SIG-Horse Geared Motor Drive Instruction Manual

Catalogs

Release Notes

version number	dates	Revisions/notes
0.1	2023.12.5	Support for the first version of odrivetool
0.2	2023.12.15	Add command list and command category
0.3	2023.12.19	Add Python, Arduino, ROS SDKs
0.4	2023.12.23	Add description of switching between USB and CAN compatibility
0.5	2023.12.29	Increase the upper computer tuning test real-world cases
0.6	2024.1.2	Adding CAN protocol and Python tuning real-world examples
0.7	2024.1.8	Add CAN interface 120R matching resistor switch option
0.8	2024.2.15	Addition of the second encoder and user zero setting
0.9	2024.2.23	Add CAN commands to modify parameters and call interface functions.
0.91	2024.3.12	Add the driver download address for the national download software
0.92	2024.3.22	Add a description of the CAN default baud rate and a description of the initial rotor position range with the second encoder acting.
1.0	2024.3.26	Adding motor temperature protection instructions
1.1	2024.7.2	Add error code table
1.2	2024.8.18	Add instructions for modifying IDs
1.3	2024.8.21	Instructions for adding a brake resistor
1.4	2024.8.31	Add different versions of the endpoints.json file downloads
1.5	2024.9.24	Addition of instructions for the use of the host computer "Motor Wizard".

caveat

1. Please use the product in accordance with the operating parameters of this manual, otherwise irreversible damage will be caused to the product!
2. During the operation of the motor, please take good measures to protect the power supply from over-current and over-voltage, so as not to damage the drive.
3. Please check the parts are intact before use, if any parts are missing or damaged, please contact technical support in time.
4. The drive is not reverse connection proof, please refer to section 2.4.1 to ensure the power supply is correctly positive and negative before connecting the power supply.
5. Do not touch the exposed part of the drive with your hands to avoid static damage!

Legal Notices

Before using this product, please be sure to read this manual carefully and operate this product according to the contents. If the user uses this product in violation of the contents of the manual, we will not be responsible for any property damage or personal injury caused. As this product consists of many parts, do not allow children to come into contact with it to avoid accidents. To prolong product life, do not use this product in a high-temperature, high-pressure environment. This manual has been printed to include as much information as possible about the functions and instructions for use. However, due to continuous improvement of product functions, design changes, etc., there may be discrepancies with the product purchased by the user.

This manual may differ from the actual product in terms of color, appearance, etc., so please refer to the actual product. We may make necessary improvements and changes to this manual for typographical errors, inaccurate and up-to-date information, or improvements to programs and/or equipment at any time without prior notice. Such changes will be uploaded to a new version of this manual, which can be obtained by contacting Technical Support. All illustrations are for illustrative purposes only and are subject to change without notice.

After-sales policy

This product after-sales service is strictly based on the "Law of the People's Republic of China on the Protection of Consumers' Rights and Interests", "Product

Quality Law of the People's Republic of China" to implement the after-sales service, the service is as follows:

1. Warranty period and content

    1) Users who place an order for this product through an online channel are entitled to return it without a reason within seven days of the date of receipt. Users must present valid proof of purchase and return the invoice when returning the product. Users must ensure that the returned products maintain their original quality and functionality, are in good condition, and that the trademarks and logos of the product itself and its accessories are intact, and that free gifts, if any, are returned as well. If the product has been damaged or dismantled, or if the box is missing, or if parts are missing, the product will not be returned. The user is responsible for any logistics costs incurred in returning the product. If the user does not settle the logistics costs, the amount of the refund will be deducted from the actual amount of growth. The refund will be made within seven days from the date of receipt of the returned product. The method of refund is the same as the method of payment. The exact date of receipt may be affected by factors such as banks and supporting organizations.

    2) If non-manufactured damage or malfunction occurs within 7 days from the day after the user signs for the product and is confirmed by our service center, the product will be returned to the user, and the user will be required to show proof of purchase and return the invoice. If there is a free gift, it must be returned as well.

    3) If non-manufactured damage or malfunction occurs within 15 days of the date of receipt by the customer, the customer will be entitled to an exchange of the entire set of product after the Center's inspection and confirmation of non-manufactured damage. After the exchange, the product's three-year warranty period will be recalculated.

    4) Repair service is provided free of charge after 15 days from the next date of receipt by the user to within 365 days after the product has been tested and confirmed by our after-sales service center to be of the product's own quality failure. Replacement of defective products will be the property of AUO. Products that are not faulty will be returned in the same condition. If the product is not faulty, we reserve the right to refuse the return of the product after strict testing.

If this manual's after-sale policy is different from the store's after-sale policy, the store's after-sale policy shall prevail.

2. Non-warranty regulations

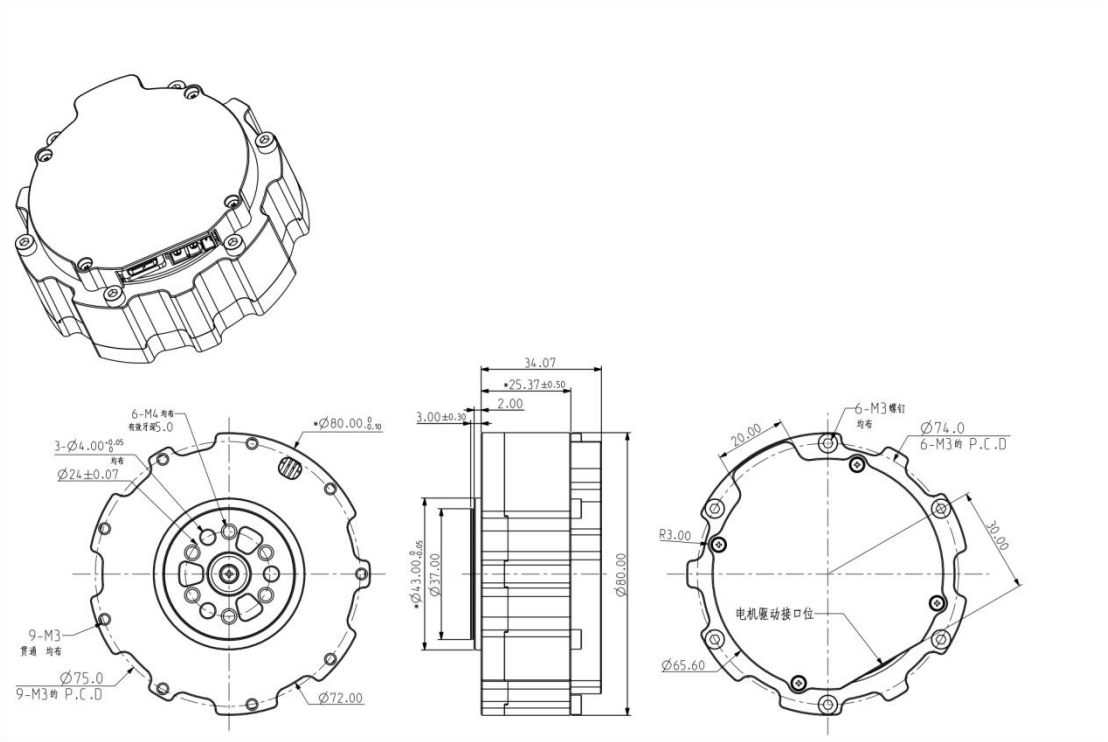
The following are not covered by the warranty:

1) Exceeds the warranty period limited by the terms of the warranty.
2) Damage caused by misuse of the product not in accordance with the instruction manual.
3) Damage caused by improper operation, maintenance, installation, modification, testing, or other improper use.
4) Instead of regular mechanical wear and tear caused by quality failures.
5) Damage caused by non-normal conditions, including but not limited to drops, impacts, liquid immersion, severe impacts, etc.
6) Damage caused by acts of God (e.g. flooding, fire, lightning, earthquake, etc.) or acts of force majeure.
7) Damage caused by exceeding peak torque.
8) It is not our original genuine product or no legal proof of purchase can be provided.
9) Failure or damage caused by other non-product design, technology, manufacturing, quality and other issues.
10) Damage caused by private disassembly of this product.

In such cases, users are required to support expenses on their own.

1 Motor specification parameters

1.1 Drawings and Dimensions



1.2 Electrical Characteristics

rated speed	170rpm±10%
Maximum speed	490rpm±10%
Rated torque	6.5N.m
Blocking torque	18N.m
rated current	16.5A
Plugging current	50A
No-load current	0.05A
phase resistance	0.158Ω
phase inductance	107μH
RPM constant	104rpm/v
torque constant	0.042N.M/A

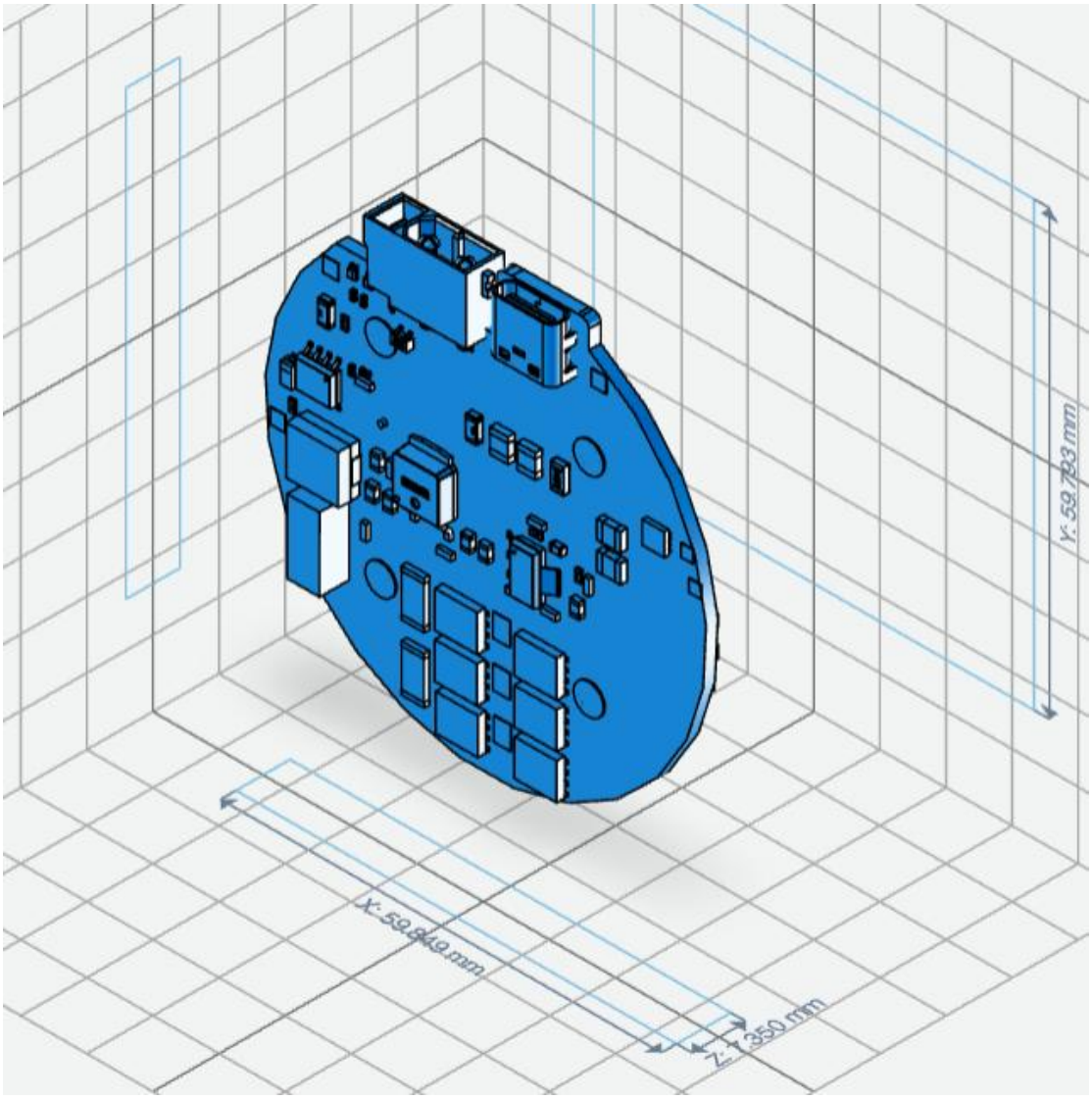
The characteristic curve is shown below:

