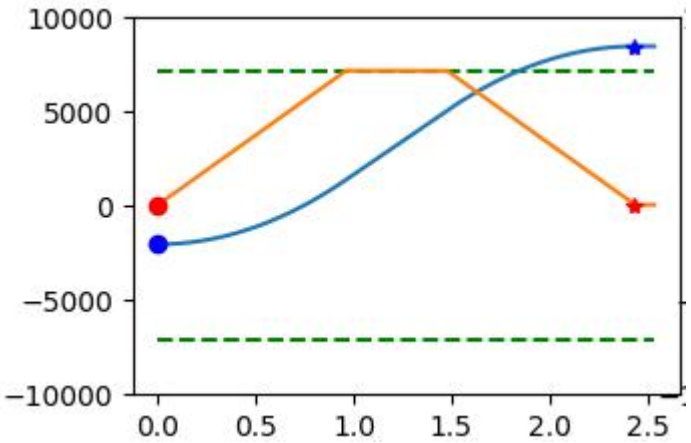
然后进行位置控制：

```
odrv0.axis0.controller.config.control_mode = 3
odrv0.axis0.controller.config.input_mode = 3
```

```
odrv0.axis0.controller.input_pos = 10 #单位 turns
```

- **梯形曲线位置控制（Trajectory Control）**

这个模式可以让用户设置加速度，滑行速度和减速度，来控制电机从一个位置平滑移动到另一个位置。所谓“梯形”是指其速度曲线看起来是梯形，如下图所示，橙色是速度，蓝色是位置：




```
odrv0.axis0.trap_traj.config.vel_limit #滑行速度最大值，单位 turn/s
odrv0.axis0.trap_traj.config.accel_limit #加速度最大值，单位 turn/s^2
odrv0.axis0.trap_traj.config.decel_limit #减速度最大值，单位 turn/s^2
odrv0.axis0.controller.config.inertia #惯量，单位 Nm/(turn/s^2)
```

可调节的控制参数：

请注意，惯量 x 加速度=扭矩，这个值默认为 0。这个值可以改善系统响应，但跟电机的负载有直接关系。上述四个值均大于等于 0。同时需要注意的是，前面提到过的电流门限和速度门限仍然会在全局起作用，比如，上述滑行速度最大值如果设置为高于系统级别的 vel_limit，则起作用的是全局 vel_limit。

```
odrv0.axis0.controller.config.control_mode = 3
odrv0.axis0.controller.config.input_mode = 5
```

要使能梯形曲线控制模式，首先：

然后进行位置控制：

- **周期位置控制（Circular Position Control）**

```
odrv0.axis0.controller.input_pos = 10 #单位 turns
```

这种模式适用于连续增量位置控制，如机器人轮毂朝一个方向连接转动一段时间，或传送履带一直运转，如果采用通常的位置控制模式，则目标位置会逐渐增大到一个很大的值，从而有由于浮点数精度问题而导致定位不准的错误。

```
odrv0.axis0.controller.config.circular_setpoints = 1
```

使能：

在这种模式下，每一小步都在单圈内，input_pos 的范围是[0, 1)。如果 input_pos 增加到超过这个范围，则会被转换为单圈内的值。如果用户希望单步超过单圈，可以设置下述参数为大于

```
odrv0.axis0.controller.config.circular_setpoint_range = <N>
```

1 的数：

- **直接速度控制（Velocity Control）**

```
odrv0.axis0.controller.config.control_mode = 2
odrv0.axis0.controller.config.input_mode = 1
```

这种模式是最简单的速度控制，使能如下：

然后输入目标速度来进行控制：

```
odrv0.axis0.controller.input_vel = 10 #单位 turn/s
```

- **斜坡速度控制（Ramped Velocity Control）**

斜坡速度控制模式是指按照一定斜率来逐步将速度提升到目标值，会比上述直接速度控制更

```
odrv0.axis0.controller.config.control_mode = 2
odrv0.axis0.controller.config.input_mode = 2
```

加缓和，使能如下：

通过调整斜率来控制加速度：

然后输入目标速度来进行控制：

```
odrv0.axis0.controller.input_vel = 10 #单位 turn/s
```

- **直接力矩控制（Toruqe Control）**

```
odrv0.axis0.controller.config.control_mode = 1
odrv0.axis0.controller.config.input_mode = 1
```

这是最简单的力矩（电流）控制模式，使能如下：

力矩控制的单位是 Nm，而驱动器固件中电流单位是 A，所以还需要设置力矩常数，以让驱动器能够将 Nm 转换为电流，从而按需求驱动电机输出力矩。

```
# 力矩常数大约等于8.23/K_v
odrv0.axis0.motor.config.torque_constant = 8.23/12.3
```

```
odrv0.axis0.controller.input_torque = 1.2 #单位 Nm
```

然后输入目标速度来进行控制：

另外还需注意的是，如果用户想限制力矩模式下的最大速度，可以打开 enable_torque_mode_vel_limit，并设置 vel_limit，如：

```
odrv0.axis0.controller.config.enable_torque_mode_vel_limit = 1
odrv0.axis0.controller.config.vel_limit = 30 #单位 turn/s
```

- **斜坡力矩控制（Ramped Torque Control）**

```
odrv0.axis0.controller.config.control_mode = 1
odrv0.axis0.controller.config.input_mode = 6
```

斜坡力矩控制非常类似于斜坡速度控制，使能如下：

调整斜率如下：

```
odrv0.axis0.controller.config.torque_ramp_rate = 0.1 #斜率单位是 Nm/s
```

➢ **运动控制（MIT Control）**

运动控制模式通过综合控制位置，速度和力矩来控制电机运动到目标位置，可以下述公式来表示：

$$T_{target} = K_p \times P_{diff} + K_d \times V_{diff} + T_{ff}$$

$$P_{diff} = P_{target} - P_{current}$$

$$V_{diff} = V_{target} - V_{current}$$

其中T_{target}是目标力矩，P_{diff}是位置误差，V_{diff}是速度误差，K_p是位置控制增益，K_d是速度控制增益（或叫阻尼系数），T_{ff}是前馈力矩。

```
odrv0.axis0.controller.config.control_mode = 3
odrv0.axis0.controller.config.input_mode = 9
```

运动控制模式使能如下：

调整增益：

```
odrv0.axis0.controller.input_pos = 5 #单位 turns
odrv0.axis0.controller.input_mit_kp = <float> #位置增益，单位 Nm/turn
odrv0.axis0.controller.input_mit_kd = <float> #阻尼系数，单位 Nm/turn/s
```

然后通过输入 input_pos，input_vel，input_torque 来进行运动控制：

请注意，在 USB 控制时所输入的位置、速度和扭矩，均是指转子侧，而用 CAN 进行 MIT 控制时，协议中的位置、速度和扭矩均是指输出轴侧，这是为了与 MIT 开源协议保持一致！

