TSL-GIM6010-8 Joint Module User Manual

Table of Contents

Revision History

Version	Date	Revision / Description
0.1	2023.12.05	Initial version supporting odrivetool
0.2	2023.12.15	Added command list and command classification
0.3	2023.12.19	Added Python, Arduino and ROS SDK
0.4	2023.12.23	Added description for USB and CAN compatibility switching
0.5	2023.12.29	Added practical commissioning cases via upper computer
0.6	2024.01.02	Added CAN protocol and Python commissioning practical cases
0.7	2024.01.08	Added 120Ω termination resistor switch option for CAN interface
0.8	2024.02.15	Added contents related to secondary encoder and user zero position setting
0.9	2024.02.23	Added CAN commands for editable parameters and calling interface functions
0.91	2024.03.12	Added driver download link for national chip download tool
0.92	2024.03.22	Added description of default CAN baud rate and rotor initial position range with secondary encoder enabled
1.0	2024.03.26	Added motor over-temperature protection description
1.1	2024.05.20	Added commissioning quick start guide
1.2	2024.05.24	Corrected errors in CAN MIT protocol
1.3	2024.07.02	Added error code table
1.4	2024.08.18	Added operation instructions for ID modification
1.5	2024.08.21	Added braking resistor usage instructions
1.6	2024.08.31	Added download links for endpoints.json of different versions
1.7	2024.09.05	Added CANOpen protocol specification
1.8	2024.10.08	Reorganized fault types and added encoder fault code list
1.9	2024.10.15	Added operation guide for upper computer software "Motor Wizard"
2.0	2024.12.06	Added Modbus RTU protocol specification and C/C++ SDK
2.1	2025.01.23	Added description of Modbus random endpoint access
2.2	2025.05.20	Added descriptions for PWM control and pulse-direction control

Precautions

1. Operate the product only within the working parameters specified in this manual; otherwise, irreversible damage may occur.

2. Provide overcurrent and overvoltage protection for the power supply during motor operation to prevent damage to the driver.

3. Before use, check that all components are complete and undamaged. Contact technical support promptly if any part is missing or damaged.

4. The driver has no reverse-polarity protection. Before connecting power, refer to Section 2.4.1 and verify the positive and negative terminals.

5. Do not touch exposed parts of the driver, as electrostatic discharge may damage the device.

Legal Disclaimer

Read this manual before use and follow all instructions. The Company is not liable for property damage or personal injury caused by improper use. Keep the product away from children and do not use it in high-temperature, high-pressure, or other unsuitable environments. Product functions, design, appearance, specifications, software, and manual content may be updated without prior notice. The actual product shall prevail.

The Company may correct errors and revise this manual without prior notice. Contact technical support for the latest version. All images are for reference only.

After-Sales Service Policy

After-sales service is provided in accordance with the applicable laws and regulations of the People's Republic of China.

1. Warranty Period and Coverage

   1) Products purchased online may be returned without reason within seven days from the day following receipt.

      Valid proof of purchase is required. Return the product with its invoice, original packaging, accessories, labels, and complimentary items, complete and undamaged. Returns are not accepted for customer-caused damage, unauthorized disassembly, or missing items. The customer bears return shipping. Refunds will be issued through the original payment method within seven days after receipt; bank or payment-provider processing times may vary.

   2) A non-user-caused performance fault within seven days may be returned after inspection and confirmation. Proof of purchase, the invoice, and any complimentary items must be provided.

   3) A non-user-caused performance fault within 7–15 days may be replaced after inspection and confirmation. The replacement warranty restarts from the replacement date.

4) If a product-quality fault occurs between 15 and 365 days from the day following receipt, free repair service will be provided after inspection and confirmation by the Company's after-sales service center. Replaced defective products become the property of the Company; products found fault-free will be returned unchanged. The Company may reject return or replacement requests where the fault is unrelated to product quality.

If this policy differs from the policy of the relevant store, the store policy shall prevail.

2. Warranty Exclusions

The following circumstances are not covered by the warranty:

1) The warranty period has expired.

2) Damage caused by misuse or failure to follow this manual.

3) Damage caused by improper operation, repair, installation, modification, testing, or other unauthorized use.

4) Normal mechanical deterioration or wear not caused by a quality defect.

5) Damage caused by abnormal operating conditions, including dropping, collision, liquid ingress, or severe impact.

6) Damage caused by natural disasters, such as flood, fire, lightning, earthquake, or other force majeure events.

7) Damage caused by operation above the peak torque limit.

8) Products that are not genuine Company products or for which valid proof of purchase cannot be provided.

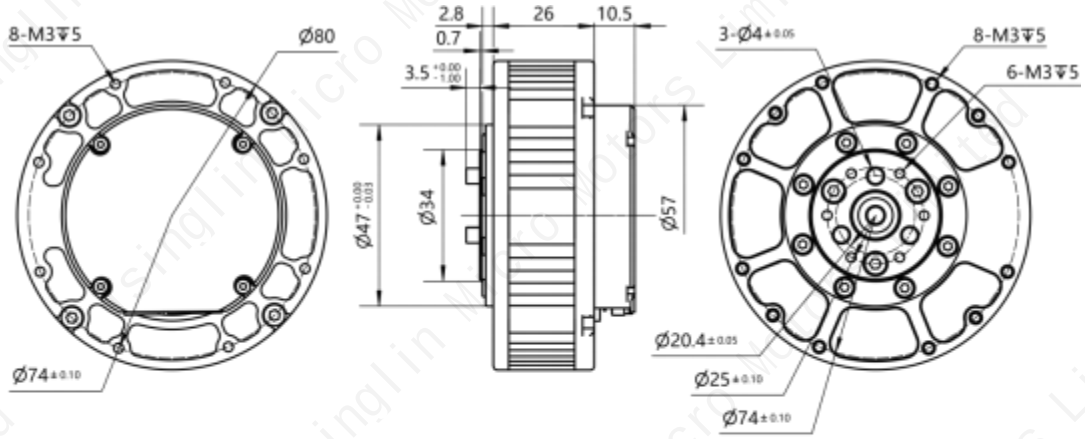
9) Faults or damage unrelated to product design, technology, manufacturing, or quality.

10) Damage caused by unauthorized disassembly.

In any of the above cases, the customer shall bear the applicable costs.

1 Motor Specifications

1.1 Drawing and Dimensions

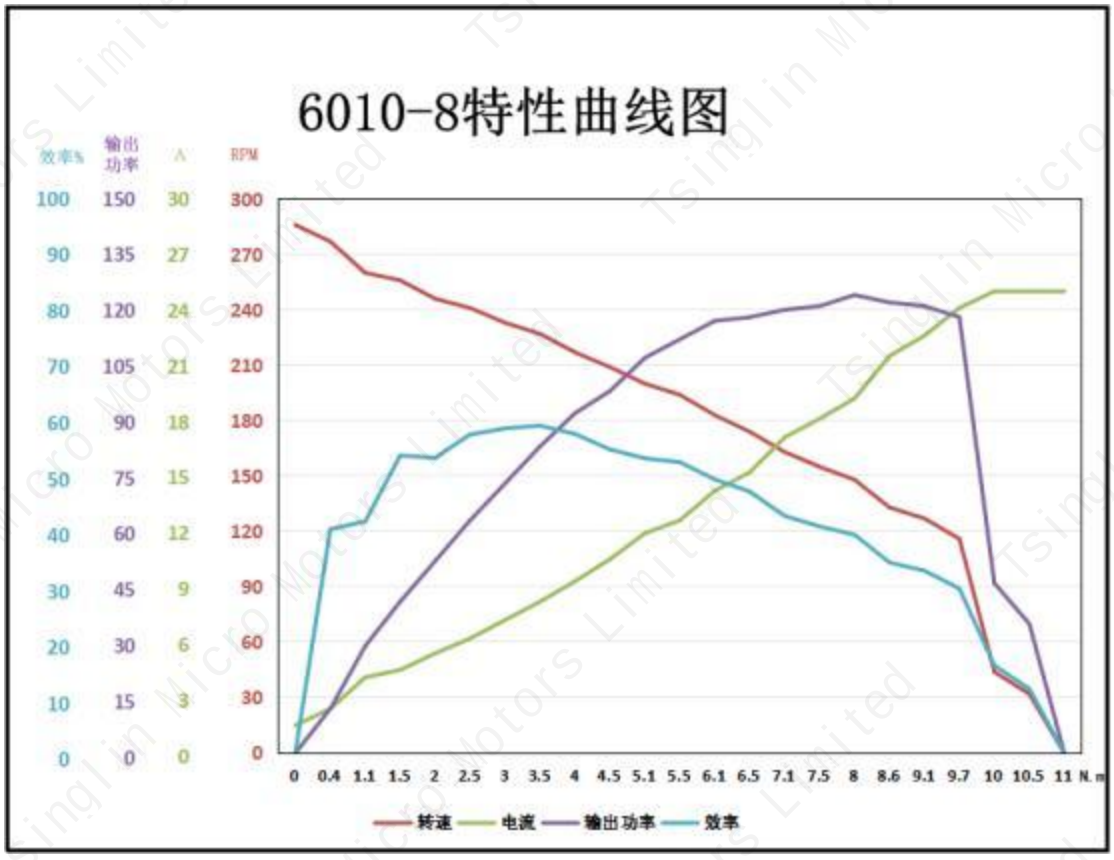


未注公差分段						
<=6	>6<=30	>30<=120	>120<=400	>400<=2000	比例	1 : 1	6010-8安装尺寸图
±0.1	±0.2	±0.3	±0.5	±1.0	重量kg	无	

1.2 Electrical Characteristics

Rated Speed	120rpm±10%
Maximum Speed	420rpm±10%
Rated Torque	5N.m
Stall Torque	11N.m
Rated Current	10.5A
Stall Current	25A
No-Load Current	0.4A
Insulation Resistance / Stator Winding	DC 500VAC, 100M Ohms
Withstand Voltage / Stator to Housing	600 VAC, 1s, 2mA
Motor Back EMF	0.054~0.057Vrms/rpm
Phase-to-Phase Resistance	0.48Ω±10%
Phase-to-Phase Inductance	368μH±10%
Speed Constant	12.3rpm/v
Torque Constant	0.47N.M/A

The characteristic curves are shown below:



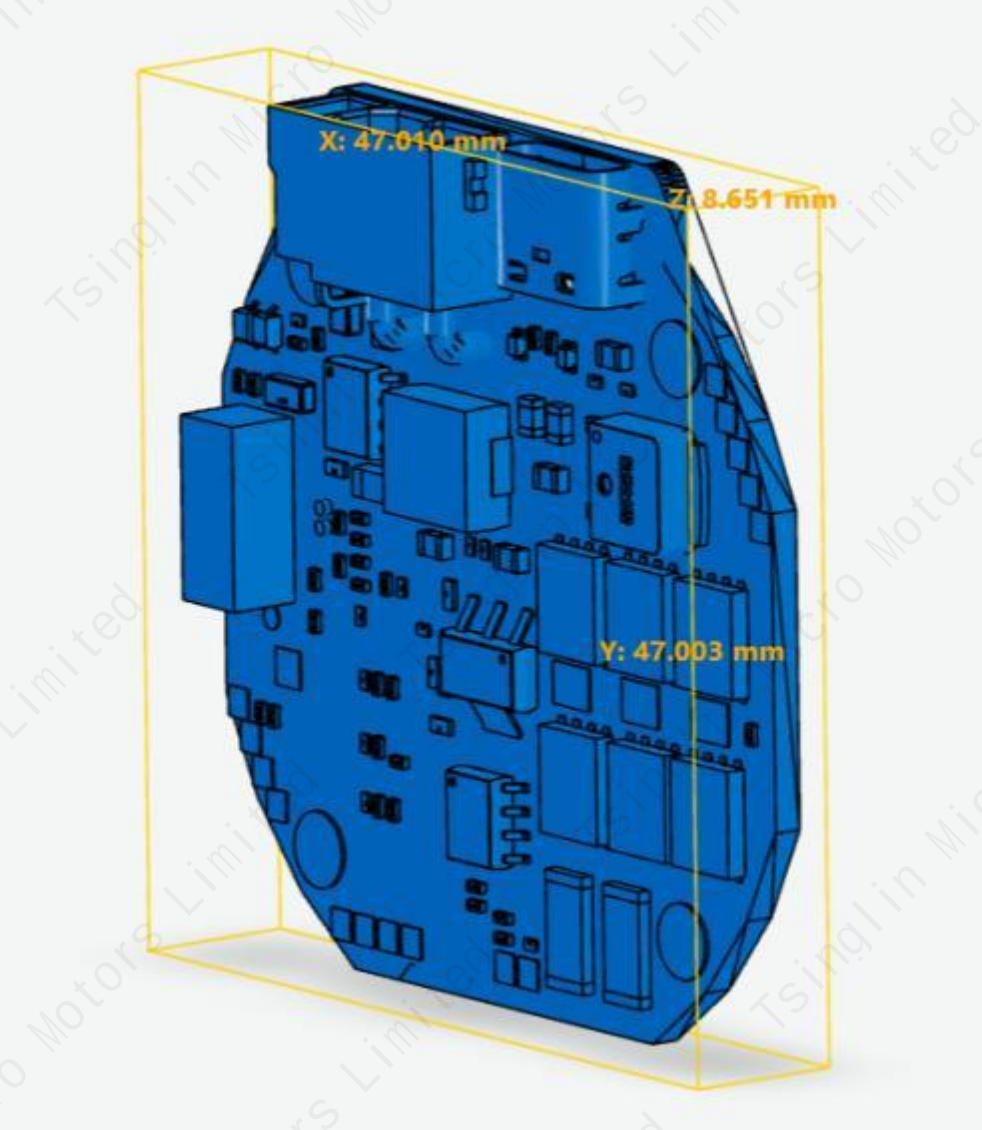
1.3 Mechanical Characteristics

Weight	388g±3

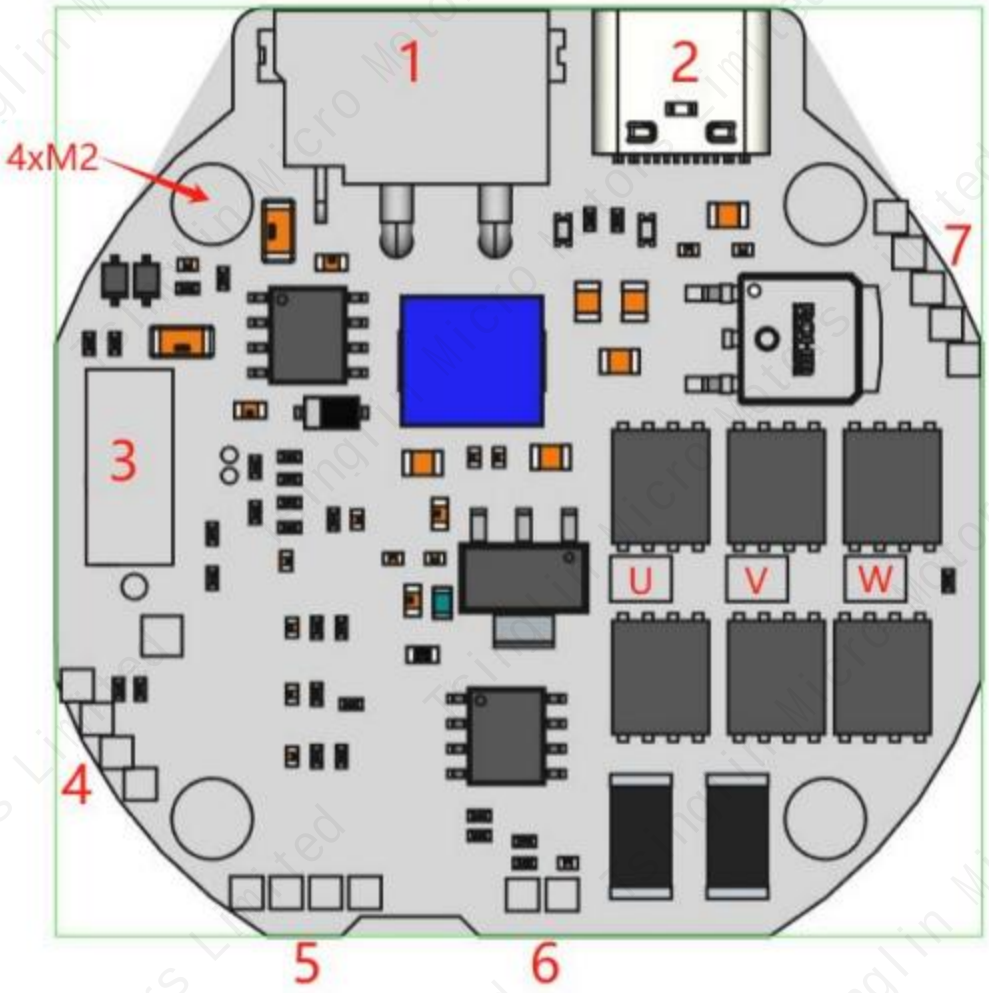
Pole Pairs	14 pairs
Number of Phases	3 phases
Drive Method	FOC
Gear Ratio	8:1

2 Driver Information

2.1 Appearance and 3D Dimensions



2.2 Interface Overview



Interface No.	Definition
1	15-60 V power and CAN communication integrated terminal
2	Type-C debugging and host-computer communication interface
3	Interface expansion slot (supports RS485, EtherCAT, RC, pulse/direction, throttle control, and other interfaces/protocols)
4	SWD debugging and firmware download interface
5	Secondary encoder interface (supports I2C and UART)
6	Motor temperature interface (NTC)
7	Holding-brake/braking-resistor interface, 12 V supply, and minimum/maximum limit-switch interfaces
U/V/W	Three-phase winding solder pads
4xM2	Mounting Holes