1.3 mechanical property

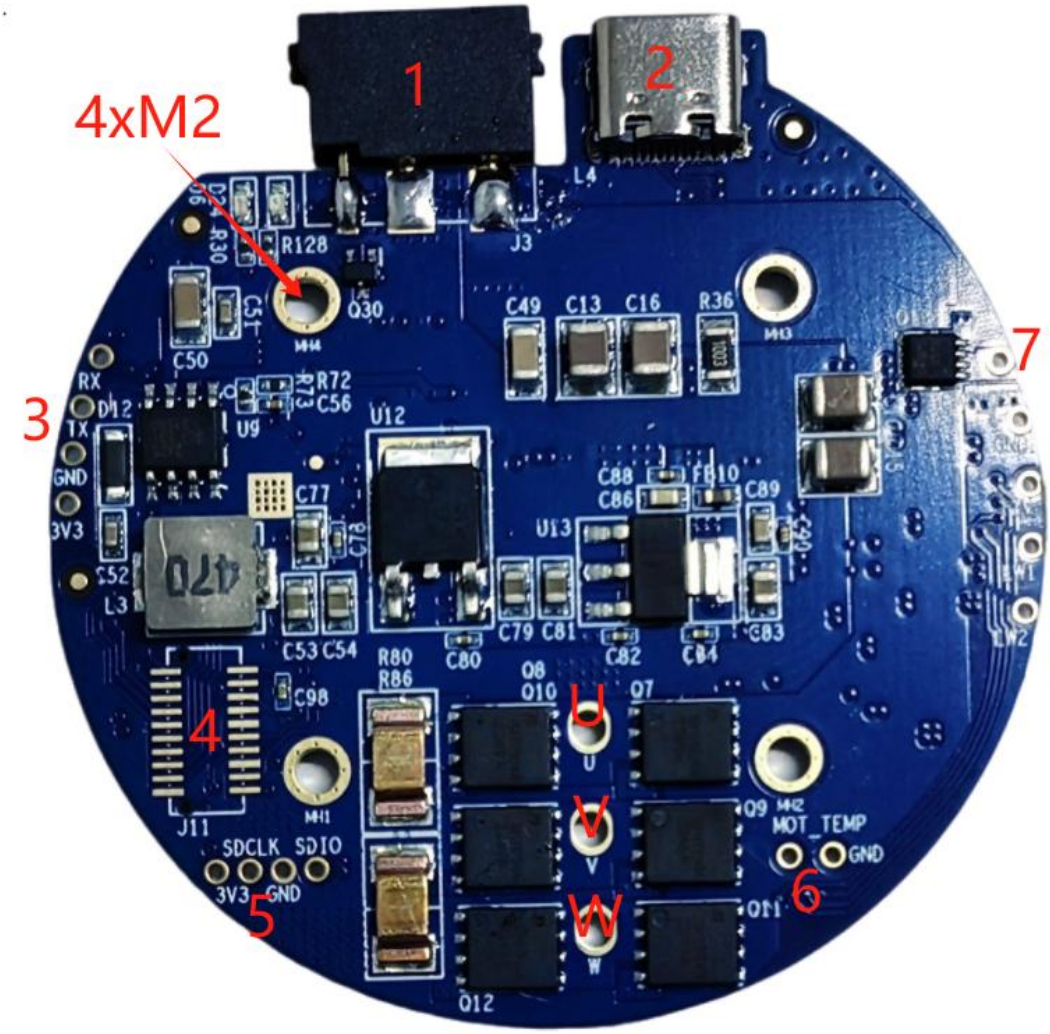
weights	382g±3
polar logarithm	10 pairs
phase (math.)	3-phase
Driver Type	FOC
decelerations比	9.67:1

2 Drive Information

2.1 Appearance and three-dimensional dimensions



2.2 Interface Overview



interface number	define
1	15~60V power supply and CAN communication integrated terminal
2	Type-C debugging interface and upper computer communication interface
3	Second encoder interface (supports I2C and UART)
4	Interface Expansion Slot (Expandable interfaces/protocols for RS485, EtherCAT, model airplane, pulse direction, throttle control, etc.)
5	SWD Debug and Download Interface
6	Motor temperature interface (NTC)
7	Brake/Brake Resistor Interface, 12V Power Supply, Min/Max

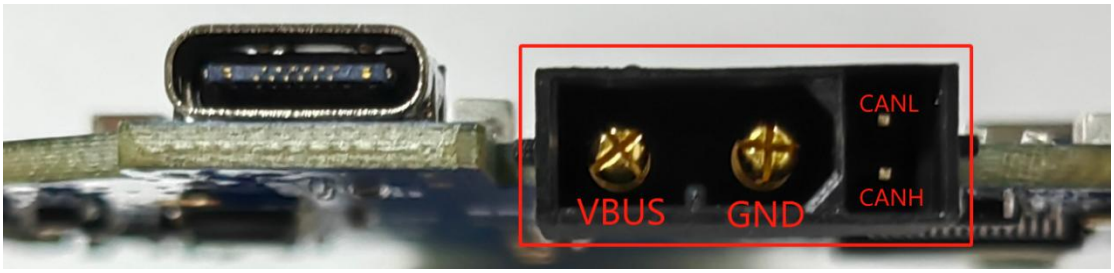
	Limit Switch Interface
U/V/W	Welding holes for three-phase windings
4xM2	mounting hole

2.3 norm

rated voltage	15~48V DC
Minimum/Maximum Voltage	12/72V DC
rated current	6A
Maximum line current	30A
Maximum phase current	90A
Standby power consumption	<10mA
Maximum CAN bus baud rate	1Mbps
Type-C rate	10Mbps
Encoder Resolution	16bit (absolute mono-turn)
Operating Temperature	-20°C to 70°C
Alarm motor temperature	90°C (adjustable)
Alarm drive board temperature	90°C (adjustable)

2.4 Interface Detailed Definition

2.4.1 Power and CAN communication terminals



Board terminal model XT30PB(2+2)-M, wire end model XT30(2+2)-F, brand manufacturer AMASS.

2.4.2 Type-C debugging interface

Type-C uses a standard data cable specification, and commonly used PC or cell phone Type-C data cables are compatible.

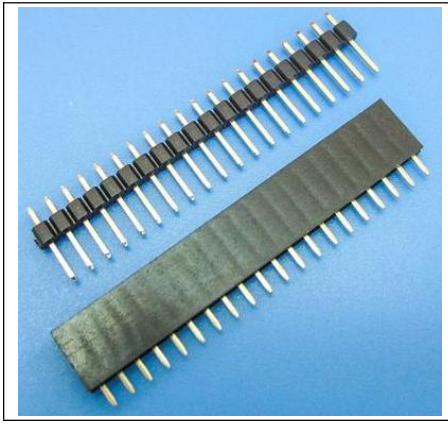
2.4.3 Second encoder interface

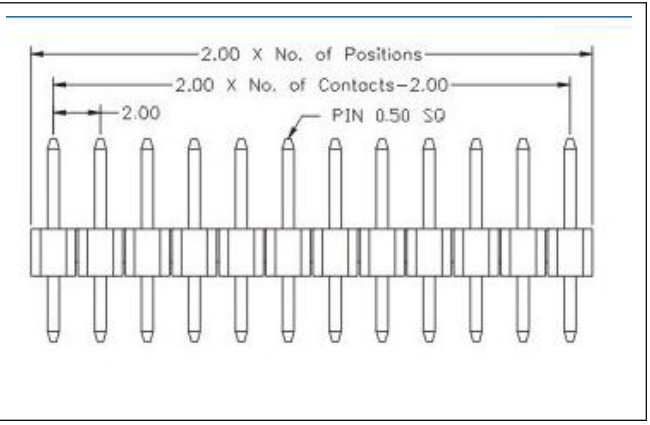
Pin holes spaced 2mm apart, user solderable 2mm inline single row pins, see 2.4.4.

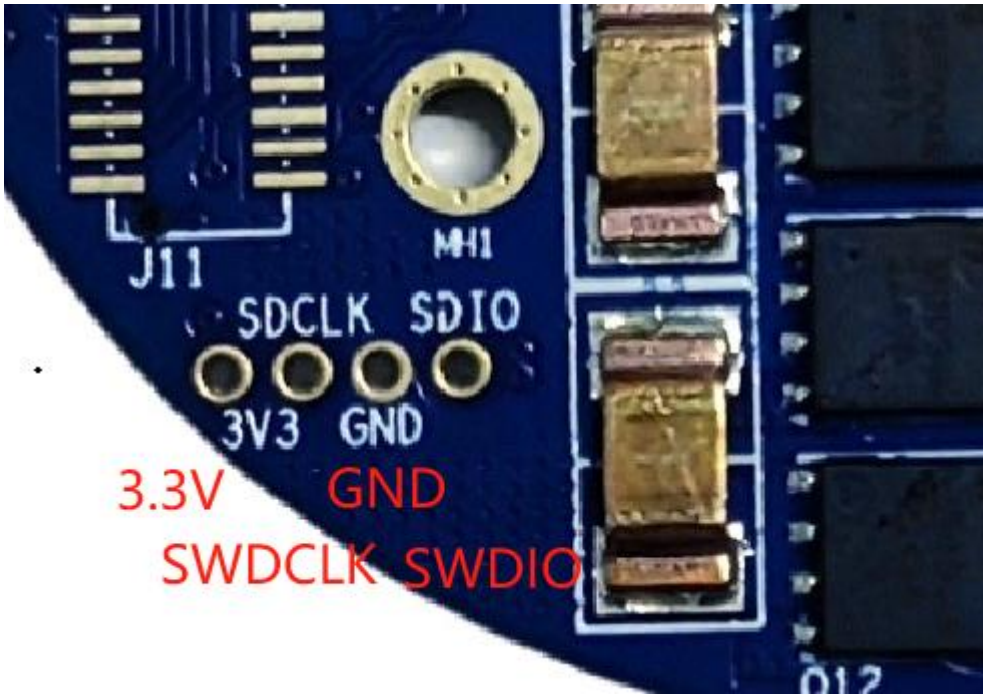
This interface can communicate with the second encoder via USART (TX/RX) or I2C (SCL/SDA).

2.4.4 SWD Debug Interface

With pin holes spaced 2mm apart, users can solder 2mm straight single row pins as shown below:

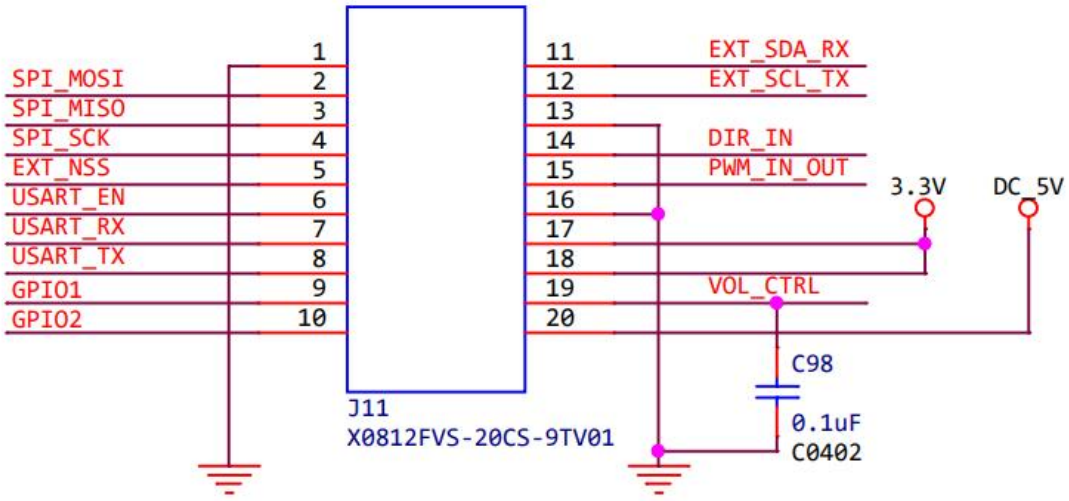






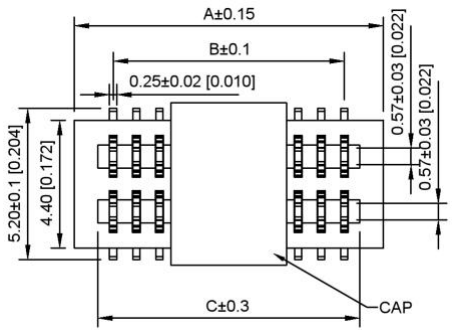
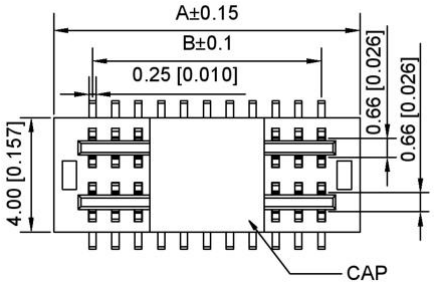
2.4.5 Interface Expansion Slot

This slot is designed in the following way to provide rich expansion interfaces between boards, allowing any expansion board to be developed by a third party:



Third parties can interact with the driver via SPI, USART, I2C, PWM, ADC, GPIO, etc. to realize various extended functions.

The on-board slot model number is X0812FVS-20CS-9TV01 (female chassis), the expansion board slot model number is X0812WVS-20AS-9TV01 (male chassis), and the brand manufacturer is Xingkun.

Female chassis connector X0812FVS-20CS-9TV01	Male Seat X0812WVS-20AS-9TV01
	

2.4.6 Motor temperature interface

The motor has a built-in 10K NTC resistor with two leads soldered to MOT_TEMP and GND with no wiring sequence.