3.1.8 进阶

1) 常用指令列表

在连接成功后，用户可通过指令对电机进行控制，并获取电机运行的参数。下表是常用指令和调测过程及说明：

类型	指令	说明
基础指令	dump_errors(odrv0)	打印所有的错误信息
	odrv0.clear_errors()	清除所有的错误信息
	odrv0.save_configuration()	在修改过参数，或电机自动识别参数或校准过后，请务必执行此指令存储修改，否则断电后丢失所有修改。
	odrv0.reboot()	重启驱动器
	odrv0.vbus_voltage	获取电源电压（V）
	odrv0.ibus	获取电源电流（A）
	odrv0.hw_version_major	硬件主版本号，8108-8 目前的主版本号为 3
	odrv0.hw_version_minor	硬件次版本号，8108-8 目前的次版本号为 8
	odrv0.hw_version_variant	同一硬件配置下的不同型号码，8108-8 对应的型号码为 1
	odrv0.can.config.r120_gpio_num	控制 CAN 接口的 120R 匹配电阻开关的 GPIO 号
	odrv0.can.config.enable_r120	控制 CAN 接口的 120R 匹配电阻开关
	odrv0.can.config.baud_rate	CAN 的波特率设置
参数配置指令	odrv0.config.dc_bus_undervoltage_trip_level	低电压告警门限（V）
	odrv0.config.dc_bus_overvoltage_trip_level	超电压告警门限（V）
	odrv0.config.dc_max_positive_current	线电流最大值（正值）（A）
	odrv0.config.dc_max_negative_current	线电流反向充电最大值（负值）（A）
	odrv0.axis0.motor.config.resistance_calib_max_voltage	电机参数识别时最大电压值，一般此值稍小于电源电压一半，如 24V 供电，可设置为 10
	odrv0.axis0.motor.config.calibration_current	电机参数识别时最大电流值，此值一般可设置为 2~5A，不可过大，也不可过小。
	odrv0.axis0.motor.config.torque_constant	电机的力矩常数（Nm/A）
	odrv0.axis0.min_endstop.config odrv0.axis0.max_endstop.config	最小（LW1）/最大（LW2）限位开关的配置： enabled：使能与否 gpio_num：对应的 IO 号，请将最小限位的 IO 号设置为 1，最大限位的 IO 号设置为 2
	odrv0.axis0.encoder.config.index_offset	用户设置的零点偏移，这个值是用户零点相对于编码器零点的偏移值。在设置这个偏移值并保存设置后，所有用户输入的位置控制目标值均是以此用户零点为基准。

	odrv0.axis0.motor.motor_thermistor.config	配置电机温度传感器： enabled：使能与否 temp_limit_lower：温度下限 temp_limit_upper：温度上限
	odrv0.axis0.motor.fet_thermistor.config	
	odrv0.axis0.motor.motor_thermistor.temperature	电机温度
	odrv0.axis0.motor.fet_thermistor.temperature	驱动器温度
校准指令	odrv0.axis0.requested_state=4	对电机进行参数识别，包括识别相电阻，相电感，以及对三相电流平衡性做出校准。这个过程会历时 3~6 秒钟，电机会发出尖利的声音。在声音停止后，或 6 秒后没有声音时，均运行 dump_errors(odrv0)查看错误，确认没有错误再进行其他操作。
	odrv0.axis0.requested_state=7	对编码器进行校准。执行此操作前，请确认电机输出轴没有任何负载，且用手或其他装置固定住电机。此操作执行后，电机会正向和反向旋转一定时间，对编码器进行识别和校准。在电机停转后，运行 dump_errors(odrv0)查看错误，确认没有错误再进行其他后续步骤。
	odrv0.axis0.encoder.config.pre_calibrated=1	写入预校准成功，表示不用每次上电进行校准。这个参数仅在上述校准成功过后才能写入，否则写入会失败。
	odrv0.axis0.controller.config.load_encoder_axis=0	确保当前操作电机为第 0 个电机。此操作仅在 BETA 中必要，量产版本中无效。
控制指令	odrv0.axis0.requested_state=1	停止电机，进入空闲状态
	odrv0.axis0.requested_state=8	启动电机，进入闭环状态
	odrv0.axis0.motor.config.current_lim	电机运行最大线电流（A），超过此值会报过流告警。**请注意，此值不得大于 100。**
	odrv0.axis0.controller.config.vel_limit	电机运行最大速度（turn/s），电机转子速度超过此值会报超速告警。
	odrv0.axis0.controller.config.enable_vel_limit	速度限制开关，为 True 时上述 vel_limit 生效，为 False 时无效。
	odrv0.axis0.controller.config.control_mode	控制模式。 0：电压控制 1：力矩控制 2：速度控制 3：位置控制
	odrv0.axis0.controller.config.input_mode	输入模式。表示用户输入的控制值以什么方式去控制电机运转： 0：闲置 1：直接控制 2：速度斜坡

		3：位置滤波 5：梯形曲线 6：力矩斜坡 9：运动控制（MIT）
	odrv0.axis0.controller.config.vel_gain	速度环 PID 控制的 P 值
	odrv0.axis0.controller.config.vel_integrator_gain	速度环 PID 控制的 I 值
	odrv0.axis0.controller.input_mit_kp	运动控制（MIT）位置增益
	odrv0.axis0.controller.input_mit_kd	运动控制（MIT）速度增益（阻尼系数）
	odrv0.axis0.controller.config.pos_gain	位置环 PID 控制的 P 值
	odrv0.axis0.controller.input_torque	力矩控制的目标，或速度控制/位置控制的力矩前馈（Nm）
	odrv0.axis0.controller.input_vel	速度控制的目标，或位置控制的速度前馈（turn/s）
	odrv0.axis0.controller.input_pos	位置控制的目标（turns）
	odrv0.axis0.encoder.set_linear_count()	设置编码器的绝对位置，括号中输入 32 位整数，此整数绝对值需要小于 odrv0.axis0.encoder.config.cpr
	odrv0.axis0.trap_traj.config	包含三个参数： ➢ accel_limit：最大加速度（rev/s^2） ➢ decel_limit：最大减速度（rev/s^2） ➢ vel_limit：最大速度（rev/s） 这三个参数在 odrv0.axis0.controller.config.input_mode 为梯形曲线的时候起作用，调整位置控制的加减速效果。
	odrv0.axis0.controller.config.input_filter_bandwidth	位置滤波带宽，这个参数在 odrv0.axis0.controller.config.input_mode 为位置滤波的时候起作用，调节位置控制的加减速效果。

2) 图形化调测

在对电机进行调测时，如果需要实时监控某些运行参数，可以利用 python 强大的计算库和图形库，以及 Type-C 接口的高速吞吐能力实时输出电机参数。

1. 环境准备

```
pip install numpy matplotlib
```

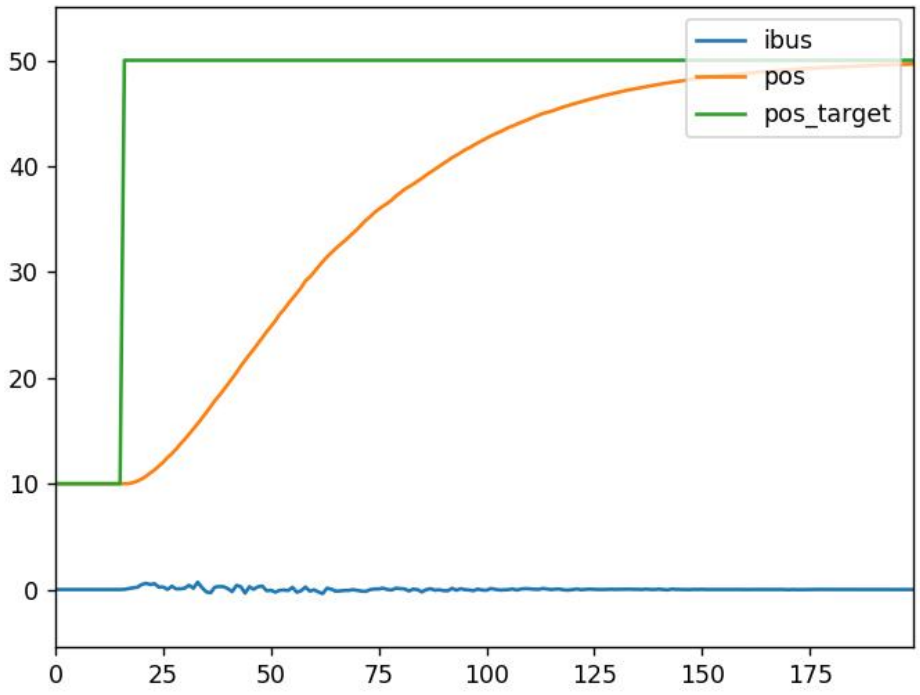
安装计算库和图形库：

2. 图形化参数输出

```
start_liveplotter(lambda:[odrv0.ibus,odrv0.axis0.encoder.pos_estimate,
odrv0.axis0.controller.input_pos],["ibus","pos","pos_target"])
```