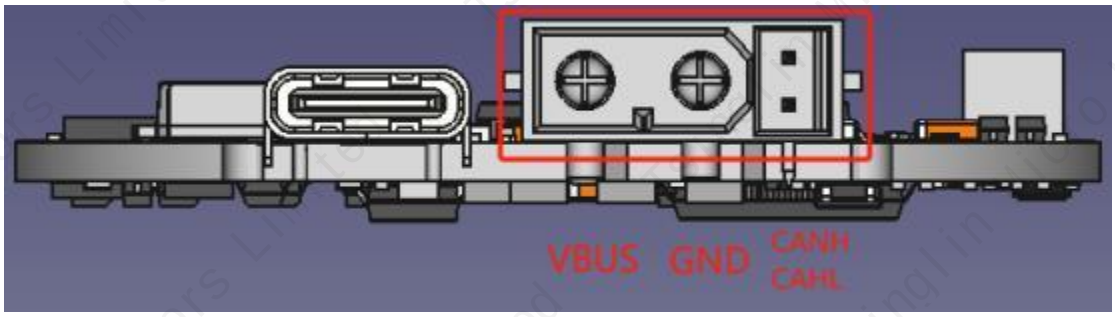
2.3 Specifications

Rated Voltage	15~48V DC

Minimum/Maximum Voltage	12/72V DC
Rated Current	6A
Maximum Line Current	30A
Maximum Phase Current	120A
Standby Power Consumption	<10mA
Maximum CAN Bus Baud Rate	1Mbps
Type-C Data Rate	10Mbps
Encoder Resolution	14-bit (single-turn absolute)
Operating Ambient Temperature	-20°C to 70°C
Motor Overtemperature Warning	90°C (adjustable)
Driver-Board Overtemperature Warning	90°C (adjustable)

2.4 Detailed Interface Definitions

2.4.1 Power and CAN/RS485/PWM Terminal



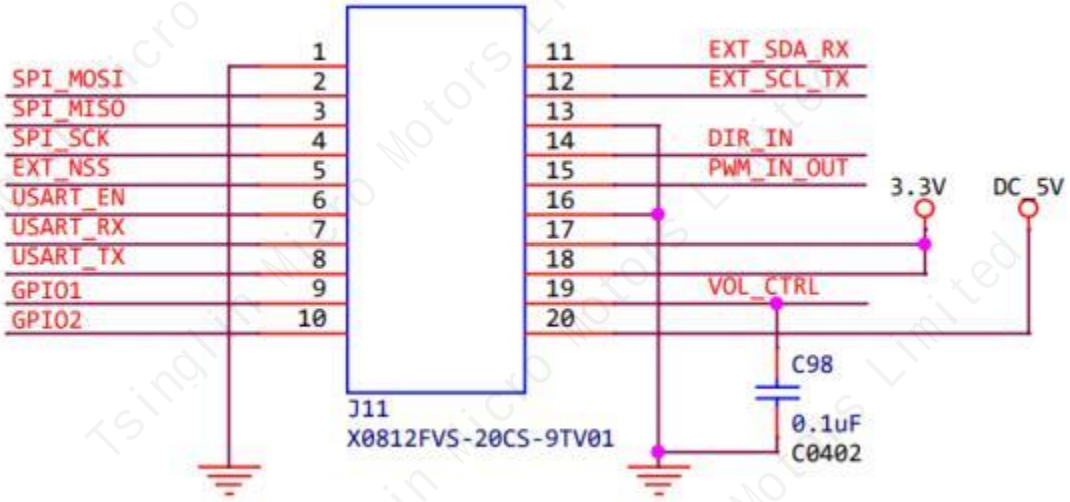
The onboard connector is XT30PB(2+2)-M and the mating cable connector is XT30(2+2)-F, both manufactured by AMASS. In PWM mode, CANH is the PWM signal input. In pulse mode, CANH is the pulse input and CANL is the direction input. On versions supporting RS485, CANH/CANL also serve as RS485 A/B.

2.4.2 Type-C Debugging Interface

The Type-C port uses a standard data-cable specification and is compatible with commonly used PC or mobile-phone Type-C cables.

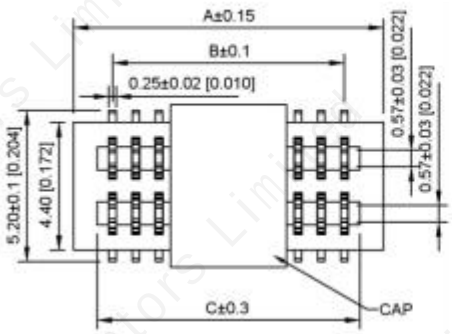
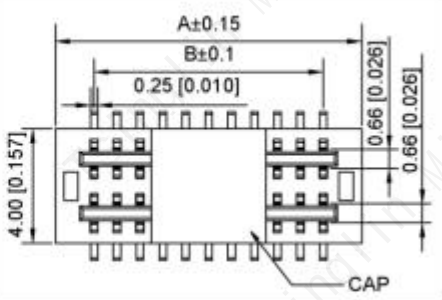
2.4.3 Interface Expansion Slot

This slot provides a wide range of board-to-board expansion interfaces, allowing third parties to develop custom expansion boards. Note: hardware version 3.9 and later no longer provide this expansion interface.



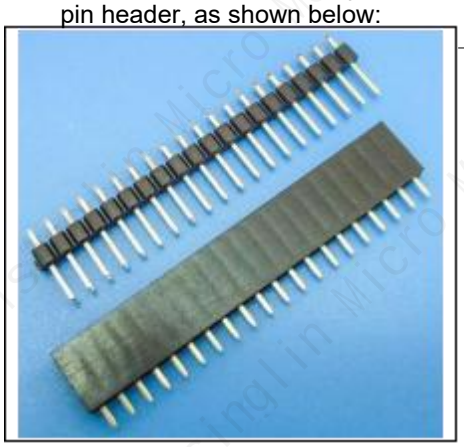
Third parties can interact with the driver through SPI, USART, I2C, PWM, ADC, GPIO, and other interfaces to implement various expansion functions.

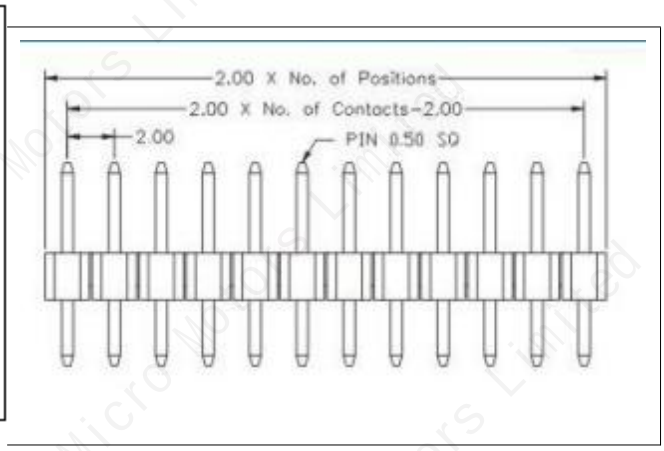
The onboard slot is X0812FVS-20CS-9TV01 (female), and the expansion-board slot is X0812WVS-20AS-9TV01 (male), manufactured by XKB Connectivity.

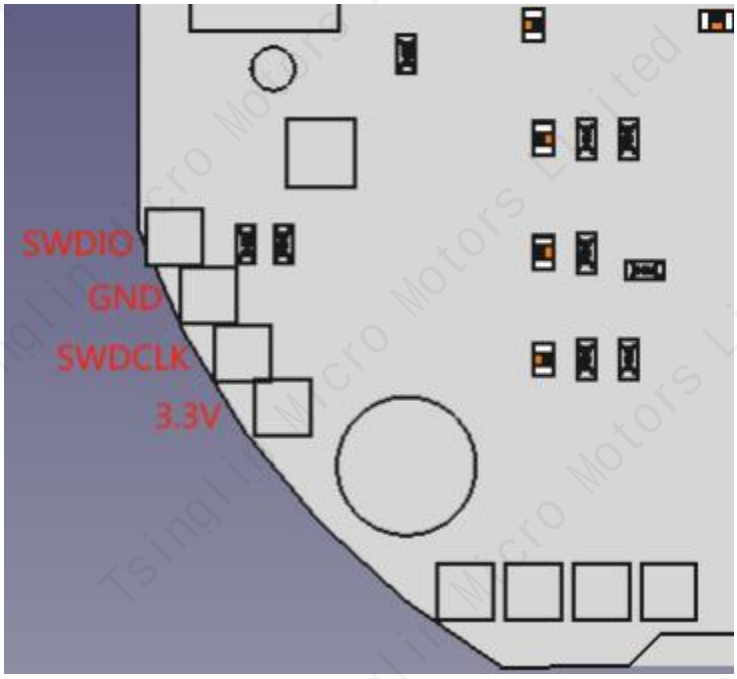
Female X0812FVS-20CS-	Male X0812WVS-20AS-
	

2.4.4 SWD Debugging Interface

The interface uses 2 mm-pitch through-holes. Users may solder a 2 mm straight single-row pin header, as shown below:



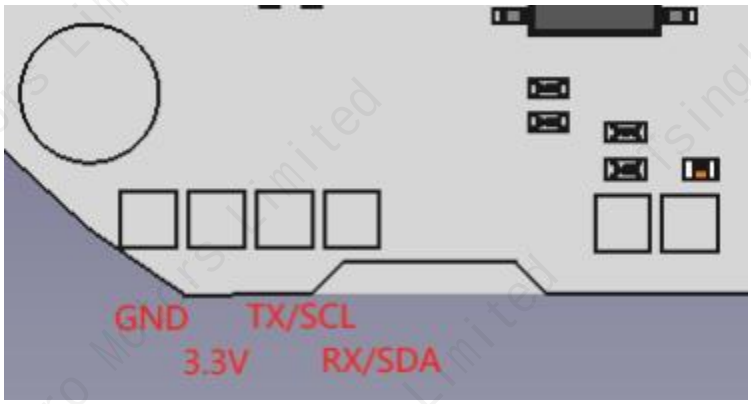




2.4.5 Secondary Encoder Interface

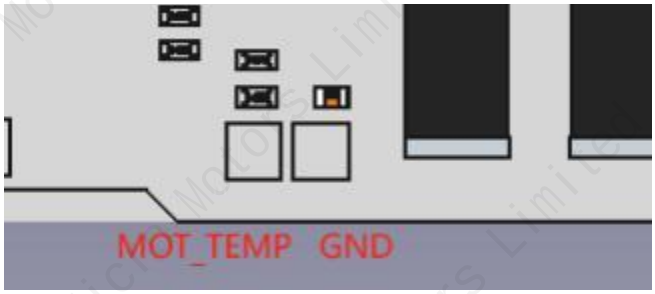
The interface uses 2 mm-pitch through-holes. Users may solder a 2 mm straight single-row pin header; see Section 2.4.4.

This interface communicates with a secondary encoder through USART (TX/RX) or I2C (SCL/SDA).



2.4.6 Motor Temperature Interface

The motor contains a built-in 10 kΩ NTC thermistor. Solder its two leads to MOT_TEMP and GND; polarity is not required.

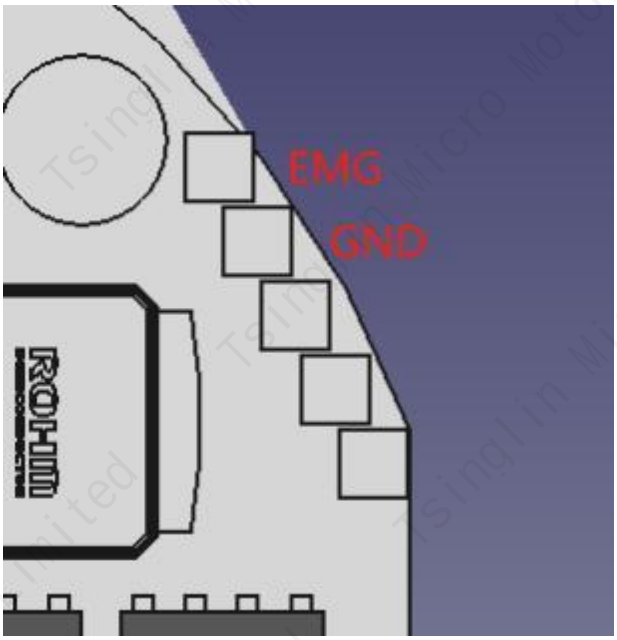

MOT_TEMP
GND

2.4.7 Holding-Brake/Braking-Resistor Interface

The upper two solder pads of the illustrated 5-pin interface are for the holding brake or braking resistor. The holes have a 2 mm pitch; users may solder a 2 mm straight single-row pin header. See Section 2.4.4.

When configured as a holding-brake interface, the driver continuously supplies current after power-on to release the brake and allow normal motor operation. If driver power is removed, the current stops, the brake engages, and the motor locks at its current position.

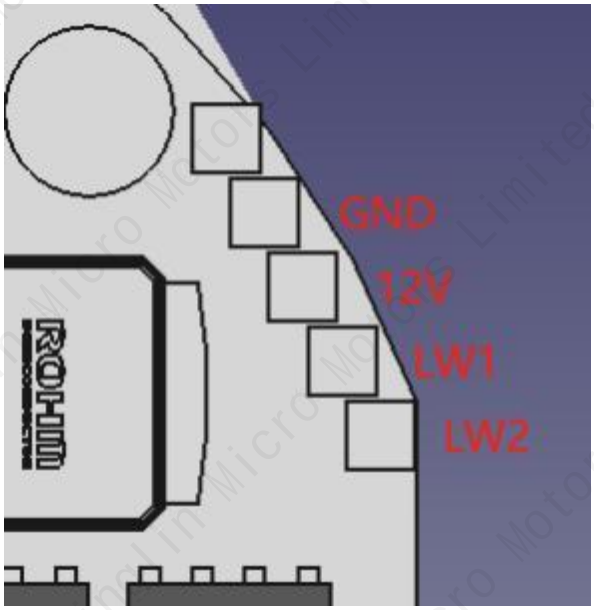
When configured as a braking-resistor interface, an external braking (discharge) resistor may be connected. If back EMF exceeds the threshold voltage, current is dissipated through the resistor to enable emergency braking and protect the driver from back-EMF damage.



2.4.8 Limit-Switch Interface

The driver provides two limit-switch interfaces and a 12 V supply for external limit switches. The interfaces use 2 mm-pitch through-holes; users may solder a 2 mm straight single-row pin header. See Section 2.4.4.

LW1 is the minimum-position limit switch and LW2 is the maximum-position limit switch. Two-wire switches or three-wire NPN switches may be connected.



2.5 Main Components and Specifications

No.	Component	Model/Specification	Quantity
1	MCU	N32G455	1
2	Driver IC	FD6288Q	1
3	Magnetic Encoder IC	MA732, 14-bit absolute	1
4	MOSFET	JMSH1004NG，100V/120A	6

3 Commissioning Instructions

3.1 Quick Start Guide

3.1.1 Preparation

To operate the motor, you will need:

✓ Power Supply

Refer to Chapter 1 for the supply-voltage requirements. A regulated power supply or battery is recommended. The following simple guidelines may help you select a suitable power source:

Key considerations when selecting a power supply:
u Current Requirement Generally, the supply should provide at least 5 A. The exact value depends on the system power demand and voltage.
u Voltage Requirement The required voltage depends on the motor Kv and the system's maximum required speed, RPMmax. The maximum supply voltage can be estimated using the following formula: $$V_{max} = \frac{RPM_{max}}{K_v} \times 1.25$$ The factor 1.25 is an empirical safety coefficient that provides voltage margin for the system.
u Power Requirement In simple terms, the power requirement depends on the maximum current Imax at the highest speed. Refer to the following formula: $$P_{max} = I_{max} \times \frac{RPM_{max}}{K_v} \times 1.25$$

✓ Power and Communication Cable

The 6010-8 uses a 2+2 power/communication connector. Contact after-sales support for a recommended 2+2 cable. Refer to Section 2.4.1 and ensure that the power polarity is correct, because the driver has no reverse-polarity protection and may be damaged. Also verify the order of the two communication wires, such as CANH/CANL; incorrect wiring will prevent normal communication.

Warning
u Do not touch the communication bus by hand. Electrostatic discharge may damage the driver interface IC, especially in dry environments or seasons. u Never connect or disconnect the power terminal while energized. u Do not use a circuit breaker as the power switch, as the inrush current may damage the driver's power IC. u Do not exceed a 72 V supply voltage.

✓ Type-C Data Cable

During initial commissioning, a USB Type-C data cable is strongly recommended for motor testing. A standard mobile-phone Type-C data cable may be used, but charge-only cables are not supported.

Note: the Type-C cable cannot power the driver or drive the motor.

✓ Power-On

First switch off the power supply, connect the power cable, and then switch on the power. Never hot-plug the connector or use a circuit breaker to switch only the positive or negative lead, as excessive inrush current may damage the driver.

After power-on, the USB Type-C cable may be connected or disconnected at any time.