2.4.7 Brake/brake resistor interface

The upper two solder holes in the 5-pin connector shown in the illustration are the holding brake/brake resistor connector, with 2mm spaced pin holes, the user can solder 2mm inline single row pins, please refer to 2.4.4.

When it is the holding gate interface, the driver continuously outputs a current to this interface when power is applied so that the holding gate is open and the motor can run normally. If the driver loses power, this current stops, the holding gate locks, and the motor locks up in the power-off position.

When the interface is a brake resistor, an external brake resistor (or called a drain resistor) can be connected to drain the current through this brake resistor when the

reverse electromotive force is higher than the threshold voltage, preventing the inability to brake in an emergency or damage to the driver by the reverse electromotive force.

2.4.8 Limit Switch Interface

The driver provides two limit switch ports and a 12V power supply for the external limit switches, with 2mm spaced pin holes and user solderable 2mm inline single row pins, see 2.4.4.

LW1 is the minimum position limit switch and LW2 is the maximum position limit switch. It can be externally connected to a two-wire switch or a three-wire NPN switch.

2.5 Main components and specifications

serial number	component	Model/Specification	quantities
1	MCU	N32G455REL7	1

2	driver chip	FD6288Q	1
3	Magnetic Encoder Chip	MA600, 16bit absolute	1
4	MOSFET	JMSH1004NG, 100V/120A	6

3 Tuning instructions

3.1 A Guide to Getting Started

3.1.1 preliminary

To get the motor working you need to:

✓ power supply

See Chapter 1 for power supply voltage requirements, a regulated power supply or battery is recommended. The question that often puzzles users is, how do I choose a power supply? Here are some simple suggestions for reference only:

A few points for choosing a power supply:

◆ **Current Requirement**

Generally, it should be at least greater than 5 A. The exact value depends on the system's power requirements and voltage.

◆ **voltage requirement**

The voltage requirement depends on two factors: the Kv of the motor and the maximum speed required by the system, RPMmax. The maximum value of the required supply voltage can be found in Eq:

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

where 1.25 is an empirical factor that gives the system a safe voltage threshold.

◆ **power requirement**

For the power, it simply depends on the maximum current value Imax at the maximum speed, which can be referred to the formula:

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

✓ Power + Communication Interface Cables

SG6010C with 2+2 power communication socket, please contact after-sales service to recommend suitable 2+2 cable. Please refer to 2.4.1, make sure to connect the power supply positive and negative terminals correctly, otherwise there is a risk of burning the driver, because the driver has no anti-reverse connection capability. Meanwhile, please

pay attention to the definition order of the 2 communication lines, such as the order of CANH/CANL, connecting wrongly will lead to abnormal communication.

warnings
✧ **Be sure to avoid touching the communication bus with your hands to prevent static electricity from damaging the driver's interface chip, especially in dry areas and during dry seasons!** ✧ **Be sure not to unplug the power terminals with electricity!** ✧ **Avoid using a blank switch as a switch for the power supply, there is a risk of destroying the power supply chip on the drive!** ✧ **Do not exceed 72V supply voltage!**

✓ Type-C Data Cable

Early in the tuning process, it is highly recommended to use a USB Type-C cable to test the motor. You can just use the most commonly used Type-C cable for cell phones; do not use a charging-only Type-C cable.

Please note that the Type-C cable does not power the driver, much less turn the motor!

✓ electrify

Please turn off the power, connect the power cable, and then turn on the power, be sure not to unplug with electricity, or use the air switch to control the single positive or negative cable for switching, which will cause excessive power-on current to burn the drive.

The USB Type-C cable can be plugged and unplugged at any time after powering up.

3.1.2 Starting with the OP

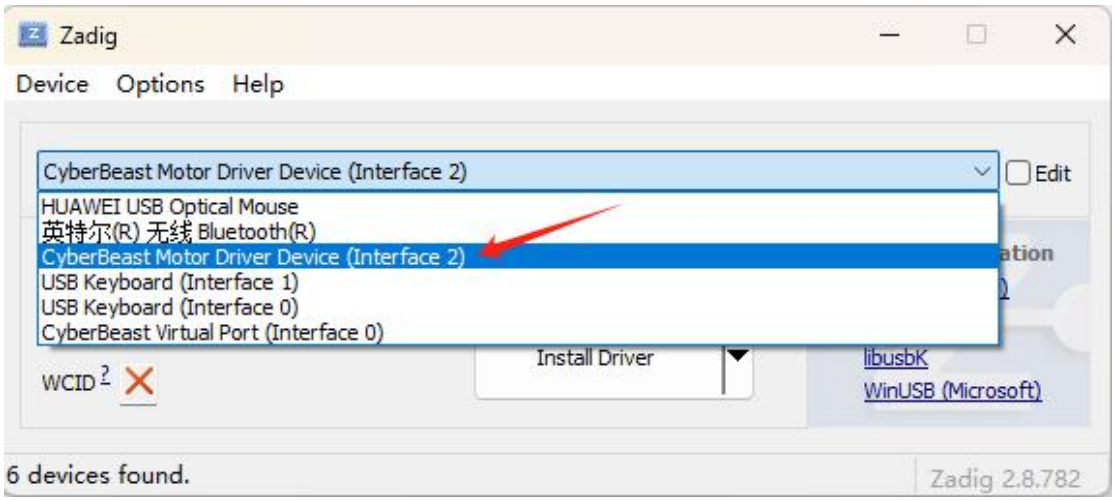
◆ *Motor Wizard*

1) Installation of Motor Wizard (Motor Wizard)

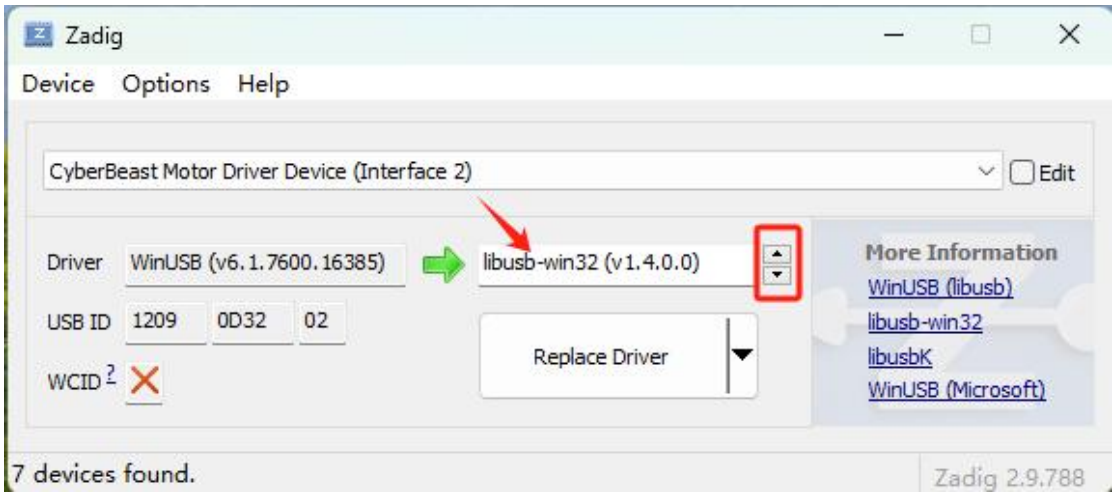
Please download the installation file for Windows (https://bl.cyberbeast.cn/actuator/motorwizard.exe), and follow the default installation procedure.

2) Installation of USB Driver

Go to the website https://zadig.akeo.ie and download the USB driver tool Zadig, use the Type-C cable to connect the driver to the computer, at this time the power light of the driver is on, open Zadig, select "CyberBeast Motor Driver Device (Interface 2":



Select a different USB driver by clicking on the up and down buttons, please select the "libusb" driver version for this interface and click on "Install Driver" or "Replace Click on "Install Driver" or "Replace Driver" to install the driver for this interface:



◆ *odrivetool*

The drive is compatible with odrive (https://github.com/odriverobotics/odrive.git), so it can also be tuned using odrivetool as the upper unit.

Follow the steps below to install odrivetool:

➢ Windows (computer)

3) Installing python

Go to the official python website https://www.python.org to download the latest python installer and follow the instructions. Do not download python versions from third-party websites or the Microsoft Microsoft Store.

4) Installation of visual c++ generator tool

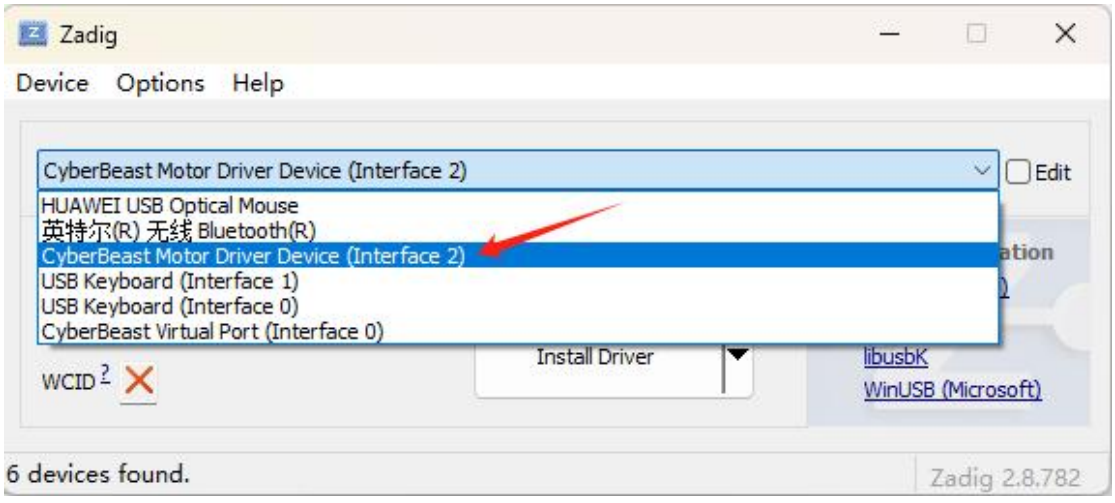
Install the visual c++ generator https://visualstudio.microsoft.com/visual-cpp-build-tools/ and check the "Desktop development with C++" box during the installation process, as shown in the following figure.

5) Install odrivetool

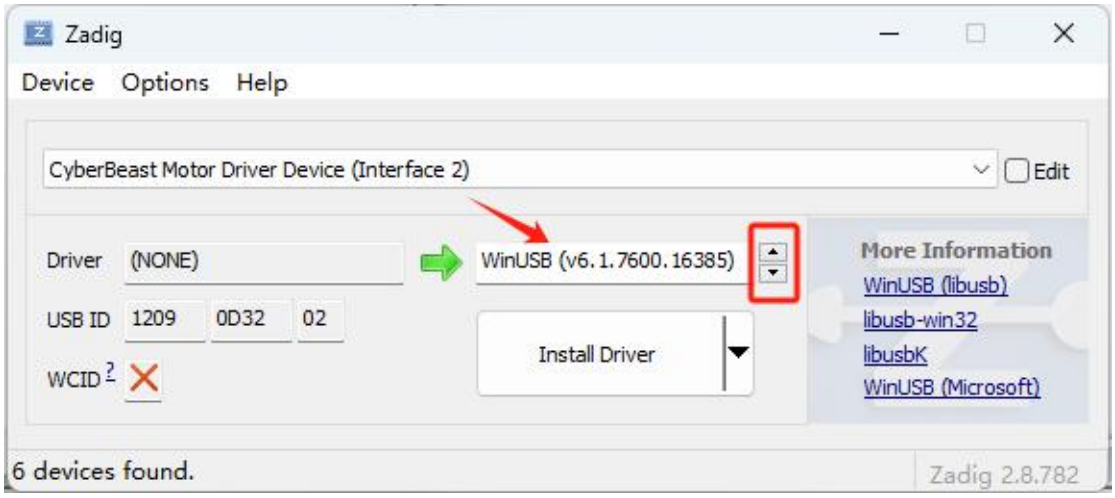
Run Windows PowerShell using administrator, run pip install odrive in it and enter to install. If you get an error in the middle of the installation, please try again. If you get another error, restart your computer and try again.

6) Installation of USB Driver

Go to the website https://zadig.akeo.ie and download the USB driver tool Zadig, use the Type-C cable to connect the driver to the computer, at this time the power light of the driver is on, open Zadig, select "CyberBeast Motor Driver Device (Interface 2":



Select a different USB driver by clicking on the up and down buttons, please select the "WinUSB" driver version for this interface and click on "Install Driver" to install the driver for this interface:



- WSL (Windows Subsystem for Linux)

1) Install python/usb/odrivetool

If the user has WSL2 installed, go to the command line for WSL2 and follow the steps below (assuming the user's WSL installation is Ubuntu):

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

The first line of instructions installs python, the second line of instructions installs the usb driver, and the third line of instructions installs the odrivetool uploader.

2) Connecting drives to WSL

Plug in Windows with a type-C cable, by default Windows will load the driver for this USB port, while WSL will not. If you need to load this USB port into WSL, please refer to Microsoft's document (https://learn.microsoft.com/zh-cn/windows/wsl/connect-usb) to do so.