在 odrivetool 命令行界面，调起图形库，读取任何电机运行指标，如：

这个指令将调起一个图形界面，实时输出下述三个指标：线电流、位置、目标位置。接下去，对电机进行位置控制，就会看到电机的实时位置控制曲线：



3) USB 和 CAN 兼容性

在本产品的早期版本（硬件版本小于等于 3.7，可通过下一小节 3.1.8 中的指令 odrv0.hw_version_major 和 odrv0.hw_version_minor 来获取）中，USB 与 CAN 不兼容，可通过下述方式在两种通信方式中切换（硬件版本大于 3.7 的可忽略本节内容）：

- USB 通信时切换到 CAN

当禁止 CAN 时，用户可使用 Type-C 接口通信，此时，可通过下列指令切换到 CAN：

```
odrv0.config.enable_can_a = True
odrv0.axis0.requested_state = AXIS_STATE_IDLE
odrv0.save_configuration()
```

- CAN 通信时切换到 USB

当使能 CAN 时，用户首先通过发送消息 Set_Axis_State（参数为 1，表示空闲（IDLE）状态）将电机切换到空闲状态，再通过发送 Disable_Can 消息来切换到 USB（参见 4.1.2）。

请注意，无论是从 USB 切换到 CAN，还是 CAN 切换到 USB，必须先让电机处于空闲（IDLE）状态，否则会切换失败。

4) CAN 匹配电阻开关

在驱动器上，已经板载一个 120 欧阻抗匹配电阻，用户可根据需要打开或关闭，指令示例

```
odrv0.can.config.r120_gpio_num = 5
odrv0.can.config.enable_r120 = True
```

如下：

5) 用户零点配置

默认情况下，用户从电机读取到的位置，以及做位置控制时的输入值，均是基于驱动器上的绝对值编码器的零点为基准。但是，在用户场景下，编码器的零点在大多数时候并不是用户零点，所以用户需要手动设置这个零点偏移。

一般来说，用户可以通过两种手段来定位这个零点，一种手段是通过限位开关，一种手段是手动设置零点偏移，即用户零点相对于编码器零点的偏移值：

```
# 用户先手动或者通过位置控制转动到期望的用户零点位置后：
odrv0.axis0.encoder.config.index_offset =
odrv0.axis0.encoder.pos_estimate
```

6) 限位开关

驱动器支持两个限位开关（LW1 和 LW2），其中 LW1 是最小位置，也是归零位置，LW2 是

```
odrv0.axis0.min_endstop.config.enabled = True
odrv0.axis0.min_endstop.config.gpio_num = 1
odrv0.axis0.max_endstop.config.enabled = True
odrv0.axis0.max_endstop.config.gpio_num = 2
```

最大位置。如要使用两个限位开关，请用下述配置：

在触发限位开关时，系统将会上报 MIN_ENDSTOP_PRESSED 或 MAX_ENDSTOP_PRESSED 错误，上位机可在此时执行相关的操作。

请注意，硬件版本 3.7 不支持限位开关功能。

3.2 固件更新下载

可以通过 SWD 接口（2.4.4）或 Type-C 接口（2.4.2）烧录固件，提供下述三种方式：

3.2.1 国民下载软件

1. USB（DFU）烧写

请注意，国民下载软件可通过 Type-C 接口进行烧录，也可通过 SWD 接口（仅支持 JLink 和 DAP）烧录，本节主要以 Type-C 接口烧录为例。

首先，下载国民烧录软件的 USB 驱动（https://www.cyberbeast.cn/filedownload/789489）并安装相应系统的驱动；然后，下载国民烧录软件（https://cyberbeast.cn/filedownload/766844），解压到任意目录，并且运行。

然后，连接 Type-C 接口，进入 odrivetool，并执行下述指令将驱动器置于 DFU 模式：

```
odrv0.enter_dfu_mode()
```

最后，用国民烧录软件进行烧写，如下图所示。请注意，在烧写完成过后，请点击“常用操作”，然后点击“复位”，可重启驱动器，并可正常通过 odrivetool 连接调测。

2. SWD（JLink 或 DAP）烧写

使用 SWD 方式下载跟 DFU 模式相似，但需要通过 SWD 调试接口（2.4.4）进行连接，且在上图中选择相应的调试工具（JLink 或 DAP）。

3.2.2 pyocd

pyocd 是 openOCD 的 python 版本，可支持 STLink，JLink，DAP 等通用调试工具进行擦除、烧写、重置等操作。**请注意，必须用 SWD 接口连接驱动器。**关于 SWD 的线序，请参见 2.4.4。**SWD 接口中有 3.3V 电源，请不要接错线序，以免损毁驱动器！**

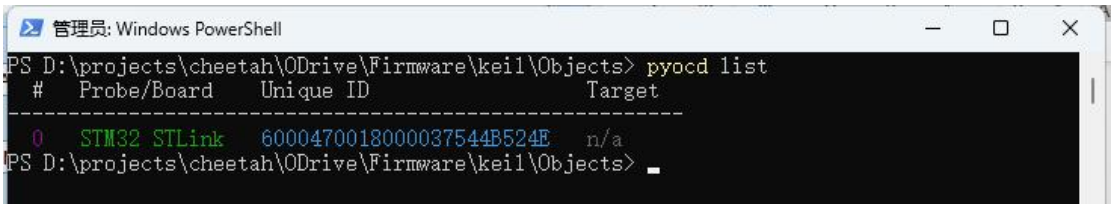
```
pip install pyocd
```

1. 安装

2. 烧写

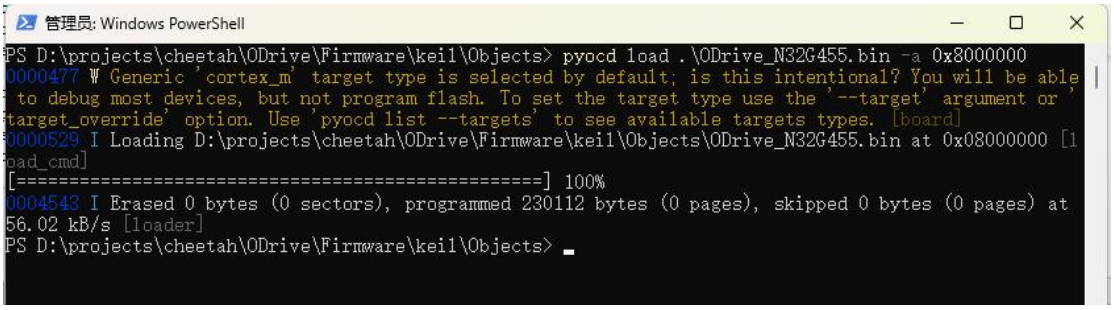
```
pyocd list
```

首先，列出连接的调试工具：



然后，执行下述指令烧写 bin 文件：

```
pyocd load .\ODrive_N32G455.bin -a 0x8000000
```



3.2.3 电机精灵（即将推出）

4 通信协议及示例

4.1 CAN 协议

默认通信接口是 CAN，最大通信速率 1Mbps（可通过 odrv0.can.config.baud_rate 读取和设置），出厂默认速率 500Kbps。**请注意：早期硬件版本（小于等于 3.7）中 USB 与 CAN 不兼容，请参见 3.1.8 中第 3)条，如何从 USB 切换到 CAN。**

4.1.1 协议帧格式

CAN 通信采用标准帧格式，数据帧，11 位 ID，8 字节数据，如下表所示（左为 MSB，右为 LSB）：

数据域	CAN ID（11bits）		Data（8 bytes）
分段	Bit10 ~ Bit5	Bit4 ~ Bit0	Byte0 ~ Byte7
描述	node_id	cmd_id	通信数据