3.1.2 Getting Started with Host Software

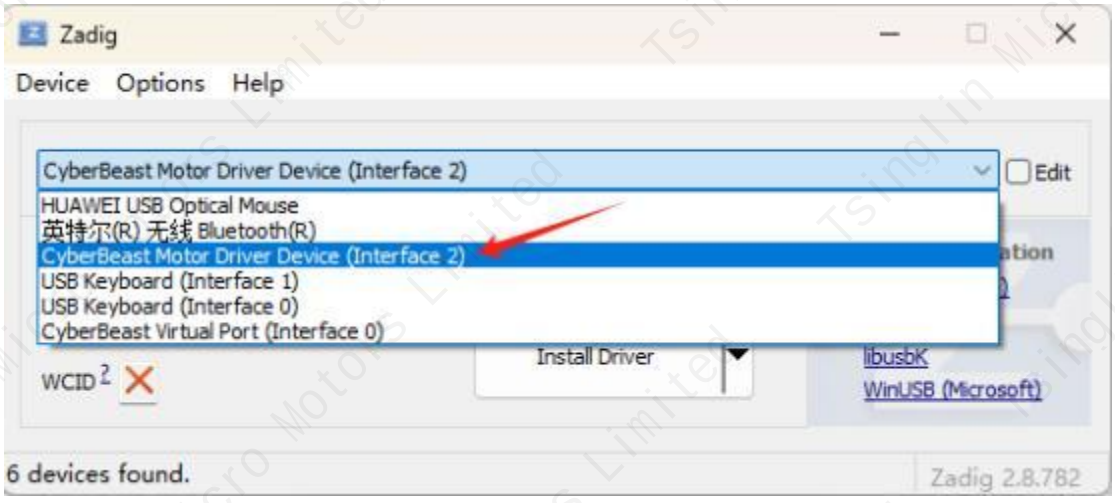
u Motor Wizard

1. Install Motor Wizard

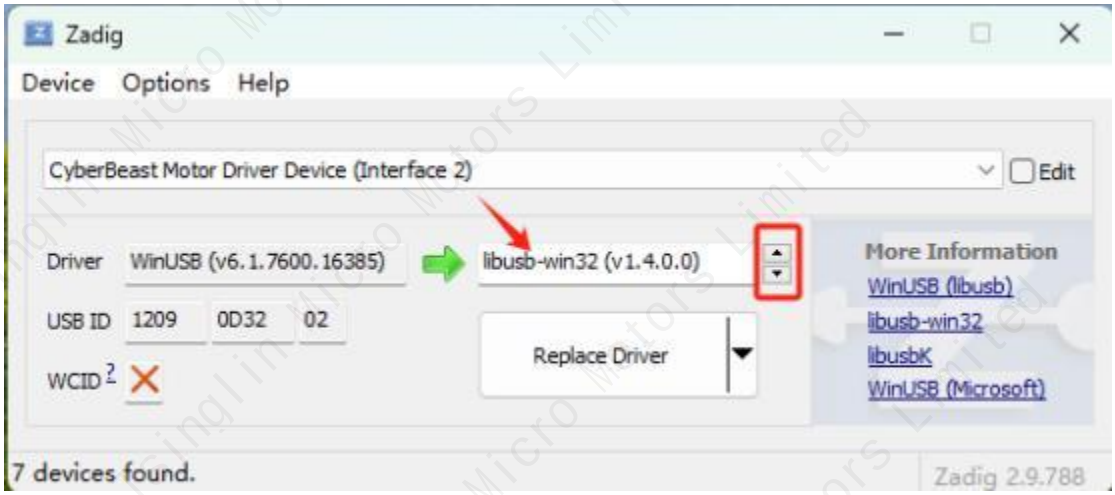
Download the Windows installer: (https://bl.cyberbeast.cn/actuator/steadywin_motorwizard.exe), and follow the default installation procedure.

2. Install the USB Driver

Go to https://zadig.akeo.ie and download Zadig. Connect the driver to the computer with a Type-C cable; the driver power indicator will illuminate. Open Zadig and select "CyberBeast Motor Driver Device (Interface 2)" from the drop-down list:



Use the up/down controls to select the USB driver. Choose the "libusb" driver for this interface, then click "Install Driver" or "Replace Driver":



◆ *odrivetool*

The driver is compatible with ODrive (https://github.com/odriverobotics/odrive.git), so odrivetool can be used as the host commissioning tool.

Install odrivetool as follows:

➢ **Windows**

1. Install Python

Go to https://www.python.org, download the latest official Python installer, and follow the prompts. Do not install Python from third-party websites or the Microsoft Store.

2. Install the Visual C++ Build Tools

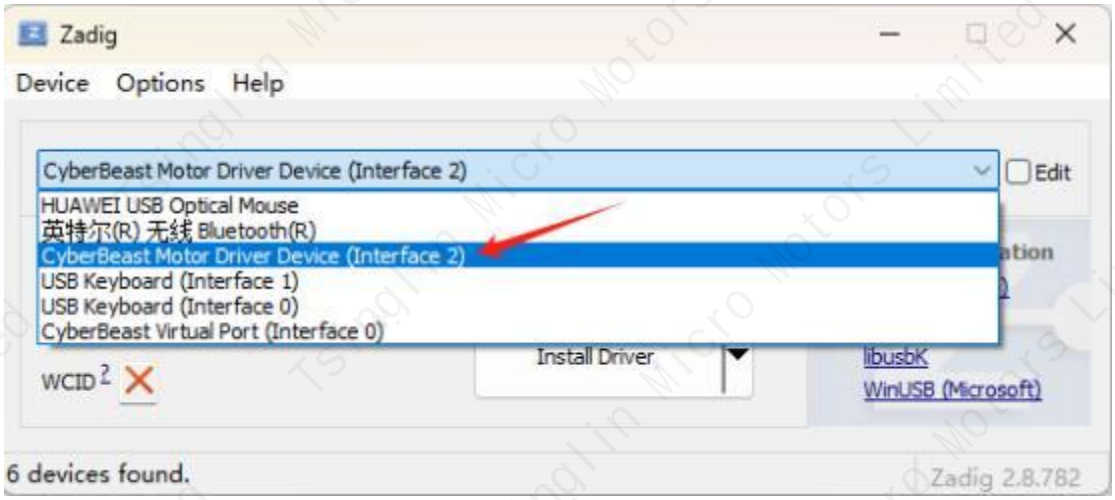
Install the Visual C++ Build Tools from https://visualstudio.microsoft.com/visual-cpp-build-tools/. During installation, select "Desktop development with C++", as shown below.

3. Install odrivetool

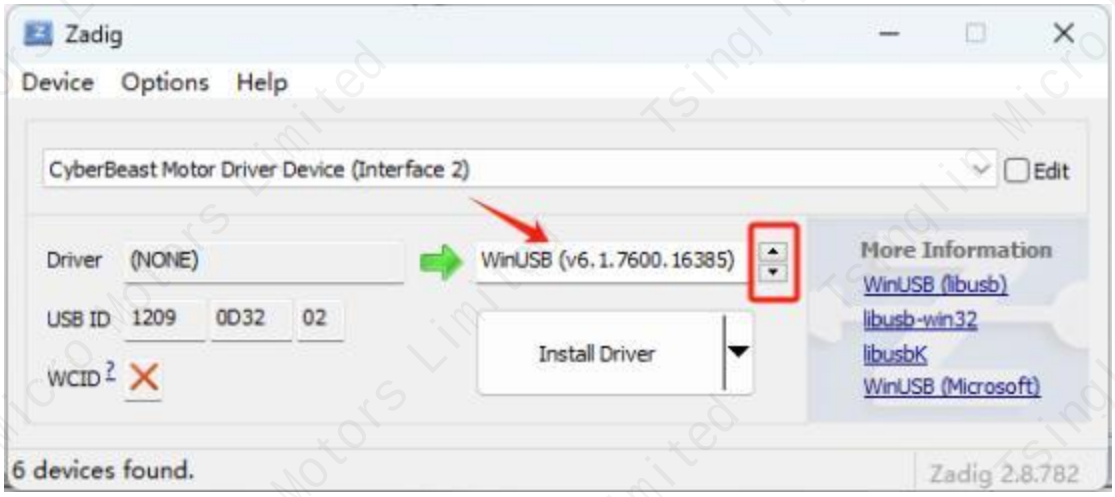
Run Windows PowerShell as administrator, enter pip install odrive, and press Enter. Retry if an error occurs; if repeated attempts fail, restart the computer and try again.

4. Install the USB Driver

Go to https://zadig.akeo.ie and download Zadig. Connect the driver to the computer with a Type-C cable; the driver power indicator will illuminate. Open Zadig and select "CyberBeast Motor Driver Device (Interface 2)" from the drop-down list:



Use the up/down controls to select the USB driver. Choose "WinUSB" for this interface and click "Install Driver":



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

1) Install Python/USB/odrivetool

If WSL2 is installed, open the WSL2 command line and perform the following steps (assuming Ubuntu is installed under WSL):

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

The first command installs Python, the second installs the USB library, and the third installs the odrivetool host software.

2) Connect the Driver to WSL

When the Type-C cable is connected to Windows, Windows loads the USB-port driver by default, but WSL does not. The USB port must be attached to WSL; refer to the Microsoft documentation:
(https://learn.microsoft.com/en-us/windows/wsl/connect-usb).

- **Ubuntu**

The installation procedure under Ubuntu is very similar to that under WSL; refer to the previous section.

3.1.3 Configuring Motor Parameters Correctly

Warning

Read this section carefully. Correct configuration is essential for successful operation and prevents motor damage.

After installing the host software (Motor Wizard or odrivetool) as described above, power the motor and connect the USB Type-C data cable:

u Motor Wizard

Open Motor Wizard. If the connection succeeds, the motor status and voltage will be displayed, as shown on the left above. If the connection fails, valid voltage information will not be shown.

Click "Motor Parameters". The driver hardware and firmware versions are shown at the top of the screen, as illustrated on the right above. In the example, the hardware version is 3.8.3 and the firmware version is 0.5.13.

◆ *odrivetool*

Run odrivetool in a shell (Windows PowerShell or Linux Terminal) by entering odrivetool and pressing Enter. The figure below shows a successful Windows connection, indicated by green text:

Command example: odrv0.axis0.controller.input_vel. odrv0 represents the currently connected motor. By default, the first connected motor is odrv0, the second is odrv1, and so on. axis0 represents the first motor connected to the driver; the current version supports only one motor. This command queries the current velocity-control target.

Operating Tips
u Use the TAB key frequently for command suggestions. Similar to Linux command completion, use the arrow keys to select a command.
u The Up/Down keys display command history.
u While entering a command, similar historical commands are suggested; press the Right Arrow key to complete the command.

➢ **Set Critical Limits**

u Motor Wizard

In Motor Wizard, click "Motor Parameters". The highlighted area below allows you to set the following critical limits: voltage range, current limit (maximum current), velocity limit (maximum speed), and calibration current.

The current limit is also affected by temperature. Refer to the related odrivetool description in the next subsection.

The velocity limit normally applies in velocity-control or position-control mode. It can also be enabled for torque-control mode. In Motor Wizard, click "Driver Parameters" and enable the switch highlighted below:

◆ *odrivetool*

l Current Limit

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

The command above sets the current limit to 30 A. Note that this is the Q-axis current, not the power-supply current. It directly limits output torque. For the 6010-8, do not set this limit above 50 A.

Other Factors Affecting the Current Limit
u Motor Temperature If motor-temperature protection is enabled (odrv0.axis0.motor.motor_thermistor.config.enabled=1), the current motor temperature also affects the Q-axis current.
u Driver-Board Temperature If driver-board temperature protection is enabled (odrv0.axis0.motor.fet_thermistor.config.enabled=1), the current driver-board temperature also affects the Q-axis current.

The two temperature effects are similar and can be expressed by the following formula:

$$I'_{lim} = \frac{T - T_l}{T_u - T_l} \times I_{lim}$$

where Ilim' is the effective current limit, Ilim is the configured current limit, T is the current temperature (motor or driver-board temperature), and Tl is the configured lower temperature limit (motor_thermistor.config.temp_limit_lower and fet_thermistor.config.temp_limit_lower); Tu is the configured upper temperature limit (motor_thermistor.temp_limit_upper and fet_thermistor.config.temp_limit_upper).

l Velocity Limit

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

This is the global system velocity limit. The command above sets it to 30 turn/s (revolutions per second). By default, this limit does not apply in pure torque mode, but it can be enabled with the following switch:

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

l Calibration Current

The default calibration current is 5 A and normally does not need to be changed. If the available supply current is low, reduce this value; otherwise, an undervoltage warning may occur during calibration.

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

➢ **Set Critical Hardware Parameters**

u Motor Wizard

In Motor Wizard, click "Motor Parameters". The highlighted area below shows several critical hardware parameters:

For the meaning of these parameters, refer to the odrivetool description in the next subsection.

l Maximum Discharge/Charge Current

Discharge current is the forward current supplied by the power source to the driver and motor; charge current is current flowing back into the power source. Set both values appropriately for the power source to avoid voltage sag caused by insufficient discharge capability or damage from back EMF. However, setting either value to a small absolute magnitude may easily trigger warnings.

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

l Pole Pairs

The number of pole pairs equals the number of rotor magnetic poles divided by two. This value must be set correctly for calibration to succeed; otherwise, a calibration warning will occur.

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

l Torque Constant

The torque constant is the motor torque divided by Q-axis current. Its relationship to the motor Kv is: Torque_constant = 8.27 / Kv.

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

An incorrect torque constant does not prevent the motor from operating, but it affects unit conversion for torque-control commands. To use amperes instead of N·m for torque control, set this value to 1.

l Temperature Sensing

Both the driver board and motor contain NTC temperature sensors. When enabled, the driver regulates output current (torque) according to temperature, thereby protecting the driver and motor.

```
# Motor Temperature Protection
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
# Driver-Board Temperature Protection
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
# Read Temperature
odrv0.axis0.motor.motor_thermistor.temperature # Motor
temperature odrv0.axis0.motor.fet_thermistor.temperature #
Driver-board temperature
```

- **PID Tuning**

The following procedure provides a reference for tuning PID parameters:

```
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
```

1. Set the initial PID values.
2. Set vel_integrator_gain to 0.

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

3. Tune vel_gain as follows:
    1) Run the motor in velocity-control mode. If rotation is unstable, jittery, or vibrating, reduce vel_gain until the motor runs smoothly.
    2) Then increase vel_gain by approximately 30% each time until obvious oscillation occurs.
    3) Reduce vel_gain by approximately 50% from that value to obtain stable operation.
4. Tune pos_gain as follows:
    1) Run the motor in position-control mode. If motion is unstable, jerky, or vibrating, reduce pos_gain until the motor moves smoothly.
    2) Then increase pos_gain by approximately 30% each time until obvious position overshoot occurs (the motor passes the target and then oscillates back to it).
    3) Gradually reduce pos_gain until the overshoot disappears.
5. After the four steps above, set vel_integrator_gain to 0.5 × bandwidth × vel_gain, where bandwidth is the system control bandwidth. For example, if the motor takes 10 ms to reach a commanded position, the control bandwidth is 100 Hz. Therefore: vel_integrator_gain=0.5*100*vel_gain。

During tuning, use the graphical method in Section 3.1.8 to observe the results in real time and avoid errors from visual judgment. With Motor Wizard, PID tuning is shown directly. The figure below illustrates the PID parameters:

3.1.4 Power-On Calibration

When the micromotor is used for the first time, both the motor and encoder must be calibrated. Before calibration, secure the motor firmly and ensure that the output shaft is unloaded. The procedure is as follows:

Warning

Before calibration, remove all load from the motor and secure it firmly.

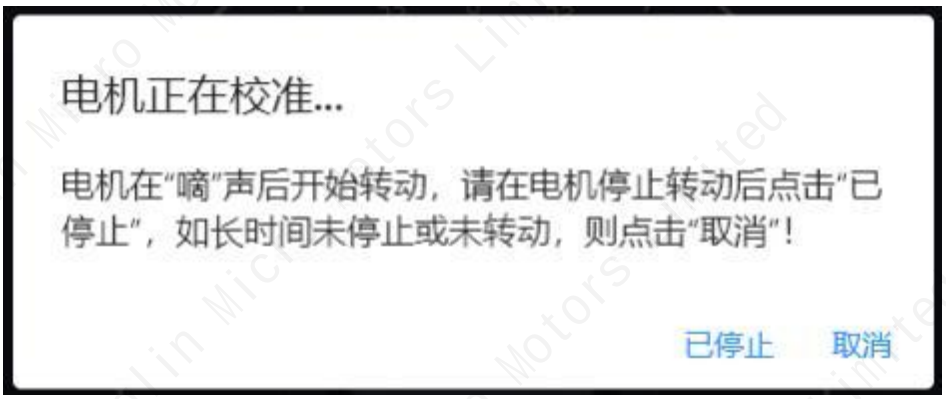
u Motor Wizard

Open Motor Wizard and click "Start Motor". If the motor has not been calibrated, the following prompt appears:

Click "OK" to enter the calibration process. Alternatively, click "Motor Parameters" on the main screen and open "Basic Parameters".

The motor calibration status is displayed there. If it shows "Not Calibrated", click the "Calibrate" button beside the status to start the same calibration process. 