- Ubuntu

The installation process under Ubuntu is very similar to that under WSL, see the previous subsection.

3.1.3 Reasonable configuration of motor parameters

warnings

It is recommended to read this section carefully as it is essential for successful

operation and to avoid burning out the motor!

After successfully installing the upper unit (motor wizard or odrivetool) according to the previous section, the motor is powered up and the USB Type-C cable is connected:

Motor Wizard

Open the Motor Wizard, if the connection is successful, as shown above left, it will display the motor status, and voltage; if it is not successfully connected, it will not display the normal voltage information.

Click on "Motor Parameters", the hardware version and firmware version of the drive will be displayed at the top of the interface, as shown in the figure above right, the hardware version is 3.8.3 and the firmware version is 0.5.13.

odrivetool

Run odrivetool (type odrivetool and enter) in a shell (Windows PowerShell or Linux Terminal), the picture below shows a successful connection under Windows (green font shows the connection information):

Instruction example: odrv0.axis0.controller.input_vel, odrv0 represents the currently connected motor, by default the first connected motor is called odrv0, the second one is called odrv1, and so on; axis0 represents the first motor connected to the drive, and only one motor is supported to be connected to the drive in the current version. The meaning of this instruction is to query the current speed control target value of the drive.

Operating tips
◆ **If you use the TAB key frequently, you will be prompted for commands, similar to the command prompts under linux, and you can use the up, down, left and right keys to select commands;**
◆ **The up and down keys will display the history of commands**
◆ **When entering a command, the history of similar commands will be prompted, and using the right click will directly complete it**

➢ Setting critical thresholds (limits)

◆ *Motor Wizard*

By clicking on "Motor Parameters" in the Motor Wizard, as shown in the red box below, you can set the following key threshold parameters: Voltage Range, Current Threshold (Maximum Current), Speed Threshold (Maximum Speed), and Calibration Current.

The current threshold is also affected by the temperature factor, see the relevant description in the next subsection (odrivetool).

Speed Threshold usually refers to the limit in speed or position control mode, but it is also possible to turn on the speed limit switch in torque control. This switch can be turned on by clicking on "Drive Parameters" in the motor wizard, as shown in the red box below:

◆ *odrivetool*

- current threshold

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

The above command sets the current threshold to 30 A. Note that this current threshold refers to the Q-axis current, not the supply current. This threshold directly limits the output torque. For the SG6010C, do not set this threshold above 50A!

Other factors affecting current threshold

◆ **Motor temperature**

If motor temperature protection is enabled (odrv0.axis0.motor.motor_thermistor.config.enabled=1), the current temperature of the motor also affects the Q axis current.

◆ **Drive board temperature**

If driver board temperature protection is enabled (odrv0.axis0.motor.fet_thermistor.config.enabled=1), the current temperature of the driver board also affects the Q-axis current.

The effects of the above two temperatures are very similar and can be expressed by the following equation:

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

whereI'_{lim} is the final effective current threshold,I_{lim} is the configured current threshold,T is the current temperature (motor temperature or driver board temperature),T_l is the set lower temperature limit (motor_thermistor.config.temp_limit_lower and fet_thermistor.config.temp_limit_lower), and $T_u$is the set upper temperature limit (motor_thermistor.temp_limit_upper and fet_thermistor.config.temp_limit_upper).

- speed limit

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

The system global speed limit, which is limited to 30turn/s by the above command. Note that by default this speed limit does not work in pure torque mode, but can be

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

enabled by turning on the following switch:

- Calibration Current

This current defaults to 5A, which does not need to be modified by default, but if the user's power supply current is small, this value can be reduced, otherwise a low

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

voltage alarm will occur during calibration.

➢ Setting Key Hardware Parameters

◆ *Motor Wizard*

Click on "Motor Parameters" in the Motor Wizard, the red box below shows some key hardware parameters:

For the meaning of these parameters, please attend the description in the next subsection (odrivetool).

odrivetool

- Maximum discharge/charge current

Discharge current is the forward current supplied by the power supply to the driver and motor, and charge current is the reverse current flowing into the power supply. These two values are related to the power supply, so please set them to an appropriate

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

value to avoid the power supply not being able to discharge causing the voltage to be pulled down or damaged by the reverse electromotive force. However, please note that if you set these two values to a smaller absolute value, it is easy to generate an alarm.

- polar logarithm

The number of pole pairs is the number of poles in the motor rotor divided by 2. The user must set this value correctly for the calibration to be successful, otherwise a

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

calibration alarm will be generated.

- torque constant

The torque constant is the torque produced by the motor divided by the Q-axis

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

current, which is related to the motor Kv value at$Torque_Constant = 8.27/K_v$.

Whether the torque constant is correct or not does not affect the operation of the motor, but it does affect the unit conversion of the value entered by the user when doing torque control. If the user wants to use the unit A instead of Nm for torque control, simply set this value to 1.

- temperature sensor

Both the driver board and the motor have internal NTC temperature sensors which, if enabled, protect the driver and motor by controlling the output current (torque)

```
# Motor temperature protection
odrv0.axis0.motor.motor_thermistor.config.enabled =
odrv0.axis0.motor.motor_thermistor.config.temp_limit_lower =
odrv0.axis0.motor.motor_thermistor.config.temp_limit_upper = 100
# Drive board temperature protection
odrv0.axis0.motor.fet_thermistor.config.enabled =
odrv0.axis0.motor.fet_thermistor.config.temp_limit_lower =
odrv0.axis0.motor.fet_thermistor.config.temp_limit_upper = 100
# Get the temperature
odrv0.axis0.motor.motor_thermistor.temperature #motor temperature
odrv0.axis0.motor.fet_thermistor.temperature #driver board temperature
```

according to temperature.

➢ PID tuning

The following procedure can provide a reference for the user to adjust the PID

```
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
```

parameters:

1. Setting the PID initial value
2. Tune vel_integrator_gain to 0

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

3. Regulates the vel_gain method:

    1) Rotate the motor in speed control mode, if the rotation is not smooth, jerky or vibrating, reduce the vel_gain until the rotation is smooth

    2) Next, increase vel_gain by about 30% each time until there is noticeable jittering

    3) At this point, reduce vel_gain by about 50% to stabilize the

4. Adjust the pos_gain method:

    1) Try to rotate the motor in position mode, if it does not rotate smoothly, pulls or vibrates, decrease pos_gain until it rotates smoothly

    2) Next, increase pos_gain by about 30% each time until there is significant overshooting of the position control (i.e., each time the position control motor goes beyond the target position and then oscillates back to the target position)

    3) Then, gradually reduce pos_gain until the overshoot disappears

5. After the above 4 adjustments, you can set vel_integrator_gain to 0.5*bandwidth*vel_gain, where bandwidth is the system control bandwidth. What is control bandwidth? For example, from the user setting the target position, to the time when the motor really reaches the target position is 10ms, then the control bandwidth is 100Hz, then vel_integrator_gain=0.5*100*vel_gain.

In the above tuning process, it is recommended to use the graphical means in 3.1.7 to view the tuning effect in real time to avoid the error of visual perception. If you use the motor wizard, the PID adjustment effect "what you see is what you get", the following figure is the PID parameter schematic:

3.1.4 Power-Up Calibration

When users use the micromotor for the first time, they need to calibrate the motor as well as the encoder. Before calibrating, please fix the motor or hold it tightly by hand with the output shaft unloaded, the calibration process is as follows:

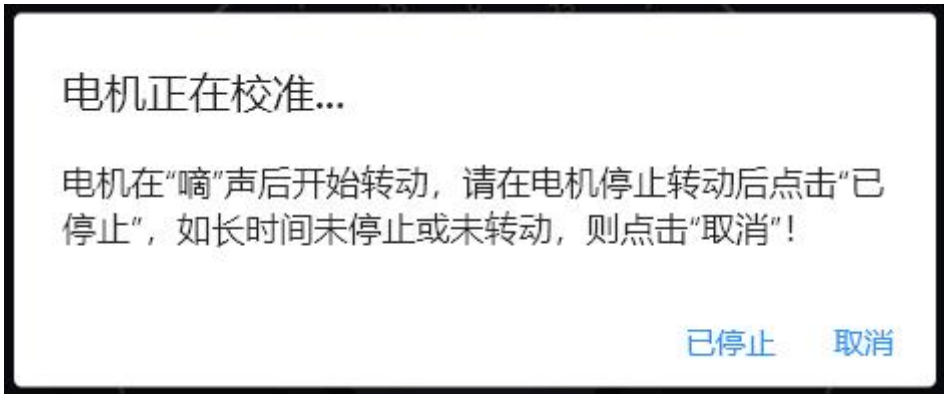
warnings
Before calibration, leave the motor unloaded and secure the motor!

◆ *Motor Wizard*

Open the Motor Wizard and click on "Start Motor". If the motor is not calibrated, the following message will appear:

At this time, click "OK", it will enter the calibration process. Alternatively, click "Motor Parameters" in the main interface, in "Basic Parameters", the calibration status of the motor will be displayed, if it is 未校准 , then click "Calibrate" button on the right side of the status, it will also enter the calibration process. If the status is , click the "Calibration" button on the right side of the status, and you will enter the calibration process as well.

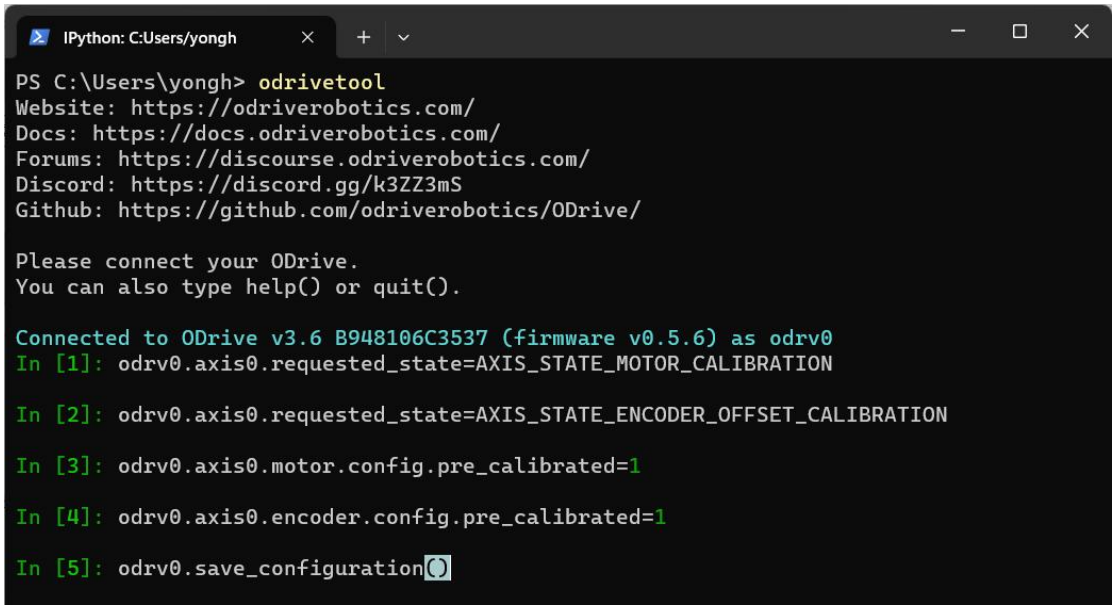
After entering the calibration process, the interface prompts:



At this point, the motor will make a beeping sound and start to rotate. Please observe the rotation of the motor, if it stops after a certain period of time, please click "Stopped" to end the calibration process; if it does not rotate, or will stop after 5 minutes, then click "Cancel".

After finishing calibration, the drive will reboot, please wait for 1~3 seconds. At this time, you can enter "Motor Parameters" again to check if 已校准 is displayed.