- node_id：代表这个电机在总线上的唯一 ID，可在 odrivetool 中用 odrv0.axis0.config.can.node_id 来读取和设置。
- cmd_id：指令编码，表示协议的消息类型，请参见本节余下内容。
- 通信数据：8 个字节，每一个消息中携带的参数会被编码成整数或浮点数，字节序为小端（small endian），其中浮点数是按照 IEEE 754 标准进行编码（可通过网站 https://www.h-schmidt.net/FloatConverter/IEEE754.html 测试编码）。

以 4.1.2 中描述的 Set_Input_Pos 消息为例，假设其三个参数分别为：Input_Pos=3.14，Vel_FF=1000（表示 1rev/s），Torque_FF=5000（表示 5Nm），而 Set_Input_Pos 消息的 CMD ID=0x00C，假设驱动器的节点（node_id）被设置成 0x05，则：

- 11 位 CAN ID=(0x05<<5)+0x0C=0xAC
- 根据 4.1.2 中对于 Set_Input_Pos 的描述可知，Input_Pos 在第 0 个字节开始的 4 个字节，编码为 C3 F5 48 40（浮点数 3.14 用 IEEE 754 标准编码为 32 位数 0x4048f5c3），Vel_FF 在第 4 个字节开始的 2 个字节，编码为 E8 03（1000=0x03E8），Torque_FF 在第 6 个字节开始的 2 个字节，编码为 88 13（5000=0x1388），则 8 个字节的通信数据为：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
C3	F5	48	40	E8	03	88	13

4.1.2 帧消息

下表列出了所有的可用消息：

CMD ID	名称	方向	参数
0x001	Heartbeat	电机→主机	Axis_Error Axis_State Motor_Flag Encoder_Flag Controller_Flag Traj_Done Life
0x002	Estop	主机→电机	
0x003	Get_Error	电机→主机	Error_Type
0x004	RxSdo	电机→主机	
0x005	TxSdo	电机→主机	
0x006	Set_Axis_Node_ID	主机→电机	Axis_Node_ID
0x007	Set_Axis_State	主机→电机	Axis_Requested_State
0x008	Mit_Control	主机→电机	

0x009	Get_Encoder_Estimates	电机→主机	Pos_Estimate Vel_Estimate
0x00A	Get_Encoder_Count	电机→主机	Shadow_Count Count_In_Cpr
0x00B	Set_Controller_Mode	主机→电机	Control_Mode Input_Mode
0x00C	Set_Input_Pos	主机→电机	Input_Pos Vel_FF Torque_FF
0x00D	Set_Input_Vel	主机→电机	Input_Vel Torque_FF
0x00E	Set_Input_Torque	主机→电机	Input_Torque
0x00F	Set_Limits	主机→电机	Velocity_Limit Current_Limit
0x010	Start_Anticogging	主机→电机	
0x011	Set_Traj_Vel_Limit	主机→电机	Traj_Vel_Limit
0x012	Set_Traj_Accel_Limits	主机→电机	Traj_Accel_Limit Traj_Decel_Limit
0x013	Set_Traj_Inertia	主机→电机	Traj_Inertia
0x014	Get_Iq	电机→主机	Iq_Setpoint Iq_Measured
0x015	Get_Sensorless_Estimates	电机→主机	Pos_Estimate Vel_Estimate
0x016	Reboot	主机→电机	
0x017	Get_Bus_Voltage_Current	电机→主机	Bus_Voltage Bus_Current
0x018	Clear_Errors	主机→电机	
0x019	Set_Linear_Count	主机→电机	Linear_Count
0x01A	Set_Pos_Gain	主机→电机	Pos_Gain
0x01B	Set_Vel_Gains	主机→电机	Vel_Gain Vel_Integrator_Gain
0x01C	Get_Torques	电机→主机	Torque_Setpoint Torque
0x01D	Get_Powers	电机→主机	Electrical_Power Mechanical_Power
0x01E	Disable_Can	主机→电机	
0x01F	Save_Configuration	主机→电机	

所有消息的详细描述如下：

- Heartbeat

CMD ID: 0x001（电机→主机）

固件版本小于（包含）0.5.11 的心跳格式如下：

起始字节	名称	类型	odrivetool 访问
0	Axis_Error	uint32	odrv0.axis0.error
4	Axis_State	uint8	odrv0.axis0.current_state
5	Motor_Flag	uint8	1：odrv0.axis0.motor.error 不为 0 0：odrv0.axis0.motor.error 为 0
6	Encoder_Flag	uint8	1：odrv0.axis0.encoder.error 不为 0 0：odrv0.axis0.encoder.error 为 0
7	Controller_Flag	uint8	bit7：odrv0.axis0.controller.trajectory_done bit0： 1：odrv0.axis0.controller.error 不为 0 0：odrv0.axis0.controller.error 为 0

固件版本大于（包含）0.5.12 的心跳格式如下：

起始字节	名称	类型	odrivetool 访问
0	Axis_Error	uint32	odrv0.axis0.error
4	Axis_State	uint8	odrv0.axis0.current_state
5	Flags	uint8	bit0：odrv0.axis0.motor.error 是否为 0 bit1：odrv0.axis0.encoder.error 是否为 0 bit2：odrv0.axis0.controller.error 是否为 0 bit7：odrv0.axis0.controller.trajectory_done，即位置曲线是否执行完毕
6	Reserved	uint8	保留
7	Life	uint8	周期消息的生命值，每一个心跳消息加 1，范围 0-255，如果此生命值不连续，表示心跳消息丢失，即通信不稳。

固件版本大于（包含）0.5.13 的心跳格式如下：

起始字节	名称	类型	odrivetool 访问
0	Axis_Error	uint32	odrv0.axis0.error
4	Axis_State	uint8	odrv0.axis0.current_state
5	Flags	uint8	bit0：odrv0.axis0.motor.error 是否为 0 bit1：odrv0.axis0.encoder.error 是否为 0 bit2：odrv0.axis0.controller.error 是否为 0 bit3：odrv0.error 是否为 0 bit7：odrv0.axis0.controller.trajectory_done，即位置曲线是否执行完毕
6	Reserved	uint8	保留
7	Life	uint8	周期消息的生命值，每一个心跳消息加 1，范围 0-255，如果此生命值不连续，表示心跳消息丢失，即通信不稳。

- Estop

CMD ID: 0x002（主机→电机）无参数无数据。

此指令会导致电机紧急停机，并报 ESTOP_REQUESTED 异常。

- Get_Error

CMD ID: 0x003（电机→主机）

输入（主机→电机）：

起始字节	名称	类型	说明
0	Error_Type	uint8	0：获取电机异常 1：获取编码器异常 2：获取无感异常 3：获取控制器异常 4：获取系统异常

输出（电机→主机）：

起始字节	名称	类型	odrivetool 访问
0	Error	uint32	不同输入 Error_Type： 0：odrv0.axis0.motor.error 1：odrv0.axis0.encoder.error 2：odrv0.axis0.sensorless_estimator.error 3：odrv0.axis0.controller.error 4：odrv0.error

- RxSdo

CMD ID: 0x004（主机→电机）

输入：

起始字节	名称	类型	说明
0	opcode	uint8	0：读 1：写
1	Endpoint_ID	uint16	请下载所有参数和接口函数对应的 ID 的 JSON 文件： https://www.cyberbeast.cn/filedownload/837298
3	预留	uint8	
4	Value	uint8[4]	根据 Endpoint_ID 不同而不同，可参见上述 JSON 中的描述。如 Endpoint_ID 对应一个可读写的 float 值，则此处 4 个字节为 IEEE 编码而成的 float 值，opcode=1 时将此值写入这个 float 值。