After calibration starts, the following prompt appears:

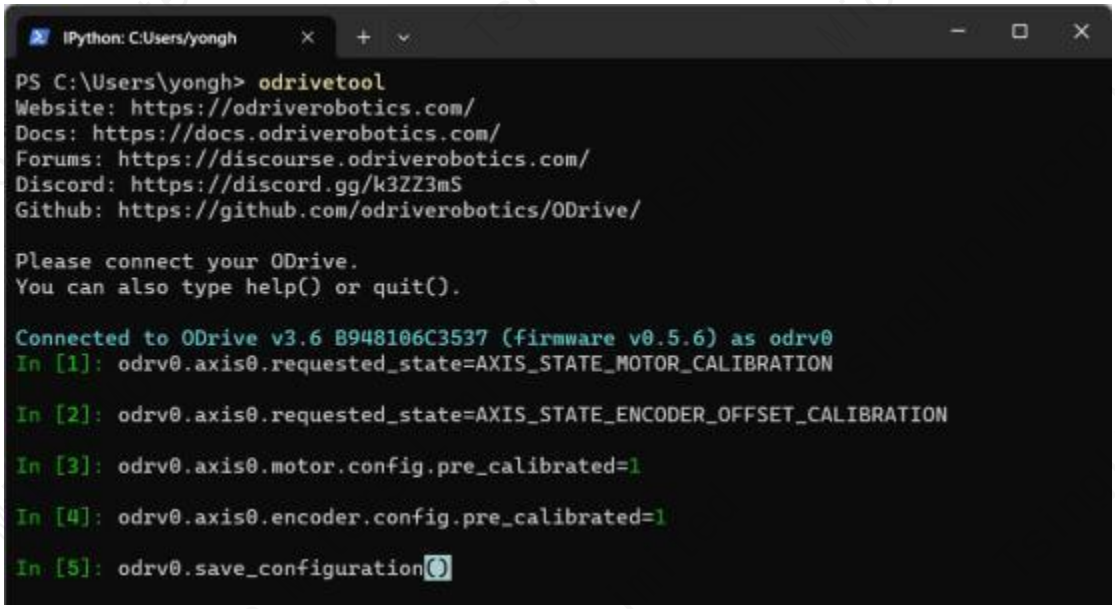


The motor will emit a beep and begin rotating. Observe the motor. If it stops after rotating for a period, click "Stopped" to complete calibration. If it does not rotate or has not stopped after five minutes, click "Cancel".

After calibration, the driver restarts. Wait 1-3 seconds, then reopen "Motor Parameters" and check whether the status

shows "Calibrated".已校准

◆ *odrivetool*




```
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()
```

The steps are explained below:

- Step 1: Motor Parameter Identification

The driver measures the motor phase resistance and phase inductance, during which a sharp beep is heard. The measurement results can be viewed with the following commands:

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

- Step 2: Check Error Codes

Check the system error codes after Step 1. If any red error code appears, restart the motor and retry, or contact after-sales support.

- Step 3: Encoder Calibration

Calibrate the encoder, including the relationship between the encoder installation angle and the motor mechanical angle, as well as the encoder itself. During calibration, the motor slowly rotates forward through an angle and then backward. If it stops after rotating only forward, an error has occurred; check the error codes in Step 4.

- Step 4: Check Error Codes

After encoder calibration in Step 3, check the system error codes. A common error is ERROR_CPR_POLEPAIRS_MISMATCH, indicating that the encoder CPR or the motor pole-pair setting is incorrect. View or set these values with the following commands:

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

- Step 5: Set the Motor Calibration-Success Flag
- Step 6: Set the Encoder Calibration-Success Flag
- Step 7: Save the Calibration Results and Restart

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

3.1.5 Changing the ID

The driver ID must be unique on the bus. Its range is 0-63 and the default is 0. When multiple motors are connected to the same bus, assign a different ID to each motor. The ID can be changed through USB or the bus:

- Change the ID in Motor Wizard

Open Motor Wizard and connect to the motor. Click "Motor Parameters", change the ID under "Communication Parameters", and click "Sync" to store it in the driver.

- Change the ID through USB

After connecting to the driver with odrivetool, change the ID with the following command:

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

- Change the ID through the Bus

Refer to the Set_Axis_Node_ID command message in Section 4.1.2.

3.1.6 Saving and Backing Up Parameters

After changing any parameter, always save the configuration; otherwise, the changes will be lost after power-off or restart. The driver restarts after the parameters are saved.

```
odrv0.save_configuration()
```

Parameter backup:

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

In the example, "d:\test.json" is the user-defined save path

and file name. The command for restoring parameters is:

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

3.1.7 Four Control Modes

After completing the preparation and parameter configuration above, you can operate the motor in different modes. The 6010-8 supports position control, velocity control, torque control, and motion control.

Position-control modes include Filtered Position Control, Trajectory Control, and Circular Position Control.

Velocity-control modes include direct Velocity Control and Ramped Velocity Control.

Torque-control modes include direct Torque Control and Ramped Torque Control.

Motion control combines position, velocity, and torque control and is generally used where high instantaneous force is required, such as a robot knee joint. It is also commonly called MIT control, a name derived from the open-source MIT quadruped robot that uses this control method.

The detailed examples below use host software and USB commands, but the same control logic also applies when using communication protocols such as CAN.

How do I make the motor rotate?
u Start the Motor (Enter Closed-Loop Control) For all subsequent control operations, the motor must be in closed-loop control before it can rotate. In Motor Wizard, click "Start Motor". In odrivetool, use the following command: odrv0.axis0.requested_state = 8
u Stop the Motor (Enter Idle State) Before stopping the motor, changing parameters, or saving parameters, place the motor in the idle state. In Motor Wizard, click "Stop Motor". In odrivetool, use the following command: odrv0.axis0.requested_state = 1

u Motor Wizard

Open Motor Wizard. The four tabs at the bottom correspond to the four control modes: "Velocity Control", "Torque Control", "Position Control", and "MIT Mode".

- **Basic Interface Operations**

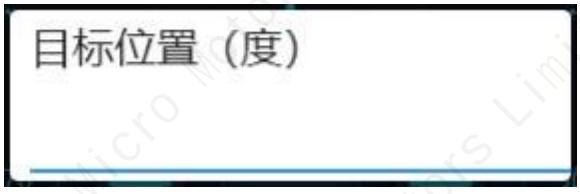
After the motor is started, the dial is highlighted in color. Almost all control operations are performed on the dial:

- ✓ Left-Click the Dial

Left-clicking the dial commands the speed, torque, or position indicated by the current dial value. Each click sends one control command. Repeated clicks can be used for consecutive control operations.

- ✓ Right-Click the Dial

Right-click the dial to open a dialog box, enter an exact target speed, torque, or position, and press Enter to execute the command. Click outside the dialog box to cancel.



✓ Click and Drag the Dial

Clicking and dragging the dial provides continuous control. Speed, torque, or position commands are sent continuously, with the target determined by the current pointer position on the dial.

➢ **Modify Driver Parameters**

Click "Driver Parameters" to adjust the control mode, velocity limit, three-loop gains (bandwidths), and other driver parameters. Stop the motor before modifying and synchronizing parameters.

◆ *odrivetool*

➢ **Filtered Position Control**

Filtered Position Control is recommended when the user generates a position profile and sends position commands at a fixed frequency, because it smoothly connects the commands. Using Trajectory Control in this situation may cause jerky or granular motor motion.

In this mode, set the filter bandwidth according to the command frequency. A useful rule is to set the bandwidth to half the command frequency in hertz. For example, when sending commands at 50 Hz:

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

Enable Filtered Position Control:

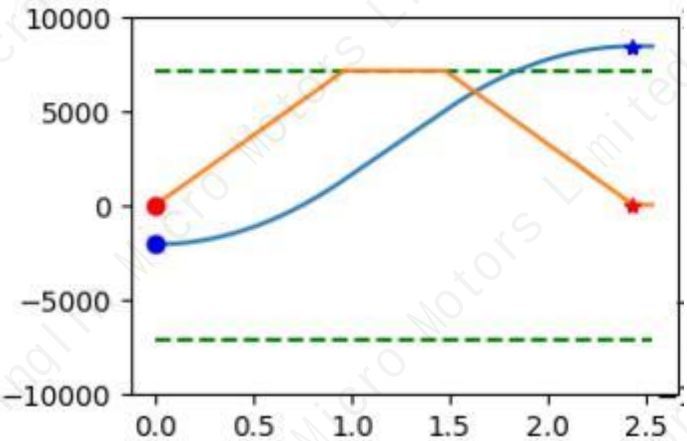
```
odrv0.axis0.controller.config.control_mode = 3
odrv0.axis0.controller.config.input_mode = 3
```

Then command the position:

```
odrv0.axis0.controller.input_pos = 10 # Unit: turns
```

➢ **Trajectory Control**

This mode allows the user to set acceleration, cruise velocity, and deceleration so that the motor moves smoothly from one position to another. The term "trapezoidal" refers to the shape of the velocity curve. In the figure below, orange indicates velocity and blue indicates position:



Adjustable control parameters:

```
odrv0.axis0.trap_traj.config.vel_limit # Maximum cruise velocity, unit: turn/s
odrv0.axis0.trap_traj.config.accel_limit # Maximum acceleration, unit: turn/s^2
odrv0.axis0.trap_traj.config.decel_limit # Maximum deceleration, unit:
turn/s^2 odrv0.axis0.controller.config.inertia # Inertia, unit:
N·m/(turn/s^2)
```

Note that inertia × acceleration = torque. The default inertia value is 0. This value can improve system response, but it is directly related to the motor load. All four values above must be greater than or equal to 0. The current and velocity limits described earlier still apply globally. For example, if the trajectory cruise-velocity limit is set above the system-level vel_limit, the global vel_limit takes effect.

To enable Trajectory Control, first set:

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

Then command the position:

```
odrv0.axis0.controller.input_pos = 10 # Unit: turns
```

- **Circular Position Control**

This mode is suitable for continuous incremental position control, such as a robot wheel rotating in one direction for an extended period or a conveyor running continuously. In normal position-control mode, the target position may grow to a very large value and eventually cause positioning errors because of floating-point precision limitations.

Enable the mode:

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

In this mode, each incremental step remains within one revolution and input_pos is in the range [0, 1). If input_pos exceeds this range, it is converted to an equivalent single-turn value. To allow a single step greater than one turn, set the following parameter to a value greater than 1:

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

- **Direct Velocity Control**

This is the simplest velocity-control mode. Enable it as follows:

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

Then enter the target velocity:

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

- **Ramped Velocity Control**

Ramped Velocity Control increases velocity gradually to the target at a specified slope, providing smoother motion than direct velocity control. Enable it as follows:

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

Adjust the ramp rate to control acceleration:

```
odrv0.axis0.controller.config.vel_ramp_rate = 0.5 # Ramp-rate unit: turn/s^2
```

Then enter the target velocity:

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

- **Direct Torque Control**

This is the simplest torque (current) control mode. Enable it as follows:

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

Torque commands use N·m, while current in the driver firmware is measured in amperes. Therefore, the torque constant must be set so the driver can convert N·m to current and produce the commanded torque.

```
# Torque constant is approximately 8.23 / Kv
odrv0.axis0.motor.config.torque_constant = 8.23/12.3
```

Then enter the target velocity:

```
odrv0.axis0.controller.input_torque = 1.2 # Unit: N·m
```

To limit the maximum speed in torque mode, enable enable_torque_mode_vel_limit and set vel_limit, for example:

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

- **Ramped Torque Control**

Ramped Torque Control is similar to Ramped Velocity Control. Enable it as follows:

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

Set the ramp rate as follows:

```
odrv0.axis0.controller.config.torque_ramp_rate = 0.1 # Ramp-rate unit: N·m/s
```

➢ **Motion Control (MIT Control)**

Motion Control drives the motor to a target position by combining position, velocity, and torque control. It can be expressed by the following formula:

$$T_{target} = K_p \times p_{diff} + K_d \times v_{diff} + T_{ff}$$

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

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

where Ttarget is the target torque, pdiff is the position error, vdiff is the velocity error, Kp is the position gain, Kd is the velocity gain (damping coefficient), and Tff is the feedforward torque.

Enable Motion Control as follows:

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

Set the gains:

```
odrv0.axis0.controller.input_mit_kp = <float> # position gain, unit: N·m/turn
odrv0.axis0.controller.input_mit_kd = <float> # damping coefficient, unit: N·m/turn/s
odrv0.axis0.controller.input_pos = 5 # Unit: turns
```

```
odrv0.axis0.controller.input_vel = 30 # Unit: turn/s
odrv0.axis0.controller.input_torque = 2 # Unit: N·m
```

Then perform motion control by setting input_pos, input_vel, and input_torque:

Note: for USB control, the entered position, velocity, and torque refer to the rotor side. For MIT control over CAN, the protocol position, velocity, and torque refer to the output-shaft side to remain compatible with the open-source MIT protocol.

3.1.8 Advanced Functions

1) Common Command List

After a successful connection, commands can be used to control the motor and obtain operating parameters. The table below lists common commands, commissioning procedures, and descriptions:

Category	Command	Description
Basic Command	dump_errors(odrv0)	Print all error information.
	odrv0.clear_errors()	Clear all error information.
	odrv0.save_configuration()	After changing parameters, identifying motor parameters, or calibrating, always execute this command to save the changes; otherwise, all changes will be lost after power-off.
	odrv0.reboot()	Restart the driver.

s	odrv0.vbus_voltage	Read the supply voltage (V).

	odrv0.ibus	Read the supply current (A).
	odrv0.hw_version_major	Hardware major version. The current major version of the 6010-8 is 3.
	odrv0.hw_version_minor	Hardware minor version. The current minor version of the 6010-8 is 8.
	odrv0.hw_version_variant	Model variant under the same hardware configuration. The model variant for the 6010-8 is 1.
	odrv0.can.config.r120_gpio_num	GPIO number controlling the 120 Ω CAN termination-resistor switch.
	odrv0.can.config.enable_r120	Enable/disable the 120 Ω CAN termination resistor.
	odrv0.can.config.baud_rate	Set the CAN baud rate.
Parameter Configuration Commands	odrv0.config.dc_bus_undervoltage_trip_level	Undervoltage warning threshold (V).
	odrv0.config.dc_bus_overvoltage_trip_level	Overvoltage warning threshold (V).
	odrv0.config.dc_max_positive_current	Maximum forward line current, positive value (A).
	odrv0.config.dc_max_negative_current	Maximum reverse charging line current, negative value (A).
	odrv0.axis0.motor.config.resistance_calib_max_voltage	Maximum voltage used during motor parameter identification. This is generally set slightly below half the supply voltage; for a 24 V supply, it may be set to 10.
	odrv0.axis0.motor.config.calibration_current	Maximum current used during motor parameter identification. This is generally set to 2-5 A and must not be too high or too low.
	odrv0.axis0.motor.config.torque_constant	Motor torque constant (N·m/A).
	odrv0.axis0.min_endstop.config odrv0.axis0.max_endstop.config	Configuration of the minimum (LW1) and maximum (LW2) limit switches: enabled: whether the function is enabled gpio_num: corresponding I/O number. Set the minimum-limit I/O to 1 and the maximum-limit I/O to 2.
	odrv0.axis0.encoder.config.index_offset	User-defined zero-point offset. This value is the offset of the user zero relative to the encoder zero. After setting and saving it, all user position-control targets are referenced to this user zero.
	odrv0.axis0.motor.motor_thermistor.config	Configure the motor temperature sensor: enabled: whether the function is enabled temp_limit_lower: lower temperature limit temp_limit_upper: upper temperature limit
	odrv0.axis0.motor.fet_thermistor.config	
	odrv0.axis0.motor.motor_thermistor.temperature	Motor temperature

	odrv0.axis0.motor.fet_thermistor.temp er ature	Driver temperature
Calibration Command	odrv0.axis0.requested_state=4	Identify motor parameters, including phase resistance, phase inductance, and three-phase current-balance cali-

		bration. The process takes 3-6 seconds and the motor emits a sharp tone. When the tone stops, or if no tone is heard after six seconds, run dump_errors(odrv0) and confirm that there are no errors before continuing.
	odrv0.axis0.requested_state=7	Calibrate the encoder. Before executing this operation, ensure that the motor output shaft is unloaded and secure the motor by hand or with a fixture. The motor will rotate forward and backward for a period to identify and calibrate the encoder. After the motor stops, run dump_errors(odrv0), confirm that there are no errors, and then continue with the subsequent steps.
	odrv0.axis0.encoder.config.pre_calibrate d=1	Set the pre-calibrated flag so calibration is not required at every power-on. This parameter can be written only after successful calibration; otherwise, the write will fail.
	odrv0.axis0.controller.config.load_encod er_axis=0	Ensure that the currently operated motor is motor 0. This operation is required only in BETA versions and has no effect in production versions.
Control Comm	odrv0.axis0.requested_state=1	Stop the motor and enter the idle state.
	odrv0.axis0.requested_state=8	Start the motor and enter the closed-loop state.
	odrv0.axis0.motor.config.current_lim	Maximum motor line current (A). Exceeding this value triggers an overcurrent warning. This value must not exceed 100. **100。**
	odrv0.axis0.controller.config.vel_limit	Maximum motor speed (turn/s). An overspeed warning is triggered if rotor speed exceeds this value.
	odrv0.axis0.controller.config.enable_vel_ limit	Velocity-limit switch. When True, the vel_limit above is active; when False, it is inactive.
	odrv0.axis0.controller.config.control_mo de	Control mode: 0: Voltage control 1: Torque control 2: Velocity control 3: Position control

and s	odrv0.axis0.controller.config.input_mo d e	Input mode, defining how the user-entered control value drives the motor: 0: Inactive 1: Direct control 2: Velocity ramp 3: Position filter 5: Trapezoidal trajectory 6: Torque ramp 9: Motion control (MIT)
	odrv0.axis0.controller.config.vel_gain	Velocity-loop PID proportional gain.
	odrv0.axis0.controller.config.vel_integr at or_gain	Velocity-loop PID integral gain.
	odrv0.axis0.controller.input_mit_kp	Motion-control (MIT) position gain.

	odrv0.axis0.controller.input_mit_kd	Motion-control (MIT) velocity gain (damping coefficient).
	odrv0.axis0.controller.config.pos_gain	Position-loop PID proportional gain.
	odrv0.axis0.controller.input_torque	Torque-control target, or torque feedforward for velocity/position control (N·m).
	odrv0.axis0.controller.input_vel	Velocity-control target, or velocity feedforward for position control (turn/s).
	odrv0.axis0.controller.input_pos	Position-control target (turns).
	odrv0.axis0.encoder.set_linear_count()	Set the encoder absolute position. Enter a signed 32-bit integer in parentheses; its absolute value must be less than the specified limit. odrv0.axis0.encoder.config.cpr
	odrv0.axis0.trap_traj.config	Contains three parameters: ➢ accel_limit: maximum acceleration (rev/s^2) ➢ decel_limit: maximum deceleration (rev/s^2) ➢ vel_limit: maximum velocity (rev/s). These three parameters take effect when odrv0.axis0.controller.config.input_mode is set to trapezoidal trajectory and determine the acceleration/deceleration behavior of position control.
	odrv0.axis0.controller.config. input_filter_bandwidth	Position-filter bandwidth. This parameter takes effect when odrv0.axis0.controller.config.input_mode is set to position filter and determines the acceleration/deceleration behavior of position control.