- 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
```

The step-by-step explanation is as follows:

- Step 1: Self-recognition of motor parameters

Measure the phase resistance and phase inductance of the motor and a sharp "beep" will be heard. The results of the phase resistance and phase inductance

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

measurements can be viewed using the following commands:

- Step 2: Check the error code

Check the system error code after the first step, if any red error code appears, you need to restart the motor and retry, or report it to the after-sales service.

- Step 3: Encoder Calibration

Calibration of the encoder includes calibration of the encoder's mounting angle to the motor's mechanical angle, as well as calibration of the encoder itself. During this calibration, the motor will slowly rotate forward by one angle and then reverse by one angle. If it stops after only a positive rotation, there is an error, so please check the error code via step four.

- Step 4: Check the error code

After the third step of encoder calibration, check the error code of the system. The error that usually occurs is ERROR_CPR_POLEPAIRS_MISMATCH, which indicates that the CPR of the encoder is set incorrectly, or the pole-pair number of the motor is set

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

incorrectly, so please check/set it by the following command:

- Step 5: Write motor calibration success flag
- Step 6: Write encoder calibration success flag

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

- Step 7: Store calibration results and reboot

3.1.5 Modify ID

The drive ID is unique to the bus and ranges from 0 to 63. The default ID is 0. If the user has more than one motor connected in series to the bus, a different ID needs to be set for each motor. the user can modify the ID in three ways:

- Motor Wizard Modify ID

Open the motor wizard, connect the motor, click "Motor Parameters", in the column of "Communication Parameters", modify the ID, and click "Synchronization" to store it in the drive.

- USB Modify ID

After the user connects to the drive via odrivetool, the ID can be modified using the

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

following command:

- Bus Modification ID

See the Set_Axis_Node_ID command message in 4.1.2.

3.1.6 Storage and backup parameters

After any parameter changes have been made, be sure to store them, otherwise the changes made will expire after a power failure or reboot. The drive will reboot after

```
odrv0.save_configuration()
```

storing the parameters.

Parameter backup:

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

Among them, "d:\test.json" is the path and file name that users can modify freely.

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

The command for parameter recovery is:

3.1.7 Four control modes

After the above preparation and parameterization, you can try to control the motor in different modes of rotation. sG6010C supports position control, speed control, torque control, and motion control modes.

In position control mode, Filtered Position Control, Trapezoidal Position Control, and Circular Position Control are supported;

In the speed control mode, direct speed control (Velocity Control), and ramp speed control (Ramped Velocity Control) are supported;

In the torque control mode, direct torque control (Torque Control), and ramped torque control (Ramped Torque Control) are supported.

Motion control mode is a control mode that combines position, velocity and torque, and is typically used in scenarios that require a strong instantaneous burst of force, such as a robotic knee joint. Some users in the industry also refer to this as MIT control mode, which is derived from the MIT open-source mechanical dog, as it uses this motion control mode to control the motors.

In the subsequent detailed description of each control mode, this document uses the host computer and USB control commands as examples, but the same communication protocols (e.g. CAN) can be used to do the same control with the same logic.

How do I get the motor to turn?

- **Start the motor (enter closed-loop control)**

To turn the motor in all subsequent control operations, it is necessary to put the motor into closed-loop control.
In the motor wizard, click on "Start motor"; in the odrivetool, the command is as follows:
odrv0.axis0.requested_state = 8

- **Stop the motor (enter idle state)**

Users who need to stop the motor, or wish to modify parameters and save them, need to put the motor into an idle state first.
In the motor wizard, click on "stop motor"; in the odrivetool, the command is as follows:
odrv0.axis0.requested_state = 1

◆ *Motor Wizard*

When you open the Motor Wizard, you can see four TAB switches at the bottom, which represent four control modes, namely, "Speed Control", "Torque Control", "Position Control" and "MIT Mode". "MIT mode".

➢ Basic operation of the interface

After starting the motor, the dial is highlighted in color and almost all operations are performed on the dial:

✓ Click on the dial

When clicked on the dial, it indicates control of speed/torque/position to the current dial value. Each click represents a control command. The user can click multiple times in a row to continuously control the motor.

✓ Right click on the dial

When right-clicking on the dial, a pop-up dialog box allows the user to enter precise target speed/torque/position values and enter to perform the control; click outside the dialog box to cancel this control.

✓ Hold and drag on the dial

Holding down the dial and dragging is performing continuous control, which will continuously send speed/torque/position control commands to the motor, targeting the current mouse dial value.

➢ Modify driver parameters

By clicking "Drive Parameters", you can adjust drive parameters such as control mode, speed limit, 3-loop gain (bandwidth), etc. It is recommended that you stop the motor first and then modify and synchronize it. It is recommended to stop the motor first, then modify and synchronize the parameters.

◆ *odrivetool*

➢ Filtered Position Control (FPC)

If the user wishes to generate their own position curves and then send position control commands at a certain frequency, it is recommended that Filtered Position Control be used as this mode will smoothly articulate these commands to be executed together. If trapezoidal curve position control is used in this case, there is a risk that the motor rotation will be stuttering or grainy.

In this mode the filter bandwidth needs to be adjusted according to the frequency of the command being sent, a good rule of thumb is to set the bandwidth to half the

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

frequency of the command (in Hz), e.g., if the command is sent at a frequency of 50 Hz:

Enable filter position control:

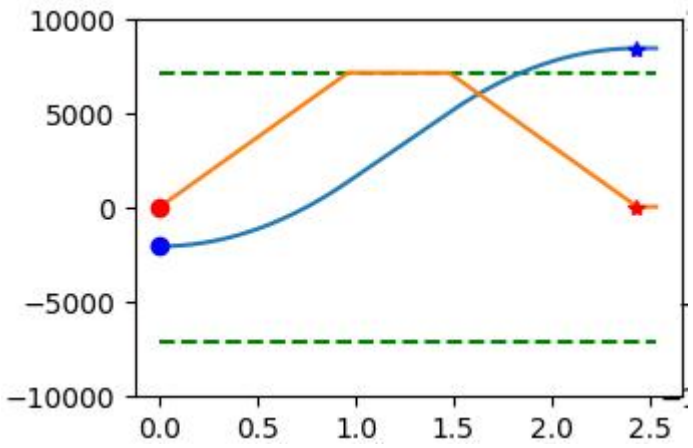
Position control is then performed:

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

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

- Trapezoidal Curve Position Control (Trajectory Control)

This mode allows the user to set the acceleration, glide speed and deceleration to control the motor to rotate smoothly from one position to another. The term "trapezoidal" refers to the trapezoidal appearance of the velocity profile, as shown in the figure below, where the orange color is velocity and the blue color is position:



```
odrv0.axis0.trap_traj.config.vel_limit #Maximum glide speed in turn/s
odrv0.axis0.trap_traj.config.accel_limit # acceleration maximum in turn/s^2
odrv0.axis0.trap_traj.config.decel_limit # deceleration maximum in
turn/s^2
```

Adjustable control parameters:

Note that inertia x acceleration = torque, which defaults to 0. This value improves the system response, but is directly related to the load on the motor. All four of the above values are greater than or equal to 0. Also note that the current threshold and velocity threshold mentioned earlier will still act globally, e.g., the maximum value of the glide velocity mentioned above will act as a global vel_limit if it is set higher than the system level vel_limit.

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

To enable the trapezoidal curve control mode, first:

Position control is then performed:

- Circular Position Control (CPC)

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

This mode is suitable for continuous incremental position control, such as when the robot hub is rotating in one direction for a connected period of time, or when the conveyor track is running all the time, and if the usual position control mode is used, the target position will gradually increase by a large value, and thus there will be an error of inaccurate localization due to floating-point number accuracy problems.

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

Enable:

In this mode, each small step is within a single circle and input_pos is in the range [0, 1). If input_pos increases beyond this range, it is converted to a value within a single lap.

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

If the user wants the single step to be more than a single lap, the following parameter can be set to a number greater than 1:

- Direct Velocity Control (Velocity Control)

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

This mode is the simplest speed control and is enabled as follows:

The target speed is then entered for control:

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

- Ramped Velocity Control (RVC)

Ramp speed control mode is to follow a certain slope to gradually increase the speed to the target value, will be more moderate than the above direct speed control,

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

enabling the following:

Acceleration is controlled by adjusting the slope:

The target speed is then entered for control:

```
odrv0.axis0.controller.config.vel_ramp_rate = 0.5 #Slope in turns/s^2
```

- Direct torque control (Toruqe Control)

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

This is the simplest torque (current) control mode, enabled as follows:

The unit of torque control is Nm, and the unit of current in the driver firmware is A. Therefore, it is also necessary to set the torque constant to allow the driver to convert

```
# The moment constant is approximately equal to8.23/K_v
odrv0.axis0.motor.config.torque_constant = 8.23/104
```

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

Nm to current to drive the motor to output torque on demand.

The target speed is then entered for control:

It should also be noted that if the user wants to limit the maximum speed in torque mode, they can turn on enable_torque_mode_vel_limit and set the vel_limit as:

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

- Ramped Torque Control (RTC)

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

Ramp torque control is very similar to ramp speed control and is enabled as follows:

Adjust the slope as follows:

```
odrv0.axis0.controller.config.torque_ramp_rate = 0.1 # slope in Nm/s
```

- Motion Control (MIT Control)

The motion control mode controls the motor movement to the target position by combining position, speed and torque, and can be expressed by the following equation:

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

WhereT_{target} is the target moment,P_{diff} is the position error,V_{diff} is the velocity error,K_p is the position control gain,K_d is the velocity control gain (or damping coefficient), andT_{ff} is the feedforward moment.

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

The motion control mode enable is as follows:

Adjust the gain:

Motion control is then performed by entering input_pos, input_vel, and

```
odrv0.axis0.controller.input_pos = 5 #unitturns
odrv0.axis0.controller.input_mit_kp = <float> #position gain in Nm/turn
odrv0.axis0.controller.input_mit_kd = <float> #damping factor in
```

input_torque:

Note that the position, speed and torque entered for USB control refer to the rotor side, whereas for MIT control with CAN, the position, speed and torque in the protocol refer to the output shaft side, which is for consistency with the MIT open source protocol!

3.1.8 advanced

1) List of common commands

After successful connection, users can control the motor through commands and get the parameters of motor operation. The following table shows the commonly used commands and the tuning and testing process and instructions:

typology	directives	clarification
basic instruction	dump_errors(odrv0)	Print all error messages
	odrv0.clear_errors()	Clear all error messages
	odrv0.save_configuration()	Be sure to execute this command to store the changes after the parameters have been modified, or after the motor has been automatically recognized as a parameter or calibrated, otherwise all changes will be lost after a power failure.

	odrv0.reboot()	Reboot the drive
	odrv0.vbus_voltage	Getting the supply voltage (V)
	odrv0.ibus	Acquisition of power supply current (A)
	odrv0.hw_version_major	Hardware major version number, the current major version number of SG6010C is 3
	odrv0.hw_version_minor	Hardware minor version number, SG6010C current minor version number is 8
	odrv0.hw_version_variant	Different model numbers under the same hardware configuration, the model number corresponding to SG6010C is 1.
	odrv0.can.config.r120_gpio_num	GPIO number to control the 120R matching resistor switch for the CAN interface
	odrv0.can.config.enable_r120	120R Matched Resistor Switch for Controlling CAN Interface
	odrv0.can.config.baud_rate	Baud rate setting for CAN
Parameter Configuration Instructions	odrv0.config.dc_bus_undervoltage_trip_level	Low Voltage Alarm Threshold (V)
	odrv0.config.dc_bus_overvoltage_trip_level	Overvoltage alarm threshold (V)
	odrv0.config.dc_max_positive_current	Line current maximum (positive) (A)
	odrv0.config.dc_max_negative_current	Line Current Reverse Charge Maximum (Negative) (A)
	odrv0.axis0.motor.config.resistance_calib_max_voltage	The maximum voltage value when the motor parameters are recognized, generally this value is slightly less than half of the power supply voltage, such as 24V power supply, can be set to 10
	odrv0.axis0.motor.config.calibration_current	Maximum current value for motor parameter recognition, this value can be set to 2~5A in general, not too big or too small.
	odrv0.axis0.motor.config.torque_constant	Torque constant of the motor (Nm/A)
	odrv0.axis0.min_endstop.config odrv0.axis0.max_endstop.config	Minimum (LW1)/Maximum (LW2) limit switch configuration: enabled: enabled or not

		gpio_num: the corresponding IO number, please set the minimum limit IO number to 1 and the maximum limit IO number to 2
	odrv0.axis0.encoder.config.index_offset	The user set zero offset, this value is the offset of the user zero point relative to the encoder zero point. After setting this offset value and saving the setting, all user input position control target values are based on this user zero point.
	odrv0.axis0.motor.motor_thermistor.config	Configure the motor temperature sensor: enabled: enabled or not temp_limit_lower: temperature lower 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.temperature	Drive Temperature
Calibration Instructions	odrv0.axis0.requested_state=4	Identify the parameters of the motor, including identifying the phase resistance, phase inductance, and calibrating the three-phase current balance. This process takes 3 to 6 seconds, and the motor will emit a sharp sound. After the sound stops, or if there is no sound after 6 seconds, run dump_errors(odrv0) to check for errors and confirm that there are no errors before proceeding with other operations.
	odrv0.axis0.requested_state=7	Calibrate the encoder. Before performing this operation, make sure that there is no load on the output shaft of the motor and that the motor is secured by hand or other device. After this operation, the motor will rotate forward and reverse for a certain period of time to identify and calibrate

		the encoder. After the motor has stopped, run dump_errors(odrv0) to check for errors and make sure there are no errors before proceeding with other subsequent steps.
	odrv0.axis0.encoder.config.pre_calibrated=1	Write Pre-Calibration Successful means that you do not need to calibrate every time you power up. This parameter can only be written after the above calibration has been successful, otherwise the write will fail.
	odrv0.axis0.controller.config.load_encoder_axis=0	Make sure that the current operating motor is the 0th motor. This operation is only necessary in BETA and is not valid in the production version.
control command	odrv0.axis0.requested_state=1	Stop the motor and enter the idle state
	odrv0.axis0.requested_state=8	Start the motor and enter closed loop
	odrv0.axis0.motor.config.current_lim	Maximum line current (A) for motor operation, exceeding this value will report an overcurrent alarm. **Please note that this value must not be greater than 100.**
	odrv0.axis0.controller.config.vel_limit	Maximum speed of motor operation (turn/s), motor rotor speed exceeding this value will report overspeed alarm.
	odrv0.axis0.controller.config.enable_vel_limit	Speed limit switch, the above vel_limit takes effect when it is True, not when it is False.
	odrv0.axis0.controller.config.control_mode	Control Mode. 0: Voltage control 1: Torque control 2: Speed control 3: Position control
	odrv0.axis0.controller.config.input_mode	Input Mode. Indicates the manner in which the control values entered by the user will control the motor operation: 0: Inactive 1: Direct control 2: Speed Ramp 3: Position filtering 5: Trapezoidal curves

		6: Torque Ramp 9: Motion Control (MIT)
	odrv0.axis0.controller.config.vel_gain	P-value for velocity loop PID control
	odrv0.axis0.controller.config.vel_integrator_gain	I-value for velocity loop PID control
	odrv0.axis0.controller.config.pos_gain	P value for position loop PID control
	odrv0.axis0.controller.input_mit_kp	Motion Control (MIT) Position Gain
	odrv0.axis0.controller.input_mit_kd	Motion Control (MIT) Speed Gain (Damping Factor)
	odrv0.axis0.controller.input_torque	Target for torque control, or torque feedforward for speed control/position control (Nm)
	odrv0.axis0.controller.input_vel	Target for velocity control, or velocity feedforward for position control (turn/s)
	odrv0.axis0.controller.input_pos	Targets for position control (turns)
	odrv0.axis0.encoder.set_linear_count()	Set the absolute position of the encoder, enter a 32-bit integer in parentheses, the absolute value of this integer needs to be less than 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 work when odrv0.axis0.controller.config.input_mode is a trapezoidal curve, adjusting the acceleration and deceleration effects of the position control.
	odrv0.axis0.controller.config.input_filter_bandwidth	Position Filter Bandwidth, this parameter works when odrv0.axis0.controller.config.input_mode is Position Filter and regulates the acceleration and deceleration effect of position control.

2) graphical survey

When tuning a motor, if you need to monitor certain operating parameters in real time, you can utilize python's powerful computational and graphical libraries, as well as the Type-C interface's high-speed throughput capability to output motor parameters in real time.

1. environmental preparation