输出（当上述 opcode=0 时）：

起始字节	名称	类型	说明
0	opcode	uint8	固定为 0
1	Endpoint_ID	uint16	请下载所有参数和接口函数对应的 ID 的 JSON 文件：https://www.cyberbeast.cn/filedownload/837298
3	预留	uint8	
4	Value	uint8[4]	根据 Endpoint_ID 不同而不同，可参见上述 JSON 中的描述。如 Endpoint_ID 对应一个可读的 uint32 值，则此处 4 个字节为小端字节序的 uint32。

- TxSdo

CMD ID: 0x005（电机→主机）

用法跟 opcode=1 时的 RxSdo 一样。

- Set_Axis_Node_ID

CMD ID: 0x006（主机→电机）

起始字节	名称	类型	odrivetool 访问
0	Axis_Node_ID	uint32	odrv0.axis0.config.can.node_id

- Set_Axis_State

CMD ID: 0x007（主机→电机）

起始字节	名称	类型	odrivetool 访问
0	Axis_Requested_State	uint32	odrv0.axis0.requested_state

- Mit_Control

CMD ID: 0x008

这是模拟 MIT 开源运动控制协议（https://github.com/mit-biomimetics/Cheetah-Software）的实现。

请注意，在 USB 控制时所输入的位置、速度和扭矩，均是指转子侧，而用 CAN 进行 MIT 控制时，协议中的位置、速度和扭矩均是指输出轴侧，这是为了与 MIT 开源协议保持一致！

✓ 主机→电机

CAN 数据帧位	含义	说明

BYTE0 BYTE1	**位置**: 总共 16 位, BYTE0 为高 8 位, BYTE1 为低 8 位 输出轴的多圈位置，单位为弧度（RAD）	实际**位置**为 double 型，需要转换为 16 位 int 型，转换过程为: pos_int = (pos_double + 12.5)*65535 / 25
BYTE2 BYTE3 BYTE4	**速度**：总共 12 位，BYTE2 为其高 8 位，BYTE3[7-4]（高 4 位）为其低 4 位。表示输出轴的角速度，单位为 RAD/s **KP 值** ： 总共 12 位，BYTE3[3-0]（低 4 位）为其高 4 位，BYTE4 为其低 8 位。	实际**速度**为 double 型，需要转换为 12 位 int 型，转换过程为: vel_int = (vel_double + 65) * 4095 / 130 **KP 值**实际为 double 型，需要转换为 12 位 int 型，转换过程为: kp_int = kp_double * 4095 / 500
BYTE5 BYTE6 BYTE7	**KD 值** ：总共 12 位, BYTE5 为其高 8 位，BYTE6[7-4]（高 4 位）为其低 4 位。 **力矩**: 总共 12 位，BYTE6[3-0]（低 4 位）为其高 4 位，BYTE7 为其低 8 位。单位是 N.m。	**KD 值**实际为 double 型，需要转换为 12 位 int 型，转换过程为: kd_int = kd_double * 4095 / 5 实际**力矩**为 double 型，需要转换为 12 位 int 型，转换过程为: t_int = (t_double + 50) * 4095 / 100 转矩常数的单位为 N.m/A

✓ 电机→主机

CAN 数据帧位	含义	说明
BYTE0	node id	驱动器 node id
BYTE1 BYTE2	**位置**: 总共 16 位, BYTE1 为高 8 位, BYTE2 为低 8 位 输出轴的多圈位置，单位为弧度（RAD）	实际**位置**为 double 型，需要从 16 位 int 型转换过来，转换过程为: pos_double = pos_int * 25 / 65535 – 12.5
BYTE3	**速度**：总共 12 位，BYTE3 为其高 8 位，	实际**速度**为 double 型，需要从 12 位 int

BYTE4	BYTE4[7-4]（高 4 位）为其低 4 位。表示输出轴的角速度，单位为 RAD/s **力矩**：总共 12 位，BYTE4[3-0]（低 4 位）为其高 4 位，BYTE5 为其低 8 位。单位是 N.m。	型转换过来，转换过程为： vel_double = vel_int * 130 / 4095 – 65 实际**力矩**为 double 型，需要从 12 位 int 型转换过来，转换过程为： t_double = t_int * 100 / 4095 - 50 转矩常数的单位为 N.m/A
BYTE5		

- Get_Encoder_Estimates

CMD ID: 0x009（电机→主机）

起始字节	名称	类型	单位	odrivetool 访问
0	Pos_Estimate	float32	rev	odrv0.axis0.encoder.pos_estimate
4	Vel_Estimate	float32	rev/s	odrv0.axis0.encoder.vel_estimate

- Get_Encoder_Count

CMD ID: 0x00A（电机→主机）

起始字节	名称	类型	odrivetool 访问
0	Shadow_Count	int32	odrv0.axis0.encoder.shadow_count
4	Count_In_Cpr	int32	odrv0.axis0.encoder.count_in_cpr

- Set_Controller_Mode

CMD ID: 0x00B（主机→电机）

起始字节	名称	类型	odrivetool 访问
0	Control_Mode	uint32	odrv0.axis0.controller.config.control_mode
4	Input_Mode	uint32	odrv0.axis0.controller.config.input_mode

- Set_Input_Pos

CMD ID: 0x00C（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Input_Pos	float32	rev	odrv0.axis0.controller.input_pos
4	Vel_FF	int16	0.001rev/s	odrv0.axis0.controller.input_vel
6	Torque_FF	int16	0.001Nm	odrv0.axis0.controller.input_torque

- Set_Input_Vel

CMD ID: 0x00D（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Input_Vel	float32	rev/s	odrv0.axis0.controller.input_vel
4	Torque_FF	float32	Nm	odrv0.axis0.controller.input_torque

- Set_Input_Torque

CMD ID: 0x00E（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Input_Torque	float32	Nm	odrv0.axis0.controller.input_torque

- Set_Limits

CMD ID: 0x00F（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Velocity_Limit	float32	rev/s	odrv0.axis0.controller.config.vel_limit
4	Current_Limit	float32	A	odrv0.axis0.motor.config.current_lim

- Start_Anticogging

CMD ID: 0x010（主机→电机）

进行力矩纹波校准。

- Set_Traj_Vel_Limit

CMD ID: 0x011（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Traj_Vel_Limit	float32	rev/s	odrv0.axis0.trap_traj.config.vel_limit

- Set_Traj_Accel_Limits

CMD ID: 0x012（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Traj_Accel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.accel_limit
4	Traj_Decel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.decel_limit

- Set_Traj_Inertia

CMD ID: 0x013（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Traj_Inertia	float32	Nm/(rev/s^2)	odrv0.axis0.controller.config.inertia

- Get_Iq

CMD ID: 0x014（电机→主机）

起始字节	名称	类型	单位	odrivetool 访问
0	Iq_Setpoint	float32	A	odrv0.axis0.motor.current_control.Idq_setpoint
4	Iq_Measured	float32	A	odrv0.axis0.motor.current_control.Iq_measured

- Get_Sensorless_Estimates

CMD ID: 0x015（电机→主机）

起始字节	名称	类型	单位	odrivetool 访问
0	Pos_Estimate	float32	rev	odrv0.axis0.sensorless_estimator.pll_pos
4	Vel_Estimate	float32	rev/s	odrv0.axis0.sensorless_estimator.vel_estimate

- Reboot

CMD ID: 0x016（主机→电机）

- Get_Bus_Voltage_Current

CMD ID: 0x017（电机→主机）

起始字节	名称	类型	单位	odrivetool 访问
0	Bus_Voltage	float32	V	odrv0.vbus_voltage
4	Bus_Current	float32	A	odrv0.ibus

- Clear_Errors

CMD ID: 0x018（主机→电机）

清除所有错误和异常。

- Set_Linear_Count

CMD ID: 0x019（主机→电机）

设置编码器绝对位置。

起始字节	名称	类型	odrivetool 访问
0	Linear_Count	int32	odrv0.axis0.encoder.set_linear_count()