2) Graphical Commissioning

When commissioning the motor, Python's calculation and graphics libraries can be combined with the high throughput of the Type-C interface to display selected operating parameters in real time.

1. Environment Setup

Install the calculation and graphics libraries:

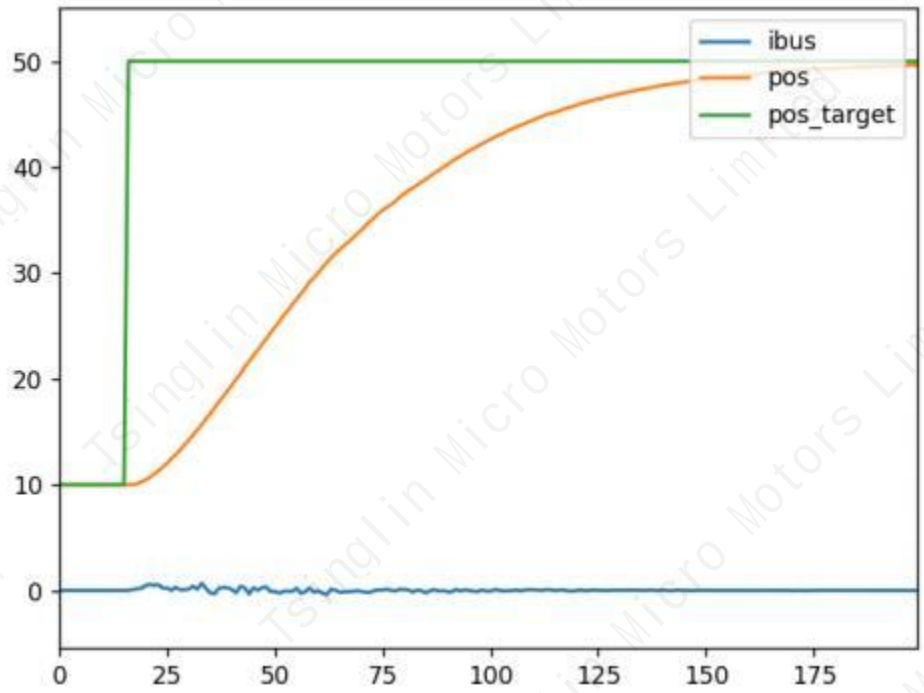
```
pip install numpy matplotlib
```

2. Graphical Parameter Output

In the odrivetool command-line interface, start the graphics library and read any motor operating parameters, for example:

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

This command opens a graphical interface and displays three values in real time: line current, position, and target position. When position control is then performed, the real-time position-control curve is displayed:



3) *USB and CAN Compatibility*

In early product versions (hardware version 3.7 or earlier, which can be read using the commands odrv0.hw_version_major and odrv0.hw_version_minor described in this section), USB and CAN cannot be used simultaneously. Switch between them as follows. Hardware versions later than 3.7 may ignore this section:

- Switching from USB Communication to CAN

When CAN is disabled, communication is available through the Type-C interface. Use the following command to switch to CAN:

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

- Switching from CAN Communication to USB

When CAN is enabled, first send Set_Axis_State with parameter 1 to place the motor in the IDLE state, and then send Disable_Can to switch to USB. See Section 4.1.2.

Whether switching from USB to CAN or from CAN to USB, first place the motor in the IDLE state; otherwise, the switch will fail.

4) *CAN Termination-Resistor Switch*

A 120 Ω termination resistor is built into the driver and can be enabled or disabled as required. Example commands are shown below:

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

5) *User Zero-Point Configuration*

By default, both the position read from the motor and the input value used for position control are referenced to the zero point of the absolute encoder on the driver. In most applications, however, the encoder zero is not the user's mechanical zero, so the user must manually set a zero-point offset.

The zero position can generally be located in two ways: by using a limit switch, or by manually setting the zero offset, which is the user zero position relative to the encoder zero:

```
# After manually moving the motor, or using position
control, to the desired user zero position:
odrv0.axis0.encoder.config.index_offset =
```

6) *Limit Switches*

The driver supports two limit switches, LW1 and LW2. LW1 is the minimum and homing position, while LW2 is the maximum position. To use both limit switches, apply the following configuration:

```
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
```

When a limit switch is triggered, the system reports MIN_ENDSTOP_PRESSED or MAX_ENDSTOP_PRESSED. The host software may then perform the required operation.

Note: hardware version 3.7 does not support the limit-switch function.

7) *Braking Resistor*

A braking resistor (also called a discharge resistor) may be connected between EMG and GND at the braking-resistor interface described in Section 2.4.7. It dissipates transient current when bus current flows back into the power source.

Enable it with the following configuration:

```
odrv0.config.enable_brake_resistor = True
odrv0.config.brake_resistance = xxx # Set the external braking
resistance odrv0.save_configuration()
```

How does the driver automatically regulate energy dissipation? The following two methods can be combined to adjust the discharge current automatically:

- By Setting the Maximum Regenerative Current

```
odrv0.config.max_regen_current = xxx # Maximum reverse current, positive value
```

When the reverse current exceeds the configured value, the driver diverts the excess current to the braking resistor.

- By Setting the Back-EMF Voltage Range

```
odrv0.config.enable_dc_bus_overvoltage_ramp = True
odrv0.config.dc_bus_overvoltage_ramp_start = xxx # Lower voltage threshold for diversion
odrv0.config.dc_bus_overvoltage_ramp_end = xxx # Upper voltage threshold for diversion
```

When back-EMF voltage exceeds the lower threshold, the driver begins diverting energy to the braking resistor. At or above the upper threshold, the full bus voltage is applied to the braking resistor to maximize dissipation. Between the two thresholds, the applied voltage is increased proportionally.

The two modes above may be enabled simultaneously; the driver combines both methods when regulating energy dissipation.

3.2 Firmware Update and Download

Firmware can be programmed through the SWD interface (Section 2.4.4) or Type-C interface (Section 2.4.2). The following three methods are provided:

3.2.1 Nations Download Tool

1. USB (DFU) Programming

The Nations download tool can program firmware through the Type-C interface or through SWD (J-Link and DAP only). This section mainly describes programming through Type-C.

First, download the USB driver for the Nations programming tool (https://www.cyberbeast.cn/filedownload/789489) and install the driver for your operating system. Then download the Nations programming tool (https://cyberbeast.cn/filedownload/766844), extract it to any directory, and run it.

Connect the Type-C interface, open odrivetool, and execute the following command to place the driver in DFU mode:

```
odrv0.enter_dfu_mode()
```

Finally, program the firmware with the Nations tool as shown below. After programming, click "Common Operations" and then "Reset" to restart the driver. The driver can then reconnect to odrivetool for normal commissioning.

2. SWD (J-Link or DAP) Programming

SWD programming is similar to DFU programming, but the connection is made through the SWD debugging interface (Section 2.4.4), and the corresponding debug tool (J-Link or DAP) must be selected in the screen above.

3.2.2 pyocd

pyOCD is a Python-based tool similar to OpenOCD and supports common debug probes such as ST-Link, J-Link, and DAP for erase, programming, reset, and other operations. The driver must be connected through SWD. See Section 2.4.4 for the SWD pinout. The SWD interface includes a 3.3 V supply; incorrect wiring may damage the driver.

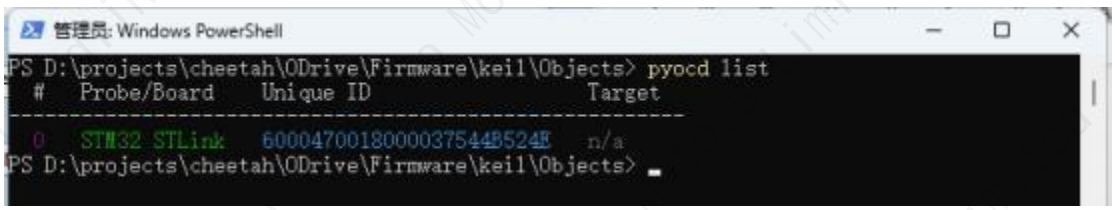
1. Installation

```
pip install pyocd
```

2. Programming

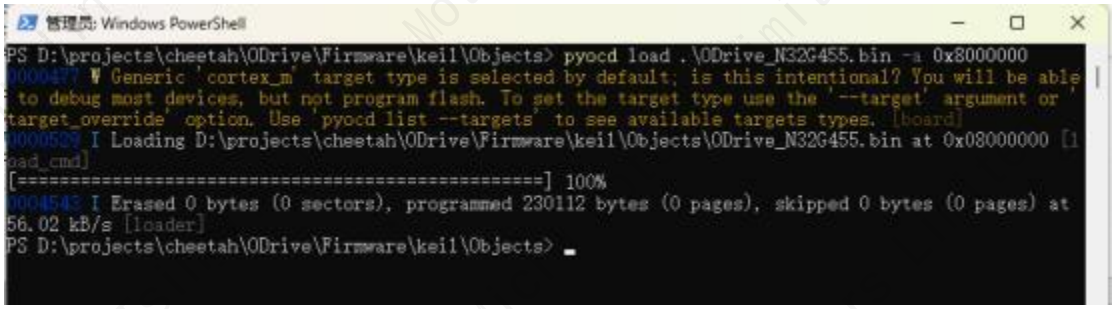
First, list the connected debug probes:

```
pyocd list
```



Then program the BIN file with the following command:

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



4 Communication Protocols and Examples

4.1 CAN Simple Protocol

The default communication interface is CAN, with a maximum data rate of 1 Mbps (read or set through odrv0.can.config.baud_rate). The factory default is 500 kbps. Note: on early hardware versions (3.7 or earlier), USB and CAN are not compatible.

Refer to Item 3) in Section 3.1.8 for switching from USB to CAN.

The driver's default protocol is CAN Simple. If the protocol has been changed to CANopen, use the following command to switch back to CAN Simple:

```
odrv0.can.config.protocol = 1
```

On hardware version 3.10 and later, the communication port also supports RS485, pulse/direction, and PWM. Use the following command to ensure that the physical interface is set to CAN:

```
odrv0.config.comm_intf_mux = 0
```

In Motor Wizard, the physical interface and CAN protocol version can also be selected under Communication Parameters.

4.1.1 Protocol Frame Format

CAN communication uses standard data frames with an 11-bit ID and 8 data bytes, as shown below (MSB on the left and LSB on the right):

Data Field	CAN ID（11bits）		Data（8 bytes）
Segment	Bit10 ~ Bit5	Bit4 ~ Bit0	Byte0 ~ Byte7
Description	node_id	cmd_id	Communication Data

l node_id: the unique motor ID on the bus. It can be read and set in odrivetool through odrv0.axis0.config.can.node_id.

l cmd_id: command code identifying the message type. Refer to the remainder of this section.

l Communication data: eight bytes. Parameters in each message are encoded as integers or floating-point values in little-endian byte order. Floating-point values follow IEEE 754 (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

Using the Set_Input_Pos message described in Section 4.1.2 as an example, assume its three parameters are Input_Pos = 3.14,
Vel_FF = 1000 (representing 1 rev/s), and Torque_FF = 5000 (representing 5 N·m). The Set_Input_Pos message CMD ID is 0x00C. If the driver node_id is 0x05:

l 11-bit CAN ID = (0x05 << 5) + 0x0C = 0xAC

l According to the Set_Input_Pos definition in Section 4.1.2, Input_Pos occupies four bytes starting at byte 0 and is encoded as C3 F5 48 40 (the floating-point value 3.14 is encoded as a 32-bit IEEE 754 value 0x4048F5C3). Vel_FF occupies two bytes starting at byte 4 and is encoded as E8 03 (1000 = 0x03E8). Torque_FF occupies two bytes starting at byte 6 and is encoded as 88 13 (5000 = 0x1388). Therefore, the eight communication data bytes are:

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

4.1.2 Frame Messages

The table below lists all available messages:

CMD ID	Name	Direction	Parameters
0x001	Heartbeat	Motor → Host→	Axis_Error Axis_State Motor_Flag Encoder_Flag Controller_Fla g Traj_Done Life
0x002	Estop	Host → Motor→	
0x003	Get_Error	Motor → Host→	Error_Type
0x004	RxSdo	Motor → Host→	
0x005	TxSdo	Motor → Host→	
0x006	Set_Axis_Node_ID	Host → Motor→	Axis_Node_ID
0x007	Set_Axis_State	Host → Motor→	Axis_Requested_State
0x008	Mit_Control	Host → Motor→	
0x009	Get_Encoder_Estimates	Motor → Host→	Pos_Estimate Vel_Estimate
0x00A	Get_Encoder_Count	Motor → Host→	Shadow_Count Count_In_Cpr
0x00B	Set_Controller_Mode	Host → Motor→	Control_Mod e Input_Mode

0x00C	Set_Input_Pos	Host → Motor→	Input_Pos

			Vel_FF Torque_FF
0x00D	Set_Input_Vel	Host → Motor→	Input_Vel Torque_FF
0x00E	Set_Input_Torque	Host → Motor→	Input_Torque
0x00F	Set_Limits	Host → Motor→	Velocity_Limit Current_Limit
0x010	Start_Anticogging	Host → Motor→	
0x011	Set_Traj_Vel_Limit	Host → Motor→	Traj_Vel_Limit
0x012	Set_Traj_Accel_Limits	Host → Motor→	Traj_Accel_Limit Traj_Decel_Limit
0x013	Set_Traj_Inertia	Host → Motor→	Traj_Inertia
0x014	Get_Iq	Motor → Host→	Iq_Setpoint Iq_Measured
0x016	Reboot	Host → Motor→	
0x017	Get_Bus_Voltage_Current	Motor → Host→	Bus_Voltage Bus_Current
0x018	Clear_Errors	Host → Motor→	
0x019	Set_Linear_Count	Host → Motor→	Linear_Count
0x01A	Set_Pos_Gain	Host → Motor→	Pos_Gain
0x01B	Set_Vel_Gains	Host → Motor→	Vel_Gain Vel_Integrator_Gain
0x01C	Get_Torques	Motor → Host→	Torque_Setpoin t Torque
0x01D	Get_Powers	Motor → Host→	Electrical_Power Mechanical_Power
0x01E	Disable_Can	Host → Motor→	
0x01F	Save_Configuration	Host → Motor→	

Detailed descriptions of all messages follow:

- Heartbeat

CMD ID: 0x001 (Motor → Host)→

For firmware version 0.5.11 and earlier, the heartbeat format is:

Start Byte	Name	Category	Description
0	Axis_Error	uint32	Axis error code (odrv0.axis0.error)
4	Axis_State	uint8	Driver state (odrv0.axis0.current_state)
5	Motor_Flag	uint8	1: Motor fault (odrv0.axis0.motor.error is not 0) 0: Motor normal (odrv0.axis0.motor.error is 0)
6	Encoder_Flag	uint8	1: Encoder fault (odrv0.axis0.encoder.error is not 0)

			0: Encoder normal (odrv0.axis0.encoder.error is 0)
7	Controller_Flag	uint8	bit7: odrv0.axis0.controller.trajectory_done, indicating whether the position trajectory is complete bit0： 1: Controller fault (odrv0.axis0.controller.error is not 0) 0: Controller normal (odrv0.axis0.controller.error is 0)

For firmware version 0.5.12 and later, the heartbeat format is:

Start Byte	Name	Category	Description
0	Axis_Error	uint32	Axis error code (odrv0.axis0.error)
4	Axis_State	uint8	Driver state (odrv0.axis0.current_state)
5	Flags	uint8	bit0: Motor-fault flag (whether odrv0.axis0.motor.error is 0) bit1: Encoder-fault flag (whether odrv0.axis0.encoder.error is 0) bit2: Controller-fault flag (whether odrv0.axis0.controller.error is 0) bit7: odrv0.axis0.controller.trajectory_done, indicating whether the position trajectory is complete
6	Reserved	uint8	Reserved
7	Life	uint8	Lifetime counter for periodic messages. It increments by 1 for each heartbeat in the range 0-255. A discontinuity indicates a lost heartbeat and unstable communication.

For firmware version 0.5.13 and later, the heartbeat format is:

Start Byte	Name	Category	Description
0	Axis_Error	uint32	Axis error code (odrv0.axis0.error)
4	Axis_State	uint8	Driver state (odrv0.axis0.current_state)
5	Flags	uint8	bit0: Motor-fault flag (whether odrv0.axis0.motor.error is 0) bit1: Encoder-fault flag (whether odrv0.axis0.encoder.error is 0) bit2: Controller-fault flag (whether odrv0.axis0.controller.error is 0) bit3: System-fault flag (whether odrv0.error is 0) bit7: odrv0.axis0.controller.trajectory_done, indicating whether the position trajectory is complete
6	Reserved	uint8	Reserved

7	Life	uint8	Lifetime counter for periodic messages. It increments by 1 for each heartbeat in the range 0-255. A discontinuity indicates a lost heartbeat and unstable communication.

- Estop

CMD ID: 0x002 (Host → Motor), no parameters or data.→

This command causes an emergency stop and reports the ESTOP_REQUESTED error.

