

GIM6010-8 Micromotor Instruction Manual

catalogs

Release Notes	2
Cautions	3
Legal Notices	3
After-sales policy	3
1 Motor specification parameters	5
1.1 Drawings and Dimensions.....	5
1.2 Electrical Characteristics.....	6
1.3 Mechanical properties.....	7
2 Drive Information	8
2.1 Appearance and 3D dimensions.....	8
2.2 Interface Overview.....	9
2.3 Specification.....	9
2.4 Interface Detailed Definition.....	10
2.5 Main components and specifications.....	14
3 Tuning Instructions	15
3.1 Handyman's Guide.....	15
3.2 Firmware Update Download.....	33
4 Communication Protocols and Examples	35
4.1 CAN Protocol.....	35
4.2 PYTHON SDK.....	46
4.3 ARDUINO SDK.....	49
4.4 ROS SDK.....	56
5 FAQs and Exception Codes (to be updated)	57
5.1 Frequently Asked Questions (FAQ).....	57
5.2 Exception Code.....	57

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 switching description for USB and CAN compatibility
0.5	2023.12.29	Increase the upper computer tuning test real-world cases
0.6	2024.1.2	Add 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	Addition of CAN commands for modifying parameters and calling 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 the second encoder role of the The initial position range of the rotor is described.
1.0	2024.3.26	Adding motor temperature protection instructions
1.1	2024.5.20	Adding a tuning guide
1.2	2024.5.24	Fixed bug in CAN MIT protocol

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 make sure 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 violates the contents of the manual to use this product, caused by any property damage, personal injury accidents, the company does not assume any responsibility. As this product consists of many parts and components, do not allow children to contact this product to avoid accidents. In order to prolong the service life of the product, please do not use the product in a high temperature and high pressure environment. This manual has been printed to include as much functionality and instructions as possible. However, due to the continuous improvement of product functions, design changes, etc., there may still be inconsistencies with the product purchased by the user.

This manual may deviate from the actual product in terms of color, appearance, etc. Please refer to the actual product. The Company 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 should be obtained by contacting Technical Support. All illustrations are for functional illustration purposes only.

After-sales policy

After-sales service of this product is strictly based on the "Law of the People's Republic of China on the Protection of Consumer Rights and Interests" and "Law of the People's Republic of China on Product Quality" to implement the after-sales service.

1. Warranty Regulations for Informal Warranty

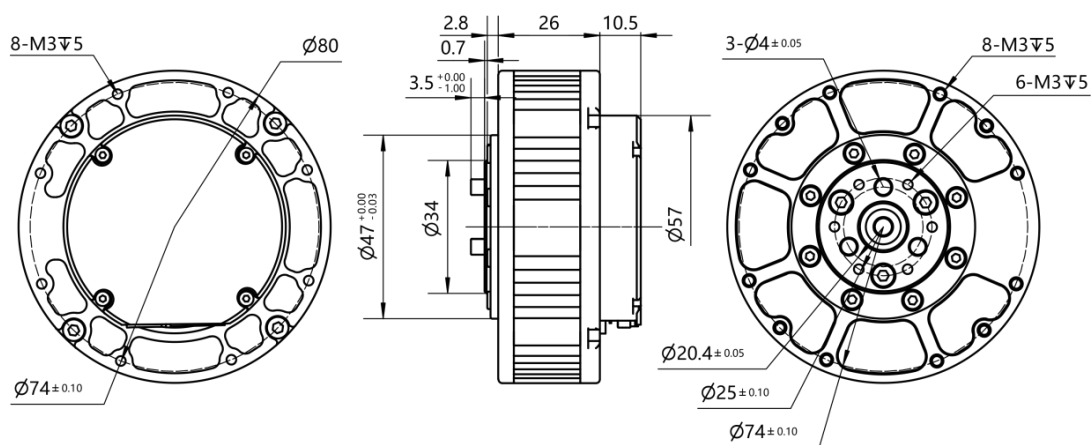
The following are not covered by the warranty:

- 1) The warranty period limited by the terms of the Super[®] Warranty.
- 2) Damage or destruction of the product caused by incorrect use without following the instructions.
- 3) Damage or destruction 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 working conditions, including but not limited to drops, impacts, liquid immersion, severe impacts, etc.
- 6) Damage caused by acts of God (such as floods, fire, lightning, earthquakes, etc.) or force majeure.
- 7) Damage caused by use beyond 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.