- Set_Pos_Gain

CMD ID: 0x01A（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问

0	Pos_Gain	float32	(rev/s)/rev	odrv0.axis0.controller.config.pos_gain

- Set_Vel_Gains

CMD ID: 0x01B（主机→电机）

起始字节	名称	类型	单位	odrivetool 访问
0	Vel_Gain	float32	Nm/(rev/s)	odrv0.axis0.controller.config.vel_gain
4	Vel_Integrator_Gain	float32	Nm/rev	odrv0. axis0.controller.config.vel_integrator_gain

- Get_Torques

CMD ID: 0x01C（电机→主机）

起始字节	名称	类型	odrivetool 访问
0	Torque_Setpoint	float32	odrv0.axis0.controller.torque_setpoint
4	Torque	float32	无。表示当前力矩值。

- Get_Powers

CMD ID: 0x01D（电机→主机）

起始字节	名称	类型	odrivetool 访问
0	Electrical_Power	float32	odrv0.axis0.controller.electrical_power
4	Mechanical_Power	float32	odrv0.axis0.controller.mechanical_power

- Disable_Can

CMD ID: 0x01E（主机→电机）

禁用 CAN，并重启驱动器。

- Save_Configuration

CMD ID: 0x01F（主机→电机）

存储当前的配置，生效并重启。

4.1.3 CAN 协议实战

4.1.3.1 实战：上电校准

发送 CAN 消息的序列如下：

CAN ID	帧类型	帧数据	说明
0x007	数据帧	04 00 00 00 00 00 00 00	消息：Set_Axis_State

			参数：4 对电机进行校准
0x007	数据帧	07 00 00 00 00 00 00 00	消息：Set_Axis_State 参数：7 对编码器进行校准

4.1.3.2 实战：速度控制

发送 CAN 消息的序列如下：

CAN ID	帧类型	帧数据	说明
0x00B	数据帧	02 00 00 00 02 00 00 00	消息：Set_Controller_Mode 参数：2/2 设置控制模式为速度控制，输入模式为速度斜坡
0x007	数据帧	08 00 00 00 00 00 00 00	消息：Set_Axis_State 参数：8 进入闭环控制状态
0x0D	数据帧	00 00 20 41 00 00 00 00	消息：Set_Input_Vel 参数：10/0 设置目标速度和力矩前馈，其中目标速度为 10（浮点数：0x41200000），力矩前馈为 0（浮点数：0x00000000）

4.1.3.3 实战：位置控制

发送 CAN 消息的序列如下：

CAN ID	帧类型	帧数据	说明
0x00B	数据帧	03 00 00 00 03 00 00 00	消息：Set_Controller_Mode 参数：3/3 设置控制模式为位置控制，输入模式为位置滤波
0x007	数据帧	08 00 00 00 00 00 00 00	消息：Set_Axis_State 参数：8 进入闭环控制状态
0x0C	数据帧	CD CC 0C 40 00 00 00 00	消息：Set_Input_Pos 参数：2.2/0/0 设置目标位置，速度前馈和力矩前馈，其中目标位置为 2.2（浮点数：0x400CCCCD），力矩前馈和速度前馈为 0

4.1.4 CANOpen 兼容性

如果 node ID 分配得当，可与 CANOpen 互通。下表列出了 CANopen 和本协议的有效 node ID 组合：

CANOpen node IDs	本协议 node IDs
32 … 127	0x10, 0x18, 0x20, 0x28
64 … 127	0x10, 0x11, 0x18, 0x19, 0x20, 0x21, 0x28, 0x29
96 … 127	0x10, 0x11, 0x12, 0x18, 0x19, 0x1A, 0x20, 0x21, 0x22, 0x28, 0x29, 0x2A

4.1.5 周期消息

用户可配置电机向上位机周期性发送消息，而不用上位机向电机发送请求消息。可通过 odrv0.axis0.config.can 下的一系列配置来打开/关闭周期消息（值为 0 表示关闭，为其他值表示周期时间，单位为 ms），如下表所示：

消息	odrivetool 配置	默认值
Heartbeat	odrv0.axis0.config.can.heartbeat_rate_ms	100
Get_Encoder_Estimates	odrv0.axis0.config.can.encoder_rate_ms	10
Get_Motor_Error	odrv0.axis0.config.can.motor_error_rate_ms	0
Get_Encoder_Error	odrv0.axis0.config.can.encoder_error_rate_ms	0
Get_Controller_Error	odrv0.axis0.config.can.controller_error_rate_ms	0
Get_Sensorless_Error	odrv0.axis0.config.can.sensorless_error_rate_ms	0
Get_Encoder_Count	odrv0.axis0.config.can.encoder_count_rate_ms	0
Get_Iq	odrv0.axis0.config.can.iq_rate_ms	0
Get_Sensorless_Estimates	odrv0.axis0.config.can.sensorless_rate_ms	0
Get_Bus_Voltage_Current	odrv0.axis0.config.can.bus_vi_rate_ms	0

默认情况下，前两种周期消息在出厂时打开，所以当用户监控 CAN 总线时，会看到两种消

```
odrv0.axis0.config.can.heartbeat_rate_ms = 0
odrv0.axis0.config.can.encoder_rate_ms = 0
```

息以设定周期进行广播。用户可通过下述指令关闭它们：

对于各个消息的详细内容，请参见 4.1.2。

4.2 Python SDK

请首先参照 3.1 节步骤安装 odrivetool（pip install –upgrade odrive）。请参见 3.1.8，可利用该小节描述的所有指令，来进行 python 开发。

下面是三个示例：

```
import odrive
import time

odrv0 = odrive.find_any()
odrive.utils.dump_errors(odrv0)
odrv0.clear_errors()
odrv0.axis0.requested_state=odrive.utils.AxisState.MOTOR_CALIBRATION
time.sleep(5)
while (odrv0.axis0.current_state!=1):
  time.sleep(0.5)
odrive.utils.dump_errors(odrv0)
odrv0.axis0.requested_state=odrive.utils.AxisState.ENCODER_OFFSET_CALI
BRATION
time.sleep(6)
while (odrv0.axis0.current_state!=1):
  time.sleep(0.5)
odrive.utils.dump_errors(odrv0)
odrv0.axis0.motor.config.pre_calibrated=1
odrv0.axis0.encoder.config.pre_calibrated=1
odrv0.save_configuration()
```

4.2.1 实战：上电校准

```
import odrive
import time

odrv0 = odrive.find_any()
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.VE
LOCITY_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.VEL_RA
MP
odrv0.axis0.controller.config.vel_ramp_rate=50
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_vel=15
odrive.utils.dump_errors(odrv0)
time.sleep(5)
odrv0.axis0.controller.input_vel=0
```

4.2.2 实战：速度控制

```
import odrive

odrv0 = odrive.find_any()
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.PO
SITION_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.POS_FI
LTER
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_pos=10
```

4.2.3 实战：位置控制

4.2.4 实战：数据采集

用户在研发集成过程中，经常需要采集电机运行的数据，比如采集电压电流变化，位置速度变化等等，Python SDK 集成了强大的数据抓取能力，可以用简单的脚本实现海量运行数据抓取，让研发和集成变得更加简单。

下面的代码是在上一个位置控制实例的基础上，增加了实时位置和电流数据抓取的功能，并

```
import odrive
import numpy as np

odrv0 = odrive.find_any()
cap =
odrive.utils.BulkCapture(lambda:[odrv0.axis0.motor.current_control.Iq_
measured,odrv0.axis0.encoder.pos_estimate],data_rate=500,duration=2.5)
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.PO
SITION_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.POS_FI
LTER
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_pos=10
np.savetxt("d:/test.csv",cap.data,delimiter=',')
```