- Get_Error

CMD ID: 0x003 (Motor → Host)→

Input (Host → Motor):→

Start Byte	Name	Category	Description
0	Error_Type	uint8	0: Get motor errors 1: Get encoder errors 3: Get controller errors 4: Get system errors

- Output (Motor → Host):→

Start Byte	Name	Category	Description
0	Error	uint32/uint64	Returned data and length for each Error_Type input: 0: Motor error (odrv0.axis0.motor.error), uint64, 8 bytes 1: Encoder error (odrv0.axis0.encoder.error), uint32, 4 bytes 3: Controller error (odrv0.axis0.controller.error), uint32, 4 bytes 4: System error (odrv0.error), uint32, 4 bytes

- RxSdo

CMD ID: 0x004 (Host → Motor)

Input:→

Start Byte	Name	Category	Description
0	opcode	uint8	0: Read 1: Write
1	Endpoint_ID	uint16	Download the JSON file containing the IDs for all parameters and interface functions: ➢ Version 0.5.13

			https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.js on ➢ Version 0.5.14 https://bl.cyberbeast.cn/actuator/endpoints_0.5.14.js on
3	Reserved	uint8	
4	Value	uint8[4]	Varies with Endpoint_ID; refer to the JSON description above. For example, if Endpoint_ID corresponds to a readable/writable float value, these four bytes contain the IEEE-encoded float. When opcode = 1, this value is written to the float endpoint.

Output (when opcode = 0 above):

Start Byte	Name	Category	Description
0	opcode	uint8	Fixed at 0
1	Endpoint_ID	uint16	Download the JSON file containing the IDs for all parameters and interface functions: ➢ Version 0.5.13 https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.js on ➢ Version 0.5.14 https://bl.cyberbeast.cn/actuator/endpoints_0.5.14.js on
3	Reserved	uint8	
4	Value	uint8[4]	Varies with Endpoint_ID; refer to the JSON description above. For example, if Endpoint_ID corresponds to a readable uint32 value, these four bytes contain the uint32 in little-endian order.

➢ TxSdo

CMD ID: 0x005 (Motor → Host)➔

Usage is the same as RxSdo when opcode = 1.

➢ Set_Axis_Node_ID

CMD ID: 0x006 (Host → Motor)➔

Start Byte	Name	Category	Description
0	Axis_Node_ID	uint32	odrv0.axis0.config.can.node_id: unique motor ID on the CAN bus

➢ Set_Axis_State

CMD ID: 0x007 (Host → Motor)➔

Start Byte	Name	Category	Description
0	Axis_Requested_State	uint32	odrv0.axis0.requested_state ➢ 0: Undefined ➢ 1: Idle ➢ 3: Full calibration (motor + encoder calibration) ➢ 4: Motor calibration ➢ 7: Encoder calibration ➢ 8: Closed loop

➢ Mit_Control

CMD ID: 0x008

This is an implementation that emulates the open-source MIT motion-control protocol (https://github.com/mit-biomimetics/Cheetah-Software).

Note: for USB control, entered position, velocity, and torque refer to the rotor side. For MIT control over CAN, protocol position, velocity, and torque refer to the output-shaft side to remain compatible with the open-source MIT protocol.

✓ Host → Motor➔

CAN Data-Frame Bits	Meaning	Description
BYTE0 BYTE1	**Position: 16 bits total; BYTE0 is the high 8 bits and BYTE1 is the low 8 bits.** Multi-turn output-shaft position, in radians (rad).	The actual position is a double and must be converted to a 16-bit integer as follows: pos_int = (pos_double + 12.5)*65535 / 25
BYTE2 BYTE3 BYTE4	**Velocity: 12 bits total. BYTE2 is the high 8 bits, and BYTE3[7:4] (upper 4 bits) is the low 4 bits. It represents output-shaft angular velocity in rad/s.** **KP: 12 bits total. BYTE3[3:0] (lower 4 bits) is the high 4 bits, and BYTE4 is the low 8 bits.**	The actual velocity is a double and must be converted to a 12-bit integer as follows: vel_int = (vel_double + 65) × 4095 / 130. The actual KP is a double and must be converted to a 12-bit integer as follows: kp_int = kp_double * 4095 / 500
BYTE5 BYTE6 BYTE7	**KD: 12 bits total. BYTE5 is the high 8 bits, and BYTE6[7:4] (upper 4 bits) is the low 4 bits.** **Torque: 12 bits total. BYTE6[3:0] (lower 4 bits) is the high 4 bits, and BYTE7 is the low 8 bits. Unit: N·m.**	**The actual KD is a double and must be converted to a 12-bit integer as follows:** kd_int = kd_double * 4095 / 5 The actual torque is a double and must be converted to a 12-bit integer as follows: t_int = (t_double + 50) × 4095 / 100. Torque-constant unit: N·m/A.

✓ Motor → Host➔

CAN Data-Frame Bits	Meaning	Description

BYTE0	node id	Driver node ID
BYTE1	**Position: 16 bits total; BYTE1 is the high 8 bits and BYTE2 is the low 8 bits.**	The actual position is a double and must be converted from the 16-bit integer as follows: pos_double = pos_int * 25 / 65535 – 12.5
BYTE2	Multi-turn output-shaft position, in radians (rad).	
BYTE3	**Velocity: 12 bits total. BYTE3 is the high 8 bits, and BYTE4[7:4] (upper 4 bits) is the low 4 bits. It represents output-shaft angular velocity in rad/s.**	The actual velocity is a double and must be converted from the 12-bit integer as follows: vel_double = vel_int × 130 / 4095 - 65.
BYTE4		
BYTE5	**Torque: 12 bits total. BYTE4[3:0] (lower 4 bits) is the high 4 bits, and BYTE5 is the low 8 bits. Unit: N·m.**	The actual torque is a double and must be converted from the 12-bit integer as follows: t_double = t_int × 100 / 4095 - 50. Torque-constant unit: N·m/A.

- Get_Encoder_Estimates

CMD ID: 0x009 (Motor → Host)→

Start Byte	Name	Category	Unit	Description
0	Pos_Estimate	float32	rev	odrv0.axis0.encoder.pos_estimate: current motor rotor position
4	Vel_Estimate	float32	rev/s	odrv0.axis0.encoder.vel_estimate: current motor rotor velocity

- Get_Encoder_Count

CMD ID: 0x00A (Motor → Host)→

Start Byte	Name	Category	Description
0	Shadow_Count	int32	odrv0.axis0.encoder.shadow_count Encoder multi-turn count
4	Count_In_Cpr	int32	odrv0.axis0.encoder.count_in_cpr Encoder single-turn count

- Set_Controller_Mode

CMD ID: 0x00B (Host → Motor)→

Start Byte	Name	Category	Description
0	Control_Mode	uint32	odrv0.axis0.controller.config.control_mode, control mode: 0: Voltage control 1: Torque control 2: Velocity control 3: Position control
4	Input_Mode	uint32	odrv0.axis0.controller.config.input_mode

			Input mode, defining how the user-entered control value drives the motor: 0: Inactive 1: Direct control 2: Velocity ramp 3: Position filter 5: Trapezoidal trajectory 6: Torque ramp 9: Motion control (MIT)

- Set_Input_Pos

CMD ID: 0x00C (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Input_Pos	float32	rev	odrv0.axis0.controller.input_pos : position-control target.
4	Vel_FF	int16	0.001rev/s	odrv0.axis0.controller.input_vel : velocity feedforward for position control.
6	Torque_FF	int16	0.001Nm	odrv0.axis0.controller.input_torque : torque feedforward for position control.

- Set_Input_Vel

CMD ID: 0x00D (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Input_Vel	float32	rev/s	odrv0.axis0.controller.input_vel : velocity-control target.
4	Torque_FF	float32	Nm	odrv0.axis0.controller.input_torque : torque feedforward for velocity control.

- Set_Input_Torque

CMD ID: 0x00E (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Input_Torque	float32	Nm	odrv0.axis0.controller.input_torque : torque-control target.

- Set_Limits

CMD ID: 0x00F (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Velocity_Limit	float32	rev/s	odrv0.axis0.controller.config.vel_limit : global velocity limit.

4	Current_Limit	float32	A	odrv0.axis0.motor.config.current_lim : motor current limit.

➢ Start_Anticogging

CMD ID: 0x010 (Host → Motor): perform torque-ripple calibration.→

➢ Set_Traj_Vel_Limit

CMD ID: 0x011 (Host → Motor)→

Start Byte	Name	Category	Unit	Description
0	Traj_Vel_Limit	float32	rev/s	odrv0.axis0.trap_traj.config.vel_limi t: velocity limit for trapezoidal position control.

➢ Set_Traj_Accel_Limits

CMD ID: 0x012 (Host → Motor)→

Start Byte	Name	Category	Unit	Description
0	Traj_Accel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.accel_l im it Acceleration limit for trapezoidal position control.
4	Traj_Decel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.decel _li mit Deceleration limit for trapezoidal position control.

➢ Set_Traj_Inertia

CMD ID: 0x013 (Host → Motor)→

Start Byte	Name	Category	Unit	Description
0	Traj_Inertia	float32	Nm/(rev/s^2)	odrv0.axis0.controller.config.inertia : inertia.

➢ Get_Iq

CMD ID: 0x014 (Motor → Host)→

Start Byte	Name	Category	Unit	Description
0	Iq_Setpoint	float32	A	odrv0.axis0.motor.current_control.Idq_se tp oint Q-axis current target.
4	Iq_Measured	float32	A	odrv0.axis0.motor.current_control.Iq_me as ured Measured Q-axis current.

- Reboot

CMD ID: 0x016 (Host → Motor)➔

- Get_Bus_Voltage_Current

CMD ID: 0x017 (Motor → Host)➔

Start Byte	Name	Category	Unit	Description
0	Bus_Voltage	float32	V	odrv0.vbus_voltage DC-bus voltage.
4	Bus_Current	float32	A	odrv0.ibus: DC-bus current.

- Clear_Errors

CMD ID: 0x018 (Host → Motor): clear all errors and faults.➔

- Set_Linear_Count

CMD ID: 0x019 (Host → Motor): set the encoder absolute position.➔

Start Byte	Name	Category	Description
0	Linear_Count	int32	odrv0.axis0.encoder.set_linear_count( ): set the encoder absolute position (count value).

- Set_Pos_Gain

CMD ID: 0x01A (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Pos_Gain	float32	(rev/s)/rev	odrv0.axis0.controller.config.pos_gain : position-loop gain Kp.

- Set_Vel_Gains

CMD ID: 0x01B (Host → Motor)➔

Start Byte	Name	Category	Unit	Description
0	Vel_Gain	float32	Nm/(rev/s)	odrv0.axis0.controller.config.vel_gain : velocity-loop gain Kp.
4	Vel_Integrat o r_Gain	float32	Nm/rev	odrv0. axis0.controller.config.vel_integrato r_ gain Velocity-loop gain Ki.

- Get_Torques

CMD ID: 0x01C (Motor → Host)➔

Start Byte	Name	Category	Unit	Description
0	Torque_Setpoint	float32	Nm	odrv0.axis0.controller.torque_setpoint : current-loop torque target.
4	Torque	float32	Nm	Indicates the current torque value.

- Get_Powers

CMD ID: 0x01D (Motor → Host)➔

Start Byte	Name	Category	Description
0	Electrical_Power	float32	odrv0.axis0.controller.electrical_power : electrical power
4	Mechanical_Power	float32	odrv0.axis0.controller.mechanical_power : mechanical power

- Disable_Can

CMD ID: 0x01E (Host → Motor)➔

Disable CAN and restart the driver.

- Save_Configuration

CMD ID: 0x01F (Host → Motor)➔

Save the current configuration, apply it, and restart.

4.1.3 CAN Protocol Examples

4.1.3.1 Example: CAN Message Sequence for Power-On Calibration

CAN ID	Frame Type	Frame Data	Description
0x007	Data Frame	04 00 00 00 00 00 00 00	Message: Set_Axis_State Parameter: 4 Calibrate the motor.
0x007	Data Frame	07 00 00 00 00 00 00 00	Message: Set_Axis_State Parameter: 7 Calibrate the encoder.

4.1.3.2 Example: CAN Message Sequence for Velocity Control

CAN ID	Frame Type	Frame Data	Description
0x00B	Data Frame	02 00 00 00 02 00 00 00	Message: Set_Controller_Mode

			Parameters: 2/2 Set the control mode to velocity control and the input mode to velocity ramp.
0x007	Data Frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameter: 8 Enter closed-loop control.
0x0D	Data Frame	00 00 20 41 00 00 00 00	Message: Set_Input_Vel Parameters: 10/0 Set the target velocity and torque feedforward. The target velocity is 10 (floating-point value: 0x41200000), and the torque feedforward is 0 (floating-point value: 0x00000000).

4.1.3.3 Example: CAN Message Sequence for Position Control

CAN ID	Frame Type	Frame Data	Description
0x00B	Data Frame	03 00 00 00 03 00 00 00	Message: Set_Controller_Mode Parameters: 3/3 Set the control mode to position control and the input mode to position filter.
0x007	Data Frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameter: 8 Enter closed-loop control.
0x0C	Data Frame	CD CC 0C 40 00 00 00 00	Message: Set_Input_Pos Parameters: 2.2/0/0 Set the target position, velocity feedforward, and torque feedforward. The target position is 2.2 (floating-point value: 0x400CCCCD), and both feedforward values are 0.

4.1.4 CANopen Compatibility

With appropriate node-ID assignment, this protocol can coexist with CANopen. The table below lists valid CANopen and protocol node-ID combinations:

CANOpen node IDs	Protocol 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 Periodic Messages

The motor can be configured to send messages periodically to the host without requiring request messages. A series of settings under odrv0.axis0.config.can enable or disable periodic messages. A value of 0 disables the message; any other value specifies the period in milliseconds, as shown below:

Message	odrivetool Setting	Default Value
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

By default, the first two periodic messages are enabled at the factory, so two messages are broadcast at their configured intervals when monitoring the CAN bus. Disable them with the following commands:

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

For details of each message, refer to Section 4.1.2.

4.2 CANopen Protocol

CANopen is a communication protocol for automation. This product is implemented in accordance with CiA 301, CiA 302, and CiA 402. Before using CANopen, switch the CAN protocol with the following odrivetool command:
CANOpen：

```
odrv0.can.config.protocol = 2
```

4.2.1 Electronic Data Sheet (EDS)

The EDS file describes device functions and communication parameters. To communicate with this product through CANopen, import the product EDS file. Download it from:
https://bl.cyberbeast.cn/actuator/cb_0.1.eds

4.2.2 Protocol Frame Format

CANopen is based on standard CAN. Its 11-bit ID contains a 4-bit function code and a 7-bit node ID (node_id), supporting up to 127 nodes on the bus; node 0 is reserved for broadcast. The frame format is shown below:

Data Field	COB-ID / CAN ID（11bits）		Data（8 bytes）
Segment	Bit10 ~ Bit7	Bit6 ~ Bit0	Byte0 ~ Byte7

Description	Function Code	node_id	Communication Data

The function-code table is as follows:

Communication Function	COB-ID	Description
NMT	0x000	Network Management
SYNC	0x080	Synchronization
EMCY	0x080+node_id	Emergency Management
TPDO1~TPDO4	0x180/0x280/0x380/0x480+node_id	Real-Time Process Data Transmit
RPDO1~RPDO4	0x200/0x300/0x400/0x500+node_id	Real-Time Process Data Receive
TSDO	0x600+node_id	Service Data Transmit
RSDO	0x580+node_id	Service Data Receive
NMT (Heartbeat)	0x700+node_id	Heartbeat

4.2.3 Frame Messages

The frame-message format varies according to the CAN ID (COB-ID).

4.2.3.1 NMT (Network Management)

NMT sends state-change commands to all nodes on the bus or to a specified node. Its COB-ID is always 0 and the data field contains two bytes, as shown below:

COB-ID	Data Byte0	Data Byte1
0x000	State Command Byte	node_id

The state command byte includes:

State Command Byte	Description
0x01	Change state to "Operational"
0x02	Change state to "Stopped"
0x80	Change state to "Pre-operational"
0x81	Reset node
0x82	Reset communication

For example, after connecting to the driver, send the following NMT command to start all nodes on the bus:

COB-ID	Data
0x000	01 00

4.2.3.2 SYNC

The synchronization function synchronizes all nodes on the bus, primarily for synchronizing PDO messages. Its COB-ID is fixed at 0x80 and the data field is empty.

For example, after using an NMT command to place the bus nodes into operation, a SYNC message is normally sent to start PDO transmission.

4.2.3.3 PDO

PDO is intended for real-time data transmission. Use PDO when periodically acquiring node data or when the node reports data in an event-driven manner. PDO is available only when the node is in the “operational” state, so the node state must first be changed using an NMT message.

The format of the PDO message data field is determined by its mapping structure. This mapping is not standardized and is defined in the EDS file.

4.2.3.4 SDO

SDO communication provides access to the product object dictionary (defined by the EDS). Object-dictionary data can be understood as the product parameters. In general, SDO is used only for configuration and not for runtime data exchange. PDO is recommended for real-time operating-data exchange.