```
pip install numpy matplotlib
```

Installation of calculation and graphics libraries:

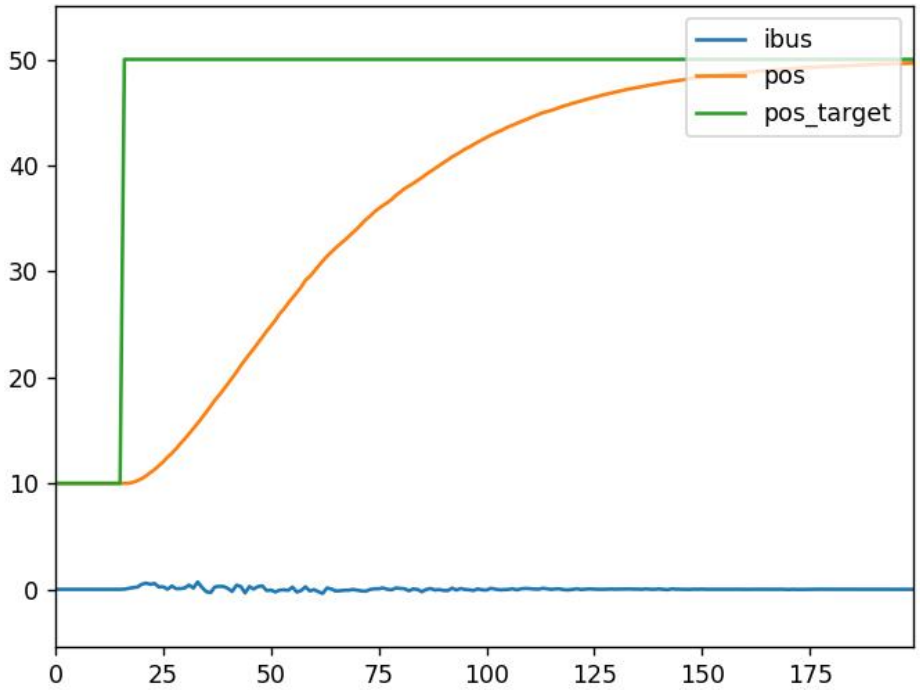
2. Graphical parameter output

In the odrivetool command line interface, bring up the graphical library and read

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

any motor operation metrics such as:

This command will bring up a graphical interface that will output the following three indicators in real time: line current, position, and target position. Next, the motor will be position controlled and the real-time position control curve of the motor will be seen:



3) CAN Matching Resistor Switch

On the driver, a 120 ohm impedance matching resistor has been on-board, which

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

can be turned on or off by the user as needed, with the following example command:

4) User Zero Configuration

By default, the user's position read from the motor, and the input values when doing position control, are based on the zero point of the absolute encoder on the drive as a reference. However, in user scenarios, the zero point of the encoder is most of the time not the user zero point, so the user needs to manually set this zero point offset.

There are generally two means by which the user can locate this zero point, either by means of a limit switch or by manually setting the zero offset, i.e. the offset value of

```
# After the user has first rotated to the desired user zero position, either
manually or via position control:
odrv0.axis0.encoder.config.index_offset =
```

the user's zero point relative to the encoder's zero point:

5) limit switch

The drive supports two limit switches (LW1 and LW2), where LW1 is the minimum

```
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
```

and zero position and LW2 is the maximum position. To use two limit switches, use the following configuration:

When the limit switch is triggered, the system will report MIN_ENDSTOP_PRESSED or MAX_ENDSTOP_PRESSED error, and the host computer can execute the related operation at this time.

Please note that the limit switch function is not supported by hardware version 3.7.

6) Brake Resistors

The user can connect a brake resistor (also called a drain resistor) between the brake resistor connector (EMG and GND) as described in 2.4.7 to shunt the current consumption transient when the bus current backs up into the power supply.

```
odrv0.config.enable_brake_resistor = True
odrv0.config.brake_resistance =  xxx # Setting the external brake resistor resistance value
```

Users can enable it through the following configuration:

How can I get the drive to automatically bleed off? The following two methods can be combined to allow the controller to automatically adjust the drain current:

```
odrv0.config.max_regen_current = xxx # Reverse current max, positive
```

- By adjusting the maximum reverse current

When the reverse current is greater than the above setting, the driver shunts the excess current to the drain resistor.

```
odrv0.config.enable_dc_bus_overvoltage_ramp = True
odrv0.config.dc_bus_overvoltage_ramp_start = xxx # Lower voltage limit of the shunt
```

- By adjusting the reverse electromotive force voltage range

When the voltage due to the counter electromotive force is higher than the above lower limit, the driver starts to shunt to the drain resistor, and when the voltage is greater than or equal to the upper limit of the above voltage, the driver adds all the voltage to the drain resistor in an attempt to shunt. And when the voltage is between the above lower and upper limits, the driver will proportionally add the supply voltage (counter electromotive force voltage) to the drain resistor.

Note that both of these modes can be enabled at the same time, and the driver will attempt to combine the two methods of draining.

3.2 Firmware Update Download

Firmware can be burned via the SWD port (2.4.4) or the Type-C port (2.4.2), both of which are provided below:

3.2.1 Nationwide Download Software

1. USB (DFU) Burning

Please note that the national download software can be burned through the Type-C interface or through the SWD interface (only JLink and DAP are supported), and this section mainly takes the Type-C interface as an example.

First, download the USB driver of Nation Burner (https://www.cyberbeast.cn/filedownload/789489) and install the driver of the corresponding system; then, download Nation Burner (https://cyberbeast.cn/filedownload/766844). Then, download the Nationwide burning software ( ), unzip it to any directory, and run it.

Then, connect the Type-C connector, enter odrivetool, and execute the following