将数据保存成一个 csv 文件。

其中进行 BulkCapture 的语句中，data_rate 代表采样频率，单位是 hz，duration 代表采样时间，单位是秒，其中的 lambda 表达式可以插入任何的数学运算，以方便数据分析。

4.3 Arduino SDK

用户使用 Arduino 可通过 CAN 总线来控制电机，底层协议如 4.1 中描述。可兼容硬件/库：

- ✓ 内置有 CAN 接口的 Arduino，如 Arduino UNO R4 Minima，Arduino UNO R4 WIFI 等
- ✓ 内置 CAN 接口的 Teensy 开发板可使用适配的 FlexCAN_T4 库（Teensy 4.0 和 Teensy 4.1）进行访问
- ✓ 其他 Arduino 兼容的板可使用基于 MCP2515 的 CAN 扩展板进行访问

下面是一个示例，展示如何配置电机响应 Arduino 的位置控制指令：

- ➢ 配置电机

除了 3.1.3 的基础配置外，将控制配置成控制带宽为 20rad/s（Arduino Uno 的发送速度受限，所以控制带宽不用太高，如果用更快的 Arduino，可提高此带宽值）：

- ➢ 配置 CAN

按照下述方法配置 CAN：

- ➢ 安装 ODriveArduino 库

按照下述步骤安装 OdriveArduino 库（假设用户已经安装 Arduino IDE）：

1) 打开 Arduino IDE
2) Sketch -> Include Library -> Manage Libraries
3) 输入"ODriveArduino"进行搜索
4) 点击搜索到的 ODriveArduino 库进行安装

- Arduino 源码

```
#include <Arduino.h>
#include "ODriveCAN.h"

// Documentation for this example can be found here:
// https://docs.odriverobotics.com/v/latest/guides/arduino-can-guide.html



/* Configuration of example sketch -------------------------------------------*/

// CAN bus baudrate. Make sure this matches for every device on the bus
#define CAN_BAUDRATE 500000

// ODrive node_id for odrv0
#define ODRV0_NODE_ID 0

// Uncomment below the line that corresponds to your hardware.
// See also "Board-specific settings" to adapt the details for your hardware setup.

// #define IS_TEENSY_BUILTIN // Teensy boards with built-in CAN interface (e.g. Teensy
4.1). See below to select which interface to use.
// #define IS_ARDUINO_BUILTIN // Arduino boards with built-in CAN interface (e.g.
Arduino Uno R4 Minima)
// #define IS_MCP2515 // Any board with external MCP2515 based extension module. See
below to configure the module.



/* Board-specific includes ---------------------------------------------------*/

#if defined(IS_TEENSY_BUILTIN) + defined(IS_ARDUINO_BUILTIN) +
defined(IS_MCP2515) != 1
#warning "Select exactly one hardware option at the top of this file."

#if CAN_HOWMANY > 0 || CANFD_HOWMANY > 0
#define IS_ARDUINO_BUILTIN
#warning "guessing that this uses HardwareCAN"
#else
#error "cannot guess hardware version"
```

```
#endif

#endif

#ifdef IS_ARDUINO_BUILTIN
// See https://github.com/arduino/ArduinoCore-API/blob/master/api/HardwareCAN.h
// and
https://github.com/arduino/ArduinoCore-renesas/tree/main/libraries/Arduino_CAN

#include <Arduino_CAN.h>
#include <ODriveHardwareCAN.hpp>
#endif // IS_ARDUINO_BUILTIN

#ifdef IS_MCP2515
// See https://github.com/sandeepmistry/arduino-CAN/
#include "MCP2515.h"
#include "ODriveMCPCAN.hpp"
#endif // IS_MCP2515

#ifdef IS_TEENSY_BUILTIN
// See https://github.com/tonton81/FlexCAN_T4
// clone https://github.com/tonton81/FlexCAN_T4.git into /src
#include <FlexCAN_T4.h>
#include "ODriveFlexCAN.hpp"
struct ODriveStatus; // hack to prevent teensy compile error
#endif // IS_TEENSY_BUILTIN




/* Board-specific settings ------------------------------------------------------*/


/* Teensy */

#ifdef IS_TEENSY_BUILTIN

FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> can_intf;

bool setupCan() {
  can_intf.begin();
  can_intf.setBaudRate(CAN_BAUDRATE);
  can_intf.setMaxMB(16);
  can_intf.enableFIFO();
```

```
  can_intf.enableFIFOInterrupt();
  can_intf.onReceive(onCanMessage);
  return true;
}

#endif // IS_TEENSY_BUILTIN


/* MCP2515-based extension modules -*/

#ifdef IS_MCP2515

MCP2515Class& can_intf = CAN;

// chip select pin used for the MCP2515
#define MCP2515_CS 10

// interrupt pin used for the MCP2515
// NOTE: not all Arduino pins are interruptable, check the documentation for your board!
#define MCP2515_INT 2

// freqeuncy of the crystal oscillator on the MCP2515 breakout board.
// common values are: 16 MHz, 12 MHz, 8 MHz
#define MCP2515_CLK_HZ 8000000


static inline void receiveCallback(int packet_size) {
  if (packet_size > 8) {
    return; // not supported
  }
  CanMsg msg = {.id = (unsigned int)CAN.packetId(), .len = (uint8_t)packet_size};
  CAN.readBytes(msg.buffer, packet_size);
  onCanMessage(msg);
}

bool setupCan() {
  // configure and initialize the CAN bus interface
  CAN.setPins(MCP2515_CS, MCP2515_INT);
  CAN.setClockFrequency(MCP2515_CLK_HZ);
  if (!CAN.begin(CAN_BAUDRATE)) {
    return false;
  }

  CAN.onReceive(receiveCallback);
```

```
  return true;
}

#endif // IS_MCP2515


/* Arduinos with built-in CAN */

#ifdef IS_ARDUINO_BUILTIN

HardwareCAN& can_intf = CAN;

bool setupCan() {
  return can_intf.begin((CanBitRate)CAN_BAUDRATE);
}

#endif


/* Example sketch ------------------------------------------------------------*/

// Instantiate ODrive objects
ODriveCAN odrv0(wrap_can_intf(can_intf), ODRV0_NODE_ID); // Standard CAN message ID
ODriveCAN* odrives[] = {&odrv0}; // Make sure all ODriveCAN instances are accounted
for here

struct ODriveUserData {
  Heartbeat_msg_t last_heartbeat;
  bool received_heartbeat = false;
  Get_Encoder_Estimates_msg_t last_feedback;
  bool received_feedback = false;
};

// Keep some application-specific user data for every ODrive.
ODriveUserData odrv0_user_data;

// Called every time a Heartbeat message arrives from the ODrive
void onHeartbeat(Heartbeat_msg_t& msg, void* user_data) {
  ODriveUserData* odrv_user_data = static_cast<ODriveUserData*>(user_data);
  odrv_user_data->last_heartbeat = msg;
  odrv_user_data->received_heartbeat = true;
}

// Called every time a feedback message arrives from the ODrive
```

```
void onFeedback(Get_Encoder_Estimates_msg_t& msg, void* user_data) {
  ODriveUserData* odrv_user_data = static_cast<ODriveUserData*>(user_data);
  odrv_user_data->last_feedback = msg;
  odrv_user_data->received_feedback = true;
}

// Called for every message that arrives on the CAN bus
void onCanMessage(const CanMsg& msg) {
  for (auto odrive: odrives) {
    onReceive(msg, *odrive);
  }
}

void setup() {
  Serial.begin(115200);

  // Wait for up to 3 seconds for the serial port to be opened on the PC side.
  // If no PC connects, continue anyway.
  for (int i = 0; i < 30 && !Serial; ++i) {
    delay(100);
  }
  delay(200);


  Serial.println("Starting ODriveCAN demo");

  // Register callbacks for the heartbeat and encoder feedback messages
  odrv0.onFeedback(onFeedback, &odrv0_user_data);
  odrv0.onStatus(onHeartbeat, &odrv0_user_data);

  // Configure and initialize the CAN bus interface. This function depends on
  // your hardware and the CAN stack that you're using.
  if (!setupCan()) {
    Serial.println("CAN failed to initialize: reset required");
    while (true); // spin indefinitely
  }

  Serial.println("Waiting for ODrive...");
  while (!odrv0_user_data.received_heartbeat) {
    pumpEvents(can_intf);
    delay(100);
  }

  Serial.println("found ODrive");
```

```
  // request bus voltage and current (1sec timeout)
  Serial.println("attempting to read bus voltage and current");
  Get_Bus_Voltage_Current_msg_t vbus;
  if (!odrv0.request(vbus, 1)) {
    Serial.println("vbus request failed!");
    while (true); // spin indefinitely
  }

  Serial.print("DC voltage [V]: ");
  Serial.println(vbus.Bus_Voltage);
  Serial.print("DC current [A]: ");
  Serial.println(vbus.Bus_Current);

  Serial.println("Enabling closed loop control...");
  while (odrv0_user_data.last_heartbeat.Axis_State !=
ODriveAxisState::AXIS_STATE_CLOSED_LOOP_CONTROL) {
    odrv0.clearErrors();
    delay(1);
    odrv0.setState(ODriveAxisState::AXIS_STATE_CLOSED_LOOP_CONTROL);

    // Pump events for 150ms. This delay is needed for two reasons;
    // 1. If there is an error condition, such as missing DC power, the ODrive might
    //    briefly attempt to enter CLOSED_LOOP_CONTROL state, so we can't rely
    //    on the first heartbeat response, so we want to receive at least two
    //    heartbeats (100ms default interval).
    // 2. If the bus is congested, the setState command won't get through
    //    immediately but can be delayed.
    for (int i = 0; i < 15; ++i) {
      delay(10);
      pumpEvents(can_intf);
    }
  }

  Serial.println("ODrive running!");
}

void loop() {
  pumpEvents(can_intf); // This is required on some platforms to handle incoming
feedback CAN messages

  float SINE_PERIOD = 2.0f; // Period of the position command sine wave in seconds

  float t = 0.001 * millis();
```

```
  float phase = t * (TWO_PI / SINE_PERIOD);

  odrv0.setPosition(
    sin(phase), // position
    cos(phase) * (TWO_PI / SINE_PERIOD) // velocity feedforward (optional)
  );

  // print position and velocity for Serial Plotter
  if (odrv0_user_data.received_feedback) {
    Get_Encoder_Estimates_msg_t feedback = odrv0_user_data.last_feedback;
    odrv0_user_data.received_feedback = false;
    Serial.print("odrv0-pos:");
    Serial.print(feedback.Pos_Estimate);
    Serial.print(",");
    Serial.print("odrv0-vel:");
    Serial.println(feedback.Vel_Estimate);
  }
}
```