SDO has two different processes: read and write. The following table shows the write process (also called upload):

	COB-ID	Data Byte0	Data Byte1	Data Byte2	Data Byte3	Data Byte4	Data Byte5	Data Byte6	Data Byte7
write	0x600+ node_id	0x23	index		subindex	data			
		0x27	index		subindex	data			/
		0x2B	index		subindex	data		/	/
		0x2F	index		subindex	data	/	/	/
response	0x580+ node_id	0x60	index		subindex	/	/	/	/
		0x80	Index		subindex	abortion code			

Read process (also called download):

	COB-ID	Data Byte0	Data Byte1	Data Byte2	Data Byte3	Data Byte4	Data Byte5	Data Byte6	Data Byte7
read	0x600+ node_id	0x40	index		subindex	/	/	/	/
response	0x580+ node_id	0x43	index		subindex	data			
		0x47	index		subindex	data			/
		0x4B	index		subindex	data		/	/
		0x4F	index		subindex	data	/	/	/
		0x80	index		subindex	abortion code			

4.2.3.5 NMT (Heartbeat)

The heartbeat message is used only to indicate the node status on the bus and confirm that the node is operating normally. The frame format is as follows:

COB-ID	Data Byte0
0x700+node_id	Status

4.3 RS485 Protocol (Modbus RTU)

Note: On hardware version 3.10 or later, the communication interface supports RS485, pulse/direction, and PWM. Use the following command to ensure that the communication interface is set to RS485:

```
odrv0.config.comm_intf_mux = 1
```

In Motor Wizard, the physical interface can also be switched under Communication Parameters.

RS485-based Modbus RTU supports a maximum communication rate of 115200 bps. The factory-default rate is 115200 bps.

4.3.1 Modbus RTU

Based on the industrial-standard Modbus RTU protocol, the following function codes are currently supported:

Function Code	Meaning
0x04	Read one or more input registers
0x03	Read multiple holding registers
0x06	Write a single holding register
0x10	Write multiple holding registers

4.3.2 Input Registers

Input registers are read-only and represent system status or dynamic values, such as error information, voltage, and current. The currently supported input registers are as follows:

➢**0x01: Motor State**

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 01	00 01	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	02	State0 State1	XX XX

State0 is always 0, and State1 is the motor state code:

Motor State Code	Description
1	Idle
3	Full Calibration
4	Motor Calibration

7	Encoder Calibration
8	Closed-Loop Control
11	Homing

➢**0x02: Error Flags**

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 02	00 01	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	02	Flag0 Flag1	XX XX

Flag0 is always 0. Flag1 contains the error flags; a bit value of 1 indicates the corresponding error listed below:

bit7~bit5	bit4	bit3	bit2	bit1	bit0
Reserved	Driver Error	System Error	Controller Error	Encoder Error	Motor Error

➢**0x03: Driver Error Code 1**

➢**0x04: Driver Error Code 2**

The driver error code is a 32-bit unsigned integer. Driver Error Code 1 contains the upper 16 bits, and Driver Error Code 2 contains the lower 16 bits. To obtain the complete driver error code, access register address 0x03 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 03	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes

XX	04	04	E0 E1 E2 E3	XX XX

E0, E1, E2, and E3 form the driver error code. Refer to the error-code table in Section 5.2.

➢**0x05: Motor Error Code 1**

➢**0x06: Motor Error Code 2**

➢**0x07: Motor Error Code 3**

➢**0x08: Motor Error Code 4**

The motor error code is a 64-bit unsigned integer. Motor Error Code 1 contains bits 63–48, Code 2 contains bits 47–32, Code 3 contains bits 31–16, and Code 4 contains bits 15–0. To obtain the complete motor error code, access register address 0x05 and read all four registers together. None of the registers may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 05	00 04	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	8 bytes	2 bytes
XX	04	08	E0 E1 E2 E3 E4 E5 E6 E7	XX XX

E0 through E7 form the motor error code. Refer to the error-code table in Section 5.2.

➢**0x09: Encoder Error Code 1**

➢**0x0A: Encoder Error Code 2**

The encoder error code is a 32-bit unsigned integer. Encoder Error Code 1 contains the upper 16 bits, and Encoder Error Code 2 contains the lower 16 bits. To obtain the complete encoder error code, access register address 0x09 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 09	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	E0 E1 E2 E3	XX XX

E0, E1, E2, and E3 form the encoder error code. Refer to the error-code table in Section 5.2.

➢**0x0B: Controller Error Code 1**

➢**0x0C: Controller Error Code 2**

The controller error code is a 32-bit unsigned integer. Controller Error Code 1 contains the upper 16 bits, and Controller Error Code 2 contains the lower 16 bits. To obtain the complete controller error code, access register address 0x0B and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 0B	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	E0 E1 E2 E3	XX XX

E0, E1, E2, and E3 form the controller error code. Refer to the error-code table in Section 5.2.

➢**0x0D: System Error Code 1**

➢**0x0E: System Error Code 2**

The system error code is a 32-bit unsigned integer. System Error Code 1 contains the upper 16 bits, and System Error Code 2 contains the lower 16 bits. To obtain the complete system error code, access register address 0x0D and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 0D	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC

1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	E0 E1 E2 E3	XX XX

E0, E1, E2, and E3 form the system error code. Refer to the error-code table in Section 5.2.

➢**0x0F: Position Feedback 1**

➢**0x10: Position Feedback 2**

Position feedback is a 32-bit floating-point value. Position Feedback 1 contains the upper 16 bits, and Position Feedback 2 contains the lower 16 bits. To obtain the complete position feedback value, access register address 0x0F and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 0F	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	P0 P1 P2 P3	XX XX

P0, P1, P2, and P3 form the position feedback value. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x11: Velocity Feedback 1**

➢**0x12: Velocity Feedback 2**

Velocity feedback is a 32-bit floating-point value. Velocity Feedback 1 contains the upper 16 bits, and Velocity Feedback 2 contains the lower 16 bits. To obtain the complete velocity feedback value, access register address 0x11 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 11	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC

1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 form the velocity feedback value. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x13: DC-Bus Voltage Feedback 1**

➢**0x14: DC-Bus Voltage Feedback 2**

The DC-bus voltage is a 32-bit floating-point value. DC-Bus Voltage Feedback 1 contains the upper 16 bits, and Feedback 2 contains the lower 16 bits. To obtain the complete DC-bus voltage value, access register address 0x13 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 13	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 form the DC-bus voltage value. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x15: DC-Bus Current Feedback 1**

➢**0x16: DC-Bus Current Feedback 2**

DC-bus current feedback is a 32-bit floating-point value. DC-Bus Current Feedback 1 contains the upper 16 bits, and Feedback 2 contains the lower 16 bits. To obtain the complete DC -bus current feedback value, access register address 0x15 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 15	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	I0 I1 I2 I3	XX XX

I0, I1, I2, and I3 form the DC-bus current feedback value. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x17: Q-Axis Current Setpoint 1**

➢**0x18: Q-Axis Current Setpoint 2**

The Q-axis current setpoint is a 32-bit floating-point value. Q-Axis Current Setpoint 1 contains the upper 16 bits, and Setpoint 2 contains the lower 16 bits. To obtain the complete Q-axis current setpoint, access register address 0x17 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 17	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	I0 I1 I2 I3	XX XX

I0, I1, I2, and I3 form the Q-axis current setpoint. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x19: Q-Axis Current Feedback 1**

➢**0x1A: Q-Axis Current Feedback 2**

Q-axis current feedback is a 32-bit floating-point value. Q-Axis Current Feedback 1 contains the upper 16 bits, and Feedback 2 contains the lower 16 bits. To obtain the complete Q-axis current feedback value, access register address 0x19 and read both registers together. Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 19	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	I0 I1 I2 I3	XX XX

I0, I1, I2, and I3 form the Q-axis current feedback value. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x31: Random Endpoint Access Return Value 1**
- **0x32: Random Endpoint Access Return Value 2**

Random endpoint access allows the user to read or write any supported parameter. An "endpoint" represents a parameter, and each parameter has a unique endpoint number. To read a parameter, obtain its endpoint number from the endpoint description file, write the endpoint number to holding register 0xA1, and then read input register 0x31 to obtain the parameter value. Refer to the description of holding register 0xA1 for the endpoint description file download address.

The random endpoint access return value is a 32-bit value and may represent a 16-bit integer, 32-bit integer, or 32-bit floating-point value. Return Value 1 contains the upper 16 bits, and Return Value 2 contains the lower 16 bits. The user must access register address 0x31 and read both registers together to obtain the complete return value (parameter value). Neither register may be accessed separately.

Host request:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	04	00 31	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	04	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 are the four bytes of the random endpoint access return value, from most significant to least significant. A 32-bit floating-point value is encoded according to IEEE 754 (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

4.3.3 Holding Registers

Holding registers support both reading and writing. The currently supported registers and functions are as follows:

- **0x61: Emergency Stop**

Writing any value to this register immediately stops the motor.

Host request:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 61	XX XX	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 61	XX XX	XX XX

➢**0x62: Node ID**

Use function code 06 to write the node ID to this register, or function code 03 to read the current node ID. Request to write the node ID:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 62	ID0 ID1	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 62	ID0 ID1	XX XX

ID0 and ID1 are the upper and lower 8 bits of the new node ID. Request to read the current node ID:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 62	00 01	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	02	ID0 ID1	XX XX

➢0x63: State Control

Use function code 06 to write the motor state to this register (change the motor state). Host request:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 63	S0 S1	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 63	S0 S1	XX XX

S0 and S1 are the upper and lower 8 bits of the motor state. The state definitions are as follows:

Motor State Code	Description
1	Idle
3	Full Calibration
4	Motor Calibration
7	Encoder Calibration
8	Closed-Loop Control
11	Homing

For example, setting the motor state to 8 (closed-loop control) is equivalent to “enabling the motor”; setting the motor state to 1 (idle) is equivalent to “disabling the motor.”

➢0x64: Control Mode

Use function code 06 to change this register and select torque, velocity, or position control, or use function code 03 to read the current control mode.

Request to change the control mode:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 64	M0 M1	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 64	M0 M1	XX XX

Request to read the control mode:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 64	00 01	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	02	M0 M1	XX XX

M0 and M1 are the upper and lower 8 bits of the control mode. Control modes are defined as follows:

Control Mode	Description
1	Torque Control
2	Velocity Control
3	Position Control

➢0x65: Input Mode

Use function code 06 to change this register and select the target-value input mode, such as direct input or filtered input, or use function code 03 to read the current input mode.

Request to change the input mode:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 65	M0 M1	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 65	M0 M1	XX XX

Request to read the input mode:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 65	00 01	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	02	M0 M1	XX XX

M0 and M1 are the upper and lower 8 bits of the input mode. Input modes are defined as follows:

Control Mode	Description	Description
1	Direct Input	The user-entered target value is used directly as the driver target value.
2	Velocity Ramp	The driver velocity setpoint reaches the user-entered target velocity over time at a defined ramp rate. Used only for velocity control.
3	Position Filter	The driver position setpoint is the filtered value of the user-entered target position. Used only for position control.
5	Trapezoidal Trajectory	The driver moves the motor to the user-entered target using a trapezoidal velocity profile. The rising slope (acceleration), falling slope (deceleration), and cruise velocity can all be configured. Refer to registers 0x70–0x75 below.
6	Torque Ramp	The driver torque setpoint reaches the user-entered target torque over time at a defined ramp rate. Used only for torque control.

➢**0x66: Position-Control Setpoint 1**

➢**0x67: Position-Control Setpoint 2**

The position-control setpoint is a 32-bit floating-point value. Setpoint 1 contains the upper 16 bits, and Setpoint 2 contains the lower 16 bits. To read or change the complete position-control setpoint, access register address 0x66 and read/write both registers together. Neither register may be accessed separately.

Request to change/update the position-control setpoint:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes

XX	10	00 66	00 02	04	P0 P1 P2 P3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 66	00 02	XX XX

Request to read the position-control setpoint:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 66	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	P0 P1 P2 P3	XX XX

P0, P1, P2, and P3 are the four bytes of the position-control setpoint, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x68: Velocity-Control Setpoint 1**
- **0x69: Velocity-Control Setpoint 2**

The velocity-control setpoint is a 32-bit floating-point value. Setpoint 1 contains the upper 16 bits, and Setpoint 2 contains the lower 16 bits. To read or change the complete velocity-control setpoint, access register address 0x68 and read/write both registers together. Neither register may be accessed separately.

Request to change/update the velocity-control setpoint:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 68	00 02	04	V0 V1 V2 V3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes

XX	10	00 68	00 02	XX XX

Request to read the velocity-control setpoint:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 68	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 are the four bytes of the velocity-control setpoint, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x6A: Torque-Control Setpoint 1**
- **0x6B: Torque-Control Setpoint 2**

The torque-control setpoint is a 32-bit floating-point value. Setpoint 1 contains the upper 16 bits, and Setpoint 2 contains the lower 16 bits. To read or change the complete torque-control setpoint, access register address 0x6A and read/write both registers together. Neither register may be accessed separately.

Request to change/update the torque-control setpoint:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 6A	00 02	04	T0 T1 T2 T3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 6A	00 02	XX XX

Request to read the torque-control setpoint:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes

XX	03	00 6A	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	T0 T1 T2 T3	XX XX

T0, T1, T2, and T3 are the four bytes of the torque-control setpoint, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x6C: Velocity Limit 1**

➢**0x6D: Velocity Limit 2**

The velocity limit is a 32-bit floating-point value. Velocity Limit 1 contains the upper 16 bits, and Velocity Limit 2 contains the lower 16 bits. To read or change the complete velocity limit, access register address 0x6C and read/write both registers together. Neither register may be accessed separately.

Request to change/update the velocity limit:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 6C	00 02	04	V0 V1 V2 V3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 6C	00 02	XX XX

Request to read the velocity limit:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 6C	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes

XX	03	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 are the four bytes of the velocity limit, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x6E: Current Limit 1**

➢**0x6F: Current Limit 2**

The current limit is a 32-bit floating-point value. Current Limit 1 contains the upper 16 bits, and Current Limit 2 contains the lower 16 bits. To read or change the complete current limit, access register address 0x6E and read/write both registers together. Neither register may be accessed separately.

Request to change/update the current limit:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 6E	00 02	04	I0 I1 I2 I3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 6E	00 02	XX XX

Request to read the current limit:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 6E	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	I0 I1 I2 I3	XX XX

I0, I1, I2, and I3 are the four bytes of the current limit, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x70: Trapezoidal Position-Control Velocity Limit 1**

➢**0x71: Trapezoidal Position-Control Velocity Limit 2**

The trapezoidal position-control velocity limit is a 32-bit floating-point value. Limit 1 contains the upper 16 bits, and Limit 2 contains the lower 16 bits. To read or change the complete velocity limit, access register address 0x70 and read/write both registers together. Neither register may be accessed separately.

Request to change/update the trapezoidal position-control velocity limit:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 70	00 02	04	V0 V1 V2 V3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 70	00 02	XX XX

Request to read the trapezoidal position-control velocity limit:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 70	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	V0 V1 V2 V3	XX XX

V0, V1, V2, and V3 are the four bytes of the trapezoidal position-control velocity limit, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x72: Trapezoidal Position-Control Acceleration Limit 1**
- **0x73: Trapezoidal Position-Control Acceleration Limit 2**

The trapezoidal position-control acceleration limit is a 32-bit floating-point value. Limit 1 contains the upper 16 bits, and Limit 2 contains the lower 16 bits. To read or change the complete acceleration limit, access register address 0x72 and read/write both registers together. Neither register may be accessed separately.

Request to change/update the trapezoidal position-control acceleration limit:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC

1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 72	00 02	04	A0 A1 A2 A3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 72	00 02	XX XX

Request to read the trapezoidal position-control acceleration limit:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 72	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	A0 A1 A2 A3	XX XX

A0, A1, A2, and A3 are the four bytes of the trapezoidal position-control acceleration limit, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x74: Trapezoidal Position-Control Deceleration Limit 1**
- **0x75: Trapezoidal Position-Control Deceleration Limit 2**

The trapezoidal position-control deceleration limit is a 32-bit floating-point value. Limit 1 contains the upper 16 bits, and Limit 2 contains the lower 16 bits. To read or change the complete deceleration limit, access register address 0x74 and read/write both registers together. Neither register may be accessed separately.

Request to change/update the trapezoidal position-control deceleration limit:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 74	00 02	04	D0 D1 D2 D3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC

1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 74	00 02	XX XX

Request to read the trapezoidal position-control deceleration limit:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 74	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	D0 D1 D2 D3	XX XX

D0, D1, D2, and D3 are the four bytes of the trapezoidal position-control deceleration limit, from most significant to least significant. The value is a 32-bit floating-point number encoded in IEEE 754 format.

is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

- **0x76: Inertia 1**
- **0x77: Inertia 2**

Inertia is a 32-bit floating-point value. Inertia 1 contains the upper 16 bits, and Inertia 2 contains the lower 16 bits. To read or change the complete inertia value, access register address 0x76 and read/write both registers together. Neither register may be accessed separately.

Request to change/update inertia:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 76	00 02	04	I0 I1 I2 I3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 76	00 02	XX XX

Request to read inertia:

Address	Function Code	Register Address	Number of Registers	CRC

1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 76	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	I0 I1 I2 I3	XX XX

I0, I1, I2, and I3 are the four bytes of the inertia value, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x78: Position Gain 1**

➢**0x79: Position Gain 2**

Position gain is a 32-bit floating-point value. Position Gain 1 contains the upper 16 bits, and Position Gain 2 contains the lower 16 bits. To read or change the complete position gain, access register address 0x78 and read/write both registers together. Neither register may be accessed separately.

Request to change/update position gain:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 78	00 02	04	G0 G1 G2 G3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 78	00 02	XX XX

Request to read position gain:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 78	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes

XX	03	04	G0 G1 G2 G3	XX XX

G0, G1, G2, and G3 are the four bytes of the position gain, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x7A: Velocity Gain 1**

➢**0x7B: Velocity Gain 2**

Velocity gain is a 32-bit floating-point value. Velocity Gain 1 contains the upper 16 bits, and Velocity Gain 2 contains the lower 16 bits. To read or change the complete velocity gain, access register address 0x7A and read/write both registers together. Neither register may be accessed separately.