```
odrv0.enter_dfu_mode()
```

command to put the drive in DFU mode:

Finally, use the national burning software to write, as shown in the figure below. Please note that after the burning is finished, please click "Common Operations" and then click "Reset" to restart the drive and connect to the odrivetool for testing.

2. SWD (JLink or DAP) burn-in

Downloading in SWD mode is similar to DFU mode, but requires a connection via the SWD debugging interface (2.4.4) and the selection of the appropriate debugging tool (JLink or DAP) in the figure above.

3.2.2 pyocd

pyocd is the python version of openOCD, which can support STLink, JLink, DAP and other common debugging tools for erasing, burning and resetting. **Please note that the driver must be connected with the SWD interface.** Please refer to 2.4.4 for the wire sequence of SWD. **3.3V power supply is available in the SWD interface, please do not connect the wrong wire sequence to avoid damaging the driver!**

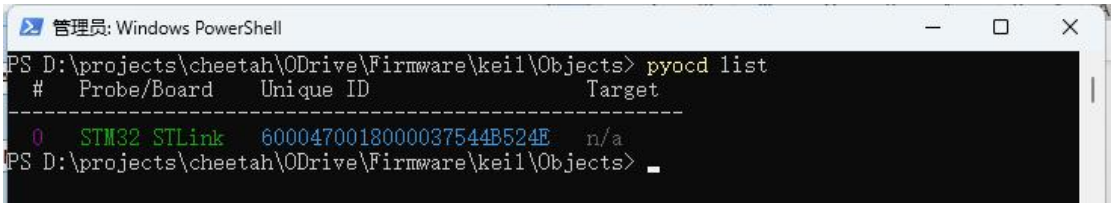
```
pip install pyocd
```

1. mounting

2. burn

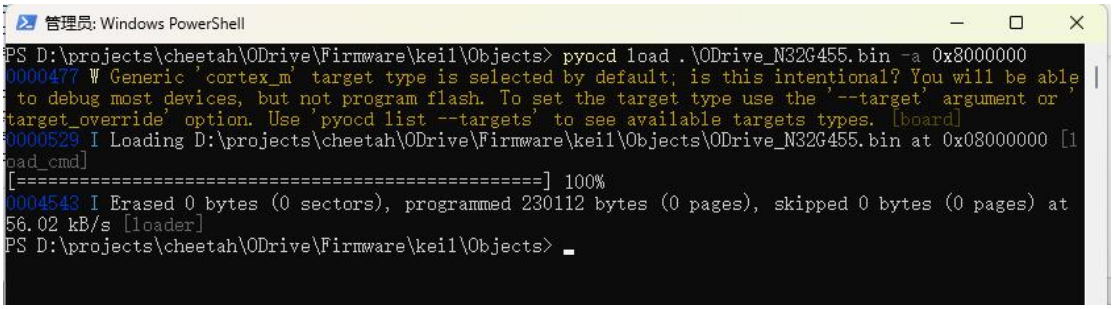
```
pyocd list
```

First, list the connected debugging tools:



Then, execute the following instruction to burn the bin file:

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



4 Integration Notes

4.1 CAN Protocol

The default communication interface is CAN, with a maximum communication rate of 1Mbps (which can be read and set via odrv0.can.config.baud_rate) and a factory default rate of 500Kbps.

4.1.1 PF format

CAN communication uses a standard frame format, data frame, 11-bit ID, 8-byte data, as shown in the table below (MSB on the left, LSB on the right):

data domain	CAN ID (11bits)		Data (8 bytes)
segmentation	Bit10 ~ Bit5	Bit4 ~ Bit0	Byte0 ~ Byte7
descriptive	node_id	cmd_id	communications data

- node_id: represents the unique ID of this motor on the bus, which can be read and set in odrivetool with odrv0.axis0.config.can.node_id.

- cmd_id: command code indicating the message type of the protocol, see the rest of this section.

- Communication data: 8 bytes, the parameters carried in each message are encoded as integers or floats in small endian byte order, where floats are encoded according to the IEEE 754 standard (can be tested on the web site https://www.h-schmidt.net/FloatConverter/IEEE754. html to test the encoding).

Taking the Set_Input_Pos message described in 4.1.2 as an example, assuming that its three parameters are Input_Pos=3.14, Vel_FF=1000 (which means 1rev/s), Torque_FF=5000 (which means 5Nm), and the CMD ID of the Set_Input_Pos message is = 0x00C, assuming the The node_id of the drive is set to 0x05, then:

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

- According to the description of Set_Input_Pos in 4.1.2, it can be seen that Input_Pos is encoded as C3 F5 48 40 (floating point number 3.14 is encoded as a 32-bit number 0x4048f5c3 using the IEEE 754 standard) for the 4 bytes starting from the 0th byte, Vel_FF is encoded as E8 03 for the 2 bytes starting from the 4th byte (1000=0x03E8), Torque_FF 2 bytes at the beginning of the 6th byte, encoded as 88 13 (5000=0x1388), the 8 bytes of communication data are:

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

4.1.2 frame message

The following table lists all available messages:

CMD ID	name (of a thing)	orientations	parameters
0x001	Heartbeat	Motor→ Mainframe	Axis_Error Axis_State Motor_Flag Encoder_Flag Controller_Flag Traj_Done Life
0x002	Estop	Mainframe→ Motor	
0x003	Get_Error	Motor→ Mainframe	Error_Type
0x004	RxSdo	Motor→ Mainframe	

0x005	TxSdo	Motor→ Mainframe	
0x006	Set_Axis_Node_ID	Mainframe→ Motor	Axis_Node_ID
0x007	Set_Axis_State	Mainframe→ Motor	Axis_Requested_State
0x008	Mit_Control	Mainframe→ Motor	
0x009	Get_Encoder_Estimates	Motor→ Mainframe	Pos_Estimate Vel_Estimate
0x00A	Get_Encoder_Count	Motor→ Mainframe	Shadow_Count Count_In_Cpr
0x00B	Set_Controller_Mode	Mainframe→ Motor	Control_Mode Input_Mode
0x00C	Set_Input_Pos	Mainframe→ Motor	Input_Pos Vel_FF Torque_FF
0x00D	Set_Input_Vel	Mainframe→ Motor	Input_Vel Torque_FF
0x00E	Set_Input_Torque	Mainframe→ Motor	Input_Torque
0x00F	Set_Limits	Mainframe→ Motor	Velocity_Limit Current_Limit
0x010	Start_Anticogging	Mainframe→ Motor	
0x011	Set_Traj_Vel_Limit	Mainframe→ Motor	Traj_Vel_Limit
0x012	Set_Traj_Accel_Limits	Mainframe→ Motor	Traj_Accel_Limit Traj_Decel_Limit
0x013	Set_Traj_Inertia	Mainframe→ Motor	Traj_Inertia
0x014	Get_Iq	Motor→ Mainframe	Iq_Setpoint Iq_Measured
0x015	Get_Sensorless_Estimates	Motor→ Mainframe	Pos_Estimate Vel_Estimate
0x016	Reboot	Mainframe→ Motor	
0x017	Get_Bus_Voltage_Current	Motor→ Mainframe	Bus_Voltage Bus_Current
0x018	Clear_Errors	Mainframe→ Motor	
0x019	Set_Linear_Count	Mainframe→ Motor	Linear_Count
0x01A	Set_Pos_Gain	Mainframe→ Motor	Pos_Gain
0x01B	Set_Vel_Gains	Mainframe→ Motor	Vel_Gain Vel_Integrator_Gain
0x01C	Get_Torques	Motor→ Mainframe	Torque_Setpoint Torque
0x01D	Get_Powers	Motor→ Mainframe	Electrical_Power Mechanical_Power
0x01E	Disable_Can	Mainframe→ Motor	
0x01F	Save_Configuration	Mainframe→ Motor	

Detailed descriptions of all the messages are given below:

- Heartbeat

CMD ID: 0x001 (motor→ host)

The heartbeat format for firmware versions less than (and including) 0.5.11 is as follows:

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Error	uint32	Driver exception code (odrv0.axis0.error)
4	Axis_State	uint8	Drive state (odrv0.axis0.current_state)
5	Motor_Flag	uint8	1: Abnormal motor (odrv0.axis0.motor.error is not 0) 0: motor normal (odrv0.axis0.motor.error is 0)
6	Encoder_Flag	uint8	1: Encoder exception (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, i.e., whether the position profile is executed or not bit0: 1: Control exception (odrv0.axis0.controller.error is not 0) 0: control is normal (odrv0.axis0.controller.error is 0)

The heartbeat format for firmware versions greater than (and including) 0.5.12 is as follows:

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Error	uint32	Driver exception code (odrv0.axis0.error)
4	Axis_State	uint8	Drive state (odrv0.axis0.current_state)
5	Flags	uint8	bit0: motor exception bit (whether odrv0.axis0.motor.error is 0) bit1: encoder exception bit (whether odrv0.axis0.encoder.error is 0) bit2: control exception bit (whether odrv0.axis0.controller.error is 0)

			bit7: odrv0.axis0.controller.trajectory_done, i.e., whether the position profile is executed or not
6	Reserved	uint8	reservations
7	Life	uint8	The life value of the cycle message, plus 1 for each heartbeat message, range 0-255, if this life value is not consecutive, it means that the heartbeat message is lost, i.e. the communication is unstable.

The heartbeat format for firmware versions greater than (and including) 0.5.13 is as follows:

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Error	uint32	Driver exception code (odrv0.axis0.error)
4	Axis_State	uint8	Drive state (odrv0.axis0.current_state)
5	Flags	uint8	bit0: motor exception bit (whether odrv0.axis0.motor.error is 0) bit1: encoder exception bit (whether odrv0.axis0.encoder.error is 0) bit2: control exception bit (whether odrv0.axis0.controller.error is 0) bit3: system exception bit (whether odrv0.error is 0) bit7: odrv0.axis0.controller.trajectory_done, i.e., whether the position profile is executed or not
6	Reserved	uint8	reservations
7	Life	uint8	The life value of the cycle message, plus 1 for each heartbeat message, range 0-255, if this life value is not consecutive, it means that the heartbeat message is lost, i.e. the communication is unstable.

- Estop

CMD ID: 0x002 (host→ motor) No parameters no data.

This command causes an emergency motor stop and reports an ESTOP_REQUESTED exception.

- Get_Error

CMD ID: 0x003 (motor→ host)

Input (host→ motor):

start byte	name (of a thing)	typology	clarification
0	Error_Type	uint8	0: Get motor abnormality 1: Get Encoder Exception 3: Acquisition control exception 4: Getting system exceptions

- Output (motor→ mainframe)

start byte	name (of a thing)	typology	odrivetool access
0	error	uint32/uint64	The data and length returned by the different input Error_Type: 0: motor exception (odrv0.axis0.motor.error), uint64, 8 bytes 1: Encoder exception (odrv0.axis0.encoder.error), uint32, 4 bytes 3: control exception (odrv0.axis0.controller.error), uint32, 4 bytes 4: System exception (odrv0.error), uint32, 4 bytes

- RxSdo

CMD ID: 0x004 (host→ motor)

Input:

start byte	name (of a thing)	typology	clarification
0	opcode	uint8	0: Read 1: Write
1	Endpoint_ID	uint16	Please download the JSON file with the IDs corresponding to all parameters and interface functions: ➢ Version 0.5.8 https://bl.cyberbeast.cn/actuator/endpoints_0.5.8.json ➢ Version 0.5.10 https://bl.cyberbeast.cn/actuator/endpoints_0.5.10.json ➢ Version 0.5.11 https://bl.cyberbeast.cn/actuator/endpoints_0.5.11.json ➢ Version 0.5.13

			https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.json
3	reserve	uint8	
4	Value	uint8[4]	It varies according to the Endpoint_ID, see the description in the JSON above. If the Endpoint_ID corresponds to a read-write float value, then the 4 bytes here are the IEEE encoded float value, and the value is written to this float value when opcode=1.

Output (when the above opcode = 0):

start byte	name (of a thing)	typology	clarification
0	opcode	uint8	Fixed to 0
1	Endpoint_ID	uint16	Please download the JSON file with the IDs corresponding to all parameters and interface functions: ➢ Version 0.5.8 https://bl.cyberbeast.cn/actuator/endpoints_0.5.8.json ➢ Version 0.5.10 https://bl.cyberbeast.cn/actuator/endpoints_0.5.10.json ➢ Version 0.5.11 https://bl.cyberbeast.cn/actuator/endpoints_0.5.11.json ➢ Version 0.5.13 https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.json
3	reserve	uint8	
4	Value	uint8[4]	Varies depending on the Endpoint_ID, as described in the JSON above. If the Endpoint_ID corresponds to a readable uint32 value, then the 4 bytes here are little endian byte-ordered uint32.

➢ TxSdo

CMD ID: 0x005 (motor→ host)

Usage is the same as RxSdo with opcode=1.

➢ Set_Axis_Node_ID

CMD ID: 0x006 (host→ motor)

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Node_ID	uint32	odrv0.axis0.config.can.node_id

➢ Set_Axis_State

CMD ID: 0x007 (host→ motor)

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Requested_State	uint32	odrv0.axis0.requested_state

- Mit_Control

CMD ID: 0x008

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

Note that the position, speed and torque entered for USB control refer to the rotor side, whereas for MIT control with CAN, the position, speed and torque in the protocol refer to the output shaft side, which is for consistency with the MIT open source protocol!

- ✓ Mainframe→ Motor

CAN data frame bits	hidden meaning	clarification
BYTE0 BYTE1	**Position**: 16 bits in total, BYTE0 is the high 8 bits, BYTE1 is the low 8 bits Multiturn position of the output shaft in radians (RAD)	The actual **position** is of double type, which needs to be converted to 16-bit int type, and the conversion process is: pos_int = (pos_double + 12.5)*65535 / 25
BYTE2 BYTE3 BYTE4	**Speed**: 12 bits in total, BYTE2 is its high 8 bits and BYTE3[7-4] (high 4 bits) is its low 4 bits. Indicates the angular velocity of the output axis, the unit is RAD/s **KP value** : 12 bits in total, BYTE3[3-0] (low 4 bits) is its high 4 bits and BYTE4 is its low 8 bits.	The actual **speed** is of double type and needs to be converted to 12-bit int type, the conversion process is: vel_int = (vel_double + 65) * 4095 / 130 **The KP value** is actually of double type and needs to be converted to 12-bit int type, the conversion process is: kp_int = kp_double * 4095 / 500
BYTE5 BYTE6 BYTE7	**KD value** : 12 bits in total, BYTE5 is its high 8 bits and BYTE6[7-4] (high 4 bits) is its low 4 bits. **Torque**: 12 bits in total, BYTE6[3-0] (lower 4 bits) is its higher 4 bits and	**The KD value** is actually of double type and needs to be converted to 12-bit int type, the conversion process is: kd_int = kd_double * 4095 / 5

	BYTE7 is its lower 8 bits. The unit is N.m.	The actual **torque** is of double type and needs to be converted to 12-bit int type, the conversion process is: t_int = (t_double + 50)* 4095 / 100 Torque constant in N.m/A

✓ Motor→ Mainframe

CAN data frame bits	hidden meaning	clarification
BYTE0	node id	Drive node id
BYTE1 BYTE2	**Position**: 16 bits in total, BYTE1 is the high 8 bits, BYTE2 is the low 8 bits Multiturn position of the output shaft in radians (RAD)	The actual **position** is of type double, which needs to be converted from 16-bit int. The conversion process is: pos_double = pos_int * 25 / 65535 - 12.5
BYTE3 BYTE4 BYTE5	**Speed**: 12 bits in total, BYTE3 is its high 8 bits and BYTE4[7-4] (high 4 bits) is its low 4 bits. Indicates the angular velocity of the output axis, the unit is RAD/s **Torque**: 12 bits in total, BYTE4[3-0] (lower 4 bits) is its higher 4 bits and BYTE5 is its lower 8 bits. The unit is N.m.	The actual **speed** is of double type and needs to be converted from 12-bit int type, the conversion process is: vel_double = vel_int * 130 / 4095 - 65 The actual **torque** is of double type and needs to be converted from 12-bit int type, the conversion process is: t_double = t_int * 100 / 4095 - 50 Torque constant in N.m/A

➢ Get_Encoder_Estimates

CMD ID: 0x009 (motor→ host)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Pos_Estimate	float32	rev	odrv0.axis0.encoder.pos_estimate
4	Vel_Estimate	float32	rev/s	odrv0.axis0.encoder.vel_estimate

➢ Get_Encoder_Count

CMD ID: 0x00A (motor→ host)

start byte	name (of a thing)	typology	odrivetool access
0	Shadow_Count	int32	odrv0.axis0.encoder.shadow_count
4	Count_In_Cpr	int32	odrv0.axis0.encoder.count_in_cpr

- Set_Controller_Mode

CMD ID: 0x00B (host→ motor)

start byte	name (of a thing)	typology	odrivetool access
0	Control_Mode	uint32	odrv0.axis0.controller.config.control_mode
4	Input_Mode	uint32	odrv0.axis0.controller.config.input_mode

- Set_Input_Pos

CMD ID: 0x00C (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Input_Pos	float32	rev	odrv0.axis0.controller.input_pos
4	Vel_FF	int16	0.001rev/s	odrv0.axis0.controller.input_vel
6	Torque_FF	int16	0.001Nm	odrv0.axis0.controller.input_torque

- Set_Input_Vel

CMD ID: 0x00D (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Input_Vel	float32	rev/s	odrv0.axis0.controller.input_vel
4	Torque_FF	float32	Nm	odrv0.axis0.controller.input_torque

- Set_Input_Torque

CMD ID: 0x00E (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Input_Torque	float32	Nm	odrv0.axis0.controller.input_torque

- Set_Limits

CMD ID: 0x00F (host→ motor)

start byte	name (of a thing)	typology	unit (of measur	odrivetool access

			e)	
0	Velocity_Limit	float32	rev/s	odrv0.axis0.controller.config.vel_limit
4	Current_Limit	float32	A	odrv0.axis0.motor.config.current_lim

- Start_Anticogging

CMD ID: 0x010 (host→ motor)

Perform moment ripple calibration.

- Set_Traj_Vel_Limit

CMD ID: 0x011 (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Traj_Vel_Limit	float32	rev/s	odrv0.axis0.trap_traj.config.vel_limit

- Set_Traj_Accel_Limits

CMD ID: 0x012 (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Traj_Accel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.accel_limit
4	Traj_Decel_Limit	float32	rev/s^2	odrv0.axis0.trap_traj.config.decel_limit

- Set_Traj_Inertia

CMD ID: 0x013 (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Traj_Inertia	float32	Nm/(rev/s^2)	odrv0.axis0.controller.config.inertia

- Get_Iq

CMD ID: 0x014 (motor→ host)

start byte	name (of a thing)	typology	unit (of	odrivetool access

			measure)	
0	Iq_Setpoint	float32	A	odrv0.axis0.motor.current_control.Idq_setpoint
4	Iq_Measured	float32	A	odrv0.axis0.motor.current_control.Iq_measured

- Get_Sensorless_Estimates

CMD ID: 0x015 (motor→ host)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Pos_Estimate	float32	rev	odrv0.axis0.sensorless_estimator.pll_pos
4	Vel_Estimate	float32	rev/s	odrv0.axis0.sensorless_estimator.vel_estimate

- Reboot

CMD ID: 0x016 (host→ motor)

- Get_Bus_Voltage_Current

CMD ID: 0x017 (motor→ host)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Bus_Voltage	float32	V	odrv0.vbus_voltage
4	Bus_Current	float32	A	odrv0.ibus

- Clear_Errors

CMD ID: 0x018 (host→ motor)

Clear all errors and exceptions.

- Set_Linear_Count

CMD ID: 0x019 (host→ motor)

Sets the absolute encoder position.

start byte	name (of a thing)	typology	odrivetool access

0	Linear_Count	int32	odrv0.axis0.encoder.set_linear_count()

- Set_Pos_Gain

CMD ID: 0x01A (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Pos_Gain	float32	(rev/s)/rev	odrv0.axis0.controller.config.pos_gain

- Set_Vel_Gains

CMD ID: 0x01B (host→ motor)

start byte	name (of a thing)	typology	unit (of measure)	odrivetool access
0	Vel_Gain	float32	Nm/(rev/s)	odrv0.axis0.controller.config.vel_gain
4	Vel_Integrator_Gain	float32	Nm/rev	odrv0.axis0.controller.config.vel_integrator_gain

- Get_Torques

CMD ID: 0x01C (motor→ host)

start byte	name (of a thing)	typology	odrivetool access
0	Torque_Setpoint	float32	odrv0.axis0.controller.torque_setpoint
4	Torque	float32	None. Indicates the current torque value.

- Get_Powers

CMD ID: 0x01D (motor→ host)

start byte	name (of a thing)	typology	odrivetool access
0	Electrical_Power	float32	odrv0.axis0.controller.electrical_power
4	Mechanical_Power	float32	odrv0.axis0.controller.mechanical_power

- Disable_Can

CMD ID: 0x01E (host→ motor)

Disable CAN and reboot the drive.

- Save_Configuration

CMD ID: 0x01F (host→ motor)

Stores the current configuration, takes effect and reboots.

4.1.3 CAN protocol in practice

1) Hands-on: power-up calibration

The sequence for sending a CAN message is as follows:

CAN ID	Frame Type	frame data	clarification
0x007	data frame	04 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 4 Calibration of motors
0x007	data frame	07 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 7 Calibration of the encoder

2) Practical: speed control

The sequence for sending a CAN message is as follows:

CAN ID	Frame Type	frame data	clarification
0x00B	data frame	02 00 00 00 02 00 00 00	Message: Set_Controller_Mode Parameters: 2/2 Set control mode to speed control, input mode to speed ramp
0x007	data frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 8 Entering closed-loop control
0x0D	data frame	00 00 20 41 00 00 00 00	Message: Set_Input_Vel Parameter: 10/0 Set the target speed and moment feedforward, where the target speed is 10 (floating point number: 0x41200000) and the moment feedforward is 0

			(floating point number: 0x00000000)

3) Practical: position control

The sequence for sending a CAN message is as follows:

CAN ID	Frame Type	frame data	clarification
0x00B	data frame	03 00 00 00 03 00 00 00	Message: Set_Controller_Mode Parameters: 3/3 Set control mode to position control and input mode to position filtering
0x007	data frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 8 Entering closed-loop control
0x0C	data frame	CD CC 0C 40 00 00 00 00	Message: Set_Input_Pos Parameter: 2.2/0/0 Set the target position, velocity feedforward and moment feedforward, where the target position is 2.2 (floating point number: 0x400CCCCD), and the moment feedforward and velocity feedforward are 0

4.1.4 CANOpen Compatibility

Interoperability with CANOpen is possible if the node ID is properly assigned. The following table lists the valid node ID combinations for CANopen and this protocol:

CANOpen node IDs	This 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 news

The user can configure the motor to send periodic messages to the host computer without the host computer sending request messages to the motor. Periodic messages

can be turned on/off through a series of configurations under odrv0.axis0.config.can (a value of 0 means off, and any other value means the cycle time in ms), as shown in the table below:

messages	odrivetool configuration	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 cycle messages are turned on at the factory, so when the user monitors the CAN bus, he or she will see two messages broadcast at set cycles. The

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

user can turn them off with the following command:

For details on individual messages, see 4.1.2.

4.2 Python SDK

Please first install odrivetool (pip install -upgrade odrive) by referring to the steps in section 3.1. See 3.1.8 for python development with all the commands described in that subsection.

Here are three examples:

```
import odrive
import time

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

4.2.1 Hands-on: power-up calibration

```
import odrive
import time

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

4.2.2 Practical: speed control

```
import odrive

odrv0 = odrive.find_any()
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.POSI
TION_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.POS_FILT
ER
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTR
OL
```

4.2.3 Practical: position control

4.2.4 Practical: data collection

Users in the R & D integration process, often need to collect the motor running data, such as collecting voltage and current changes, position and speed changes, etc. Python SDK integrates a powerful data capture capabilities, you can use a simple script to realize

the massive running data capture, so that the R & D and integration has become more simple.

The following code adds real-time position and current data grabbing to the

```
import odrive
import numpy as np

odrv0 = odrive.find_any()
cap =
odrive.utils.BulkCapture(lambda:[odrv0.axis0.motor.current_control.Iq_mea
sured,odrv0.axis0.encoder.pos_estimate],data_rate=500 .duration=2.5)
odrv0.axis0.controller.config.control_mode=odrive.utils.ControlMode.POSI
TION_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.POS_FILT
ER
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTR
OL
odrv0.axis0.controller.input_pos=10
```

previous position control example and saves the data as a csv file.

In the statement where BulkCapture is performed, data_rate represents the sampling frequency in hz and duration represents the sampling time in seconds, where the lambda expression can be inserted with any mathematical operation to facilitate data analysis.

4.3 Arduino SDK

Users can use Arduino to control the motor via CAN bus, the underlying protocol is described in 4.1. Compatible hardware/libraries:

✓ Arduino with built-in CAN interface, such as Arduino UNO R4 Minima, Arduino UNO R4 WIFI, etc.

✓ Teensy development boards with built-in CAN interface can be accessed using the adapted FlexCAN_T4 library (Teensy 4.0 and Teensy 4.1)

✓ Other Arduino-compatible boards can be accessed using MCP2515-based CAN expansion boards

Here is an example showing how to configure the motor to respond to the Arduino's position control commands:

- Configuration motor

In addition to the base configuration of 3.1, configure the control to have a control bandwidth of 20 rad/s (the Arduino Uno is limited in its sending speed, so the control bandwidth doesn't have to be too high, and you can increase this bandwidth value if you use a faster Arduino):

- Configuring CAN

Configure CAN as described below:

- Installing the ODriveArduino library

Follow the steps below to install the OdriveArduino library (assuming the user has installed the Arduino IDE):

1) Open the Arduino IDE
2) Sketch -> Include Library -> Manage Libraries
3) Search by typing "ODriveArduino".
4) Click on the searched ODriveArduino libraries to install them.

- 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...") 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...") 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
    // 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 t = 0.001 * millis(); float t = 0.002 * millis()

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

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

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

4.4 ROS SDK

The following steps have been tested and verified on Ubuntu 23.04 and ROS2 Iron, but are not supported on MAC and Windows platforms, and are not verified on other ROS2 versions, and need to be corrected before they can be used.

4.4.1 Install the odrive_can package

1. Create a new ROS2 workspace (see https://docs.ros.org/en/iron/index.html)
2. Use git clone https://github.com/odriverobotics/odrive_can to download the code to the src directory in the above workspace directory

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

3. Go to the root directory of the workspace in terminal and run it:
4. Environment preparation before running:
5. Run routine node:

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

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

4.4.2 Calling services and viewing messages

Assuming the above odrive_can_node node runs in the namespace odrive_axis0 (which can be set up in . /launch/example_launch.yaml). Once the node in 4.4.1 has been run, it is possible to view the published topic messages such as:

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

and call the open service interface, e.g. the following call can start the motor

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

calibration:

5 FAQs and exception codes (to be updated)

5.1 Frequently Asked Questions (FAQ)

5.2 exception code

error category	error code	odrivetool shows	descriptive
system anomaly	0x00000002	DC_BUS_UNDER_VOLTAGE	Power supply voltage too low
	0x00000004	DC_BUS_OVER_VOLTAGE	Power supply voltage too high
	0x00000008	DC_BUS_OVER_REGEN_CURRENT	Power reverse (charging) current too high
	0x00000010	DC_BUS_OVER_CURRENT	Power supply forward (discharge) current too high
driver anomaly (computing)	0x00000001	INVALID_STATE	Drive status error
	0x00000040	MOTOR_FAILED	Motor Abnormal
	0x00000100	ENCODER_FAILED	Encoder abnormality
	0x00000200	CONTROLLER_FAILED	Controller Exception
	0x00001000	MIN_ENDSTOP_PRESSED	low limit trigger
	0x00002000	MAX_ENDSTOP_PRESSED	High Limit Trigger
	0x00004000	ESTOP_REQUESTED	emergency stop
	0x00020000	HOMING_WITHOUT_ENDSTOP	Return to zero but no limit switch
	0x00080000	UNKNOWN_POSITION	No location information
Motor Abnormal	0x00000001	PHASE_RESISTANCE_OUT_OF_RANGE	Phase-to-phase resistance out of normal range
	0x00000002	PHASE_INDUCTANCE_OUT_OF_RANGE	Interphase inductance out of normal range
	0x00000010	control_deadline_missed	FOC frequency is too high
	0x00000080	MODULATION_MAGNITUDE	SVM modulation anomaly
	0x00000400	CURRENT_SENSE_SATURATION	Phase current saturation
	0x00001000	CURRENT_LIMIT_VIOLATION	Excessive motor current

	0x00020000	MOTOR_THERMISTOR_OVER_TEMP	High motor temperature
	0x00040000	FET_THERMISTOR_OVER_TEMP	Drive temperature too high
	0x00080000	timer_update_missed	FOC not processed in a timely manner
	0x00100000	current_measurement_unavailable	Loss of phase current samples
	0x00200000	CONTROLLER_FAILED	Control anomalies
	0x00400000	I_BUS_OUT_OF_RANGE	Busbar current overrun
	0x00800000	BRAKE_RESISTOR_DISARMED	Brake resistor drive abnormality
	0x01000000	SYSTEM_LEVEL	System level exception
	0x02000000	BAD_TIMING	Phase current sampling is not timely
	0x04000000	UNKNOWN_PHASE_ESTIMATE	Motor position unknown
	0x08000000	UNKNOWN_PHASE_VEL	Motor speed unknown
	0x10000000	UNKNOWN_TORQUE	Torque unknown
	0x20000000	UNKNOWN_CURRENT_COMMAND	Torque control unknown
	0x40000000	UNKNOWN_CURRENT_MEASUREMENT	Current sampling value unknown
	0x80000000	UNKNOWN_VBUS_VOLTAGE	Voltage sampling value unknown
	0x100000000	UNKNOWN_VOLTAGE_COMMAND	Voltage control unknown
	0x200000000	UNKNOWN_GAINS	Current loop gain unknown
	0x400000000	CONTROLLER_INITIALIZING	Controller initialization exception
	0x800000000	UNBALANCED_PHASES	Three-phase unbalance
Control anomalies	0x00000001	OVERSPEED	Excessive speed
	0x00000002	INVALID_INPUT_MODE	Incorrect control input mode
	0x00000004	UNSTABLE_GAIN	Phase-locked loop gain instability
	0x00000020	INVALID_ESTIMATE	Position/speed instability
		SPINOUT_DETECTED	Mismatch between mechanical and

			electrical power (incorrect encoder calibration, or unstable magnets)
Encoder abnormality	0x00000001	UNSTABLE_GAIN	Encoder bandwidth too high
	0x00000002	CPR_POLEPAIRE_MISMATCH	CPR and polar log mismatch
	0x00000004	NO_RESPONSE	Encoder not responding
	0x00000400	SEC_ENC_COM_FAIL	Second encoder communication error