4.4 ROS SDK

以下步骤在 Ubuntu 23.04 和 ROS2 Iron 上经过测试验证，但不支持 MAC 和 Windows 平台，且在其他 ROS2 版本上未验证，需要经过修正后才可使用。

4.4.1 安装 odrive_can 包

1. 新建一个 ROS2 workspace（请参见 https://docs.ros.org/en/iron/index.html）
2. 用 git clone https://github.com/odriverobotics/odrive_can 将代码下载到上述 workspace 目录中的 src 目录

```
colcon build --packages-select odrive_can
```

3. 在 terminal 中转到 workspace 的根目录，并运行：
4. 运行之前的环境准备：
5. 运行例程节点：

```
source ./install/setup.bash
```

```
ros2 launch odrive_can example_launch.yaml
```

4.4.2 调用服务和查看消息

假设上述 odrive_can_node 节点运行于命名空间 odrive_axis0（可在./launch/example_launch.yaml 中设置）。一旦 4.4.1 中的节点运行起来过后，就可以查看所公布的话题消息，如：

```
ros2 topic echo /odrive_axis0/controller_status
ros2 topic echo /odrive_axis0/odrive_status
```

```
ros2 service call /odrive_axis0/request_axis_state
/odrive_can/srv/AxisState "{axis_requested_state: 4}"
```

并调用开放的服务接口，如下述调用可开始进行电机校准：

5 常见问题和异常码（待更新）

5.1 常见问题（FAQ）

5.2 异常码

错误类别	错误码	odrivetool 显示	描述
系统异常	0x00000002	DC_BUS_UNDER_VOLTAGE	电源电压过低
	0x00000004	DC_BUS_OVER_VOLTAGE	电源电压过高
	0x00000008	DC_BUS_OVER_REGEN_CURRENT	电源反向（充电）电流过高
	0x00000010	DC_BUS_OVER_CURRENT	电源正向（放电）电流过高
驱动异常	0x00000001	INVALID_STATE	驱动器状态错误
	0x00000040	MOTOR_FAILED	电机异常
	0x00000100	ENCODER_FAILED	编码器异常
	0x00000200	CONTROLLER_FAILED	控制器异常
	0x00001000	MIN_ENDSTOP_PRESSED	低限位触发
	0x00002000	MAX_ENDSTOP_PRESSED	高限位触发
	0x00004000	ESTOP_REQUESTED	紧急停止
	0x00020000	HOMING_WITHOUT_ENDSTOP	回零但没有限位开关
	0x00080000	UNKNOWN_POSITION	无位置信息
电机异常	0x00000001	PHASE_RESISTANCE_OUT_OF_RANGE	相间电阻超出正常范围
	0x00000002	PHASE_INDUCTANCE_OUT_OF_RANGE	相间电感超出正常范围
	0x00000010	CONTROL_DEADLINE_MISSED	FOC 频率太高
	0x00000080	MODULATION_MAGNITUDE	SVM 调制异常

	0x00000400	CURRENT_SENSE_SATURATION	相电流饱和
	0x00001000	CURRENT_LIMIT_VIOLATION	电机电流过大
	0x00020000	MOTOR_THERMISTOR_OVER_TEMP	电机温度过高
	0x00040000	FET_THERMISTOR_OVER_TEMP	驱动器温度过高
	0x00080000	TIMER_UPDATE_MISSED	FOC 处理不及时
	0x00100000	CURRENT_MEASUREMENT_UNAVAILABLE	相电流采样丢失
	0x00200000	CONTROLLER_FAILED	控制异常
	0x00400000	I_BUS_OUT_OF_RANGE	母线电流超限
	0x00800000	BRAKE_RESISTOR_DISARMED	泄放电阻驱动异常
	0x01000000	SYSTEM_LEVEL	系统级异常
	0x02000000	BAD_TIMING	相电流采样不及时
	0x04000000	UNKNOWN_PHASE_ESTIMATE	电机位置未知
	0x08000000	UNKNOWN_PHASE_VEL	电机速度未知
	0x10000000	UNKNOWN_TORQUE	力矩未知
	0x20000000	UNKNOWN_CURRENT_COMMAND	力矩控制未知
	0x40000000	UNKNOWN_CURRENT_MEASUREMENT	电流采样值未知
	0x80000000	UNKNOWN_VBUS_VOLTAGE	电压采样值未知
	0x100000000	UNKNOWN_VOLTAGE_COMMAND	电压控制未知
	0x200000000	UNKNOWN_GAINS	电流环增益未知
	0x400000000	CONTROLLER_INITIALIZING	控制器初始化异常
	0x800000000	UNBALANCED_PHASES	三相不平衡
控制异常	0x00000001	OVERSPEED	速度过高
	0x00000002	INVALID_INPUT_MODE	控制输入模式不正确
	0x00000004	UNSTABLE_GAIN	锁相环增益不稳
	0x00000020	INVALID_ESTIMATE	位置/速度不稳定
		SPINOUT_DETECTED	机械功率和电气功率不匹配（编码器校准不正确，或磁钢不稳）