Request to change/update velocity gain:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 7A	00 02	04	G0 G1 G2 G3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 7A	00 02	XX XX

Request to read velocity gain:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 7A	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	G0 G1 G2 G3	XX XX

G0, G1, G2, and G3 are the four bytes of the velocity gain, from most significant to least significant. The 32-bit floating-point value is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x7C: Velocity Integrator Gain 1**

➢**0x7D: Velocity Integrator Gain 2**

Velocity integrator gain is a 32-bit floating-point value. Gain 1 contains the upper 16 bits, and Gain 2 contains the lower 16 bits. To read or change the complete velocity integrator gain, access register address 0x7C and read/write both registers together. Neither register may be accessed separately.

Request to change/update velocity integrator gain:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 7C	00 02	04	G0 G1 G2 G3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 7C	00 02	XX XX

Request to read velocity integrator gain:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	03	00 7C	00 02	XX XX

Motor response:

Address	Function Code	Byte Count	Register Value	CRC
1 byte	1 byte	1 byte	2 bytes	2 bytes
XX	03	04	G0 G1 G2 G3	XX XX

G0, G1, G2, and G3 are the four bytes of the velocity integrator gain, from most significant to least significant. The 32-bit floating-point value is encoded according to IEEE 754 (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

➢**0x7E: Reboot**

Writing any value to this register reboots the motor driver.

Host request:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 7E	XX XX	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 7E	XX XX	XX XX

➢**0x7F: Clear Errors**

Writing any value to this register clears all errors and exceptions. Host request:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 7F	XX XX	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 7F	XX XX	XX XX

➢**0x80: Save Configuration**

Writing any value to this register saves all current parameters permanently to the driver Flash memory. Host request:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 80	XX XX	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 80	XX XX	XX XX

➢**0xA1: Random Endpoint Access — Endpoint Number**

Random endpoint access allows the user to read or write any supported parameter. An “endpoint” represents a parameter, and each parameter has a unique endpoint number. To read or write a parameter, obtain its endpoint number from the endpoint description file and write it to holding register 0xA1. Then read input register 0x31 to obtain the parameter value, or write holding register 0xA2 to update the parameter value.

The endpoint number is a 16-bit unsigned integer. Download the JSON file containing the endpoint numbers for all parameters and interface functions:

- ✓Version 0.5.13: https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.json
- ✓Version 0.5.14: https://bl.cyberbeast.cn/actuator/endpoints_0.5.14.json

Request to write a random endpoint number:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 A1	E0 E1	XX XX

Motor response:

Address	Function Code	Register Address	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	06	00 A1	E0 E1	XX XX

E0 and E1 are the two bytes of the 16-bit endpoint number, from most significant to least significant.

- **0xA2: Random Endpoint Access Write Value 1**
- **0xA3: Random Endpoint Access Write Value 2**

The random endpoint access write value is a 32-bit value and may represent a 16-bit integer, 32-bit integer, or 32-bit floating-point value. Write Value 1 contains the upper 16 bits, and Write Value 2 contains the lower 16 bits. The user must access register address 0xA2 and write both registers together to update the endpoint value (parameter value). Neither register may be accessed separately.

Write request:

Address	Function Code	Register Address	Number of Registers	Byte Count	Register Value	CRC
1 byte	1 byte	2 bytes	2 bytes	1 byte	4 bytes	2 bytes
XX	10	00 A2	00 02	04	V0 V1 V2 V3	XX XX

Motor response:

Address	Function Code	Register Address	Number of Registers	CRC
1 byte	1 byte	2 bytes	2 bytes	2 bytes
XX	10	00 A2	00 02	XX XX

V0, V1, V2, and V3 are the four bytes of the write value, from most significant to least significant. If the endpoint is a 32-bit floating-point value, it is encoded in IEEE 754 format (encoding can be tested at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

4.3.4 Modbus Protocol

Examples

Assume the motor address is 1.

1) Example: The frame sequence for velocity control is as follows:

Frame Data	Register	Description
01 06 00 64 00 02 XX XX	Holding Register 0x64	Set the control mode to 2 (velocity control).
01 06 00 65 00 02 XX XX	Holding Register 0x65	Set the input mode to 2 (velocity ramp).
01 06 00 63 00 08 XX XX	Holding Register 0x63	Change the motor state to 8 (closed-loop control).
01 10 00 68 00 02 04 40 13 33 33 XX XX	Holding Register 0x68	Set the target velocity to 2.3 turn/s. The corresponding 32-bit floating-point value is 40 13 33 33.

2) Example: The frame sequence for position control is as follows:

Frame Data	Register	Description
01 06 00 64 00 03 XX XX	Holding Register 0x64	Set the control mode to 3 (position control).
01 06 00 65 00 03 XX XX	Holding Register 0x65	Set the input mode to 3 (position filter).
01 06 00 63 00 08 XX XX	Holding Register 0x63	Change the motor state to 8 (closed-loop control).
01 10 00 68 00 02 04 40 49 0F F9 XX XX	Holding Register 0x68	Set the target position to 3.1416 turns. The corresponding 32-bit floating-point value is 40 49 0F F9.

4.4 PWM Input Control

Confirm that the hardware version is 3.10 or later before using this function.

This function supports control of writable parameters, such as position, velocity, or torque, using a standard RC PWM signal (1–2 ms pulse width, 50 Hz). Configure the GPIO pin for PWM

mode and set the mapping relationship to enable direct motor control from an RC receiver or other PWM controller.

4.4.1 Wiring

Connect the PWM signal wire to the CANH position described in Section 2.4.1. The interface supports PWM signals of 5 V or less. The PWM signal ground and motor-driver ground must be connected together.

4.4.2 Software Configuration

➢Switch the communication interface to PWM:

```
odrv0.config.comm_intf_mux = 3
```

In Motor Wizard, the physical interface can also be switched under Communication Parameters. Open Motor Parameters to locate this setting.

➢Set the GPIO to PWM mode

```
odrv0.config.gpio1_mode = GPIO_MODE_PWM
```

➢Set the PWM mapping control mode

The PWM input can be mapped to velocity, position, or torque control. Configure only one of the following modes as required.

Velocity Mode

Set the motor control mode to velocity-ramp mode (control_mode = 2, input_mode = 2).

```
odrv0.config.gpio1_pwm_mapping.endpoint =
odrv0.axis0.controller._input_vel_property
```

Position Mode

Set the motor control mode to trapezoidal-trajectory mode (control_mode = 3, input_mode = 5).

```
odrv0.config.gpio1_pwm_mapping.endpoint =
odrv0.axis0.controller._input_pos_property
```

Torque Mode

Set the motor control mode to torque-ramp mode (control_mode = 1, input_mode = 6).

```
odrv0.config.gpio1_pwm_mapping.endpoint =
odrv0.axis0.controller._input_torque_property
```

➢Set the PWM mapping parameters

```
odrv0.config.gpio1_pwm_mapping.min = -8.0
odrv0.config.gpio1_pwm_mapping.max = 8.0
```

min and max define the control-value range mapped from the PWM pulse width (1–2 ms). The driver linearly maps the pulse width to a real value within [min, max] and writes it to the configured endpoint control setpoint. min and max may be set to any required values.

➢Save the configuration (optional)

```
odrv0.save_configuration()
```

After completing the above configuration, place the motor in closed-loop control to begin PWM control.

4.5 Pulse/Direction Control

Confirm that the hardware version is 3.10 or later before using this function. This function conflicts with the limit-switch function. Pulse/direction is the simplest method of controlling the driver, but also the most basic and least robust. Do not use this control method unless specifically required.

This function supports conventional Step/Dir (pulse/direction) control of a brushless motor, providing compatibility with the control method used by standard stepper-motor drivers. In this mode:

Step signal: Each rising edge moves the motor by one preset increment (determined by steps_per_circular_range). Maximum recommended Step frequency: 50 kHz.

Dir signal: The logic level determines the direction (high = clockwise / low = counterclockwise).

4.5.1 Wiring

Connect the Step signal wire to the CANH position described in Section 2.4.1, and connect the Dir signal wire to the CANL position. The interface supports pulse signals of 5 V or less. Ensure that the Step and Dir signal grounds are connected to the motor-driver ground; otherwise, the signals may not be detected.

4.5.2 Software Configuration

➢Switch the communication interface to Step/Dir:

```
odrv0.config.comm_intf_mux = 2
```

In Motor Wizard, the physical interface can also be switched under Communication Parameters. Open Motor Parameters to locate this setting.

➢Set the GPIO to Step/Dir mode

```
odrv0.axis0.config.step_gpio_pin = 6
odrv0.axis0.config.dir_gpio_pin = 7
```

➢Configure the Step/Dir pins

```
odrv0.config.gpio6_mode = GPIO_MODE_DIGITAL_PULL_UP
odrv0.config.gpio7_mode = GPIO_MODE_DIGITAL_PULL_UP
```

➢Enable Step/Dir

```
odrv0.axis0.config.enable_step_dir = True
```

To exit pulse/direction control, set enable_step_dir to False, enter the idle state, and then return to closed-loop control.

➢Enable Circular Setpoint (optional)

To reduce precision drift when handling large target-position ranges, ODrive provides circular setpoint support. In this mode, the target position wraps around, keeping floating-point error within a manageable range. If the commanded position exceeds the circular range, the setpoint automatically wraps back into the range.

Enable the controller circular setpoint mode

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

Set the circular position range (unit: turns). The setpoint is limited to 0 through this range and wraps when it exceeds the range.

```
# 128 turns per closed loop
odrv0.axis0.controller.config.circular_setpoint_range = 128.0
```

Set the number of STEP pulses required over the circular position range. This determines the displacement represented by one step.

```
 # 1024 steps per turn
odrv0.axis0.controller.config.steps_per_circular_range = 128 * 1024
```

➢Save the configuration (optional)

```
odrv0.save_configuration()
```

After completing the above configuration, place the motor in closed-loop position control to begin pulse/direction control.

4.6 SDK

4.6.1 C/C++ SDK

A C/C++ SDK based on the CAN Simple protocol can be downloaded for integration: https://gitee.com/cyberbeast/mwmotorsdk. Refer to the detailed test examples on the open-source site for integration testing.

4.6.2 Python SDK

First install odrivetool as described in Section 3.1 (pip install --upgrade odrive). Refer to Section 3.1.8; all commands described there may be used for Python development.

The following three examples are provided:

1) Example: Power-On Calibration

```
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()
```

2) Example: Velocity Control

```
import odrive
import time

odrv0 = odrive.find_any()
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.VELOCITY_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.VEL_RAMP
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
```

3) Example: Position Control

```
import odrive

odrv0 = odrive.find_any()
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.POSITION_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.POS_FILTER
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_pos=10
```

4) Example: Data Acquisition

During development and integration, users often need to collect motor operating data, such as voltage/current and position/velocity changes. The Python SDK includes powerful data-capture capabilities, allowing large volumes of operating data to be collected with simple scripts and making development and integration easier.

The following code adds real-time position and current data capture to the previous position -control example and saves the data to a CSV file.

```
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=',')
```

In the BulkCapture statement, data_rate is the sampling frequency in Hz, duration is the sampling time in seconds, and the lambda expression may contain any mathematical operation required for data analysis.

4.6.3 Arduino SDK

Arduino users can control the motor over the CAN bus. The underlying protocol is described in Section 4.1. Compatible hardware/libraries include:

✓Arduino boards with an integrated CAN interface, such as Arduino UNO R4 Minima and Arduino UNO R4 WiFi

✓Teensy development boards with an integrated CAN interface can use the compatible FlexCAN_T4 library (Teensy 4.0 and Teensy 4.1).

✓Other Arduino-compatible boards can use an MCP2515-based CAN

expansion board. The following example shows how to configure the motor

to respond to Arduino position-control commands:

➢Configure the motor

In addition to the basic configuration in Section 3.1.3, set the control bandwidth to 20 rad/s. (The Arduino Uno transmission rate is limited, so a high control bandwidth is unnecessary. A higher value may be used with a faster Arduino.)

➢Configure CAN

Configure CAN as follows:

➢Install the ODriveArduino library

Install the ODriveArduino library as follows (assuming Arduino IDE is already installed):

1) Open Arduino IDE.

2) Sketch -> Include Library -> Manage Libraries

3) Enter “ODriveArduino” in the search field.

4) Select and install the ODriveArduino library from the search results.

➢Arduino Source Code

```
#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);
  }
}
```

4.6.4 ROS SDK

The following procedure has been tested on Ubuntu 23.04 and ROS 2 Iron. It does not support macOS or Windows and has not been verified with other ROS 2 versions; modifications may be required.

4.6.4.1 Installing the odrive_can Package

1. Create a ROS 2 workspace (refer to https://docs.ros.org/en/iron/index.html).
2. Run git clone https://github.com/odriverobotics/odrive_can to download the code into the src directory of the workspace created above.
3. In a terminal, navigate to the workspace root directory and run:

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

4. Prepare the environment before running:

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

5. Run the example node:

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

4.6.4.2 Calling Services and Viewing Messages

Assume that the odrive_can_node node above is running in the odrive_axis0 namespace (configurable
in ./launch/example_launch.yaml). Once the node in Section 4.6.4.1 is running, published topic messages can be viewed, for example:

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

The available service interfaces can also be called. For example, the following call starts motor calibration:

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

5 FAQs and Error Codes (To Be Updated)

5.6 Frequently Asked Questions (FAQ)

5.7 Error Codes

Error Category	Error Code	odrivetool Display	Description
System Error	0x00000002	DC_BUS_UNDER_VOLTAGE	DC-Bus Undervoltage
	0x00000004	DC_BUS_OVER_VOLTAGE	DC-Bus Overvoltage
	0x00000008	DC_BUS_OVER_REGEN_CURRENT	Excessive Reverse (Charging) DC-Bus Current
	0x00000010	DC_BUS_OVER_CURRENT	Excessive Forward (Discharging) DC-Bus Current
Driver Error	0x00000001	INVALID_STATE	Driver State Error
	0x00000040	MOTOR_FAILED	Motor Error
	0x00000100	ENCODER_FAILED	Encoder Error
	0x00000200	CONTROLLER_FAILED	Controller Error
	0x00001000	MIN_ENDSTOP_PRESSED	Minimum Limit Switch Triggered
	0x00002000	MAX_ENDSTOP_PRESSED	Maximum Limit Switch Triggered
	0x00004000	ESTOP_REQUESTED	Emergency Stop
	0x00020000	HOMING_WITHOUT_ENDSTOP	Homing Without a Limit

			Switch
	0x00080000	UNKNOWN_POSITION	No Position Information
	0x00000001	PHASE_RESISTANCE_OUT_OF_RANGE	Phase Resistance Outside the Normal Range

Motor Error	0x00000002	PHASE_INDUCTANCE_OUT_OF_RANGE	Phase Inductance Outside the Normal Range
	0x00000010	CONTROL_DEADLINE_MISSED	FOC Frequency Too High
	0x00000080	MODULATION_MAGNITUDE	SVM Modulation Error
	0x00000400	CURRENT_SENSE_SATURATION	Phase-Current Saturation
	0x00001000	CURRENT_LIMIT_VIOLATION	Motor Overcurrent
	0x00020000	MOTOR_THERMISTOR_OVER_TEMP	Motor Overtemperature
	0x00040000	FET_THERMISTOR_OVER_TEMP	Driver Overtemperature
	0x00080000	TIMER_UPDATE_MISSED	FOC Processing Overrun
	0x00100000	CURRENT_MEASUREMENT_UNAVAILABLE	Phase-Current Sample Missing
	0x00200000	CONTROLLER_FAILED	Controller Error
	0x00400000	I_BUS_OUT_OF_RANGE	DC-Bus Current Limit Exceeded
	0x00800000	BRAKE_RESISTOR_DISARMED	Brake-Resistor Driver Error
	0x01000000	SYSTEM_LEVEL	System-Level Error
	0x02000000	BAD_TIMING	Phase-Current Sampling Delay
	0x04000000	UNKNOWN_PHASE_ESTIMATE	Motor Position Unknown
	0x08000000	UNKNOWN_PHASE_VEL	Motor Velocity Unknown
	0x10000000	UNKNOWN_TORQUE	Torque Unknown
	0x20000000	UNKNOWN_CURRENT_COMMAND	Torque Control Unknown
	0x40000000	UNKNOWN_CURRENT_MEASUREMENT	Current Sample Unknown
	0x80000000	UNKNOWN_VBUS_VOLTAGE	Voltage Sample Unknown
	0x100000000	UNKNOWN_VOLTAGE_COMMAND	Voltage Control Unknown
	0x200000000	UNKNOWN_GAINS	Current-Loop Gain Unknown
	0x400000000	CONTROLLER_INITIALIZING	Controller Initialization Error
	0x800000000	UNBALANCED_PHASES	Three-Phase Imbalance
Controller Error	0x00000001	OVERSPEED	Overspeed
	0x00000002	INVALID_INPUT_MODE	Incorrect Control Input Mode
	0x00000004	UNSTABLE_GAIN	Unstable PLL Gain
	0x00000020	INVALID_ESTIMATE	Unstable Position/Velocity
	0x00000080	SPINOUT_DETECTED	Mechanical and Electrical Power Mismatch (incorrect encoder calibration or unstable magnets)
Encoder Error	0x00000001	UNSTABLE_GAIN	Encoder Bandwidth Too High
	0x00000002	CPR_POLEPAIRE_MISMATCH	CPR and Pole-Pair Mismatch
	0x00000004	NO_RESPONSE	Encoder Not Responding
	0x00000400	SEC_ENC_COM_FAIL	Secondary Encoder Communication Error