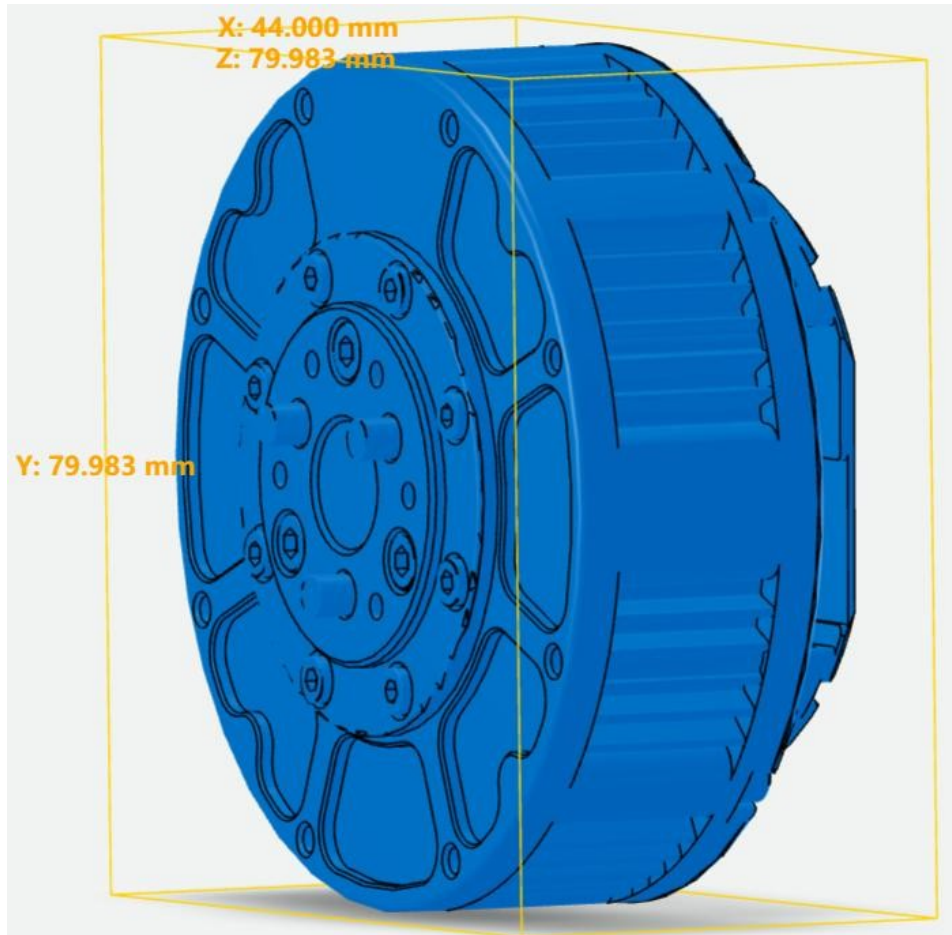
If [®] the above situation occurs, the user will be required to pay the fee at his/her own expense.

1 Motor specification parameters

1.1 Drawings and Dimensions



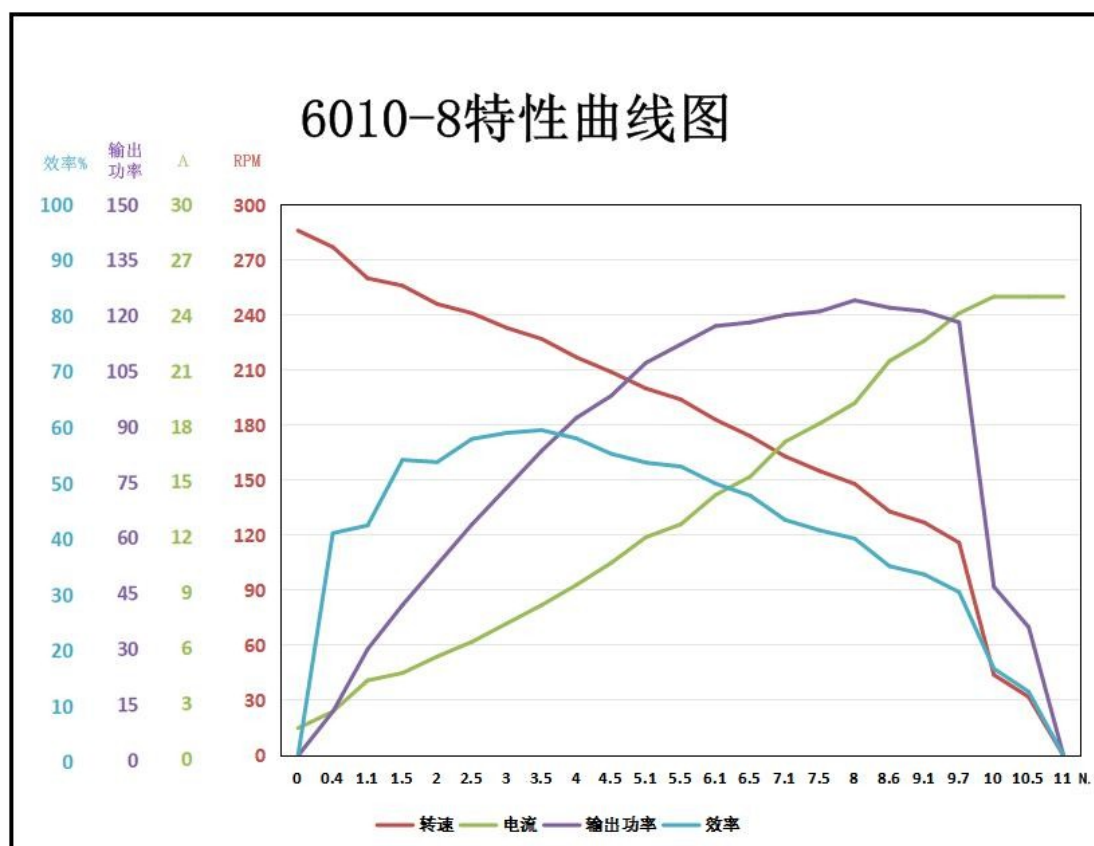
未注公差分段					比例	1:1	6010-8安装尺寸图
<=6	>6<=30	>30<=120	>120<=400	>400<=2000	重量kg	无	
± 0.1	± 0.2	± 0.3	± 0.5	± 1.0			



1.2 Electrical Characteristics

Rated speed	120rpm ±10%
Maximum speed	420rpm ±10%
Rated torque	5N.m
Blocking torque	11N.m
rated current	10.5A
Plugging current	25A
No-load current	0.4A
Insulation resistance/stator winding	DC 500VAC, 100M Ohms
High pressure resistance/stator and housing	600 VAC, 1s, 2mA
motor reverse electromotive force	0.054~0.057Vrms/rpm
phase resistance	0.48Ω ±10%
phase inductance	368μH ±10%
RPM constant	12.3rpm/v
torque constant	0.47N.M/A

The characteristic curve is shown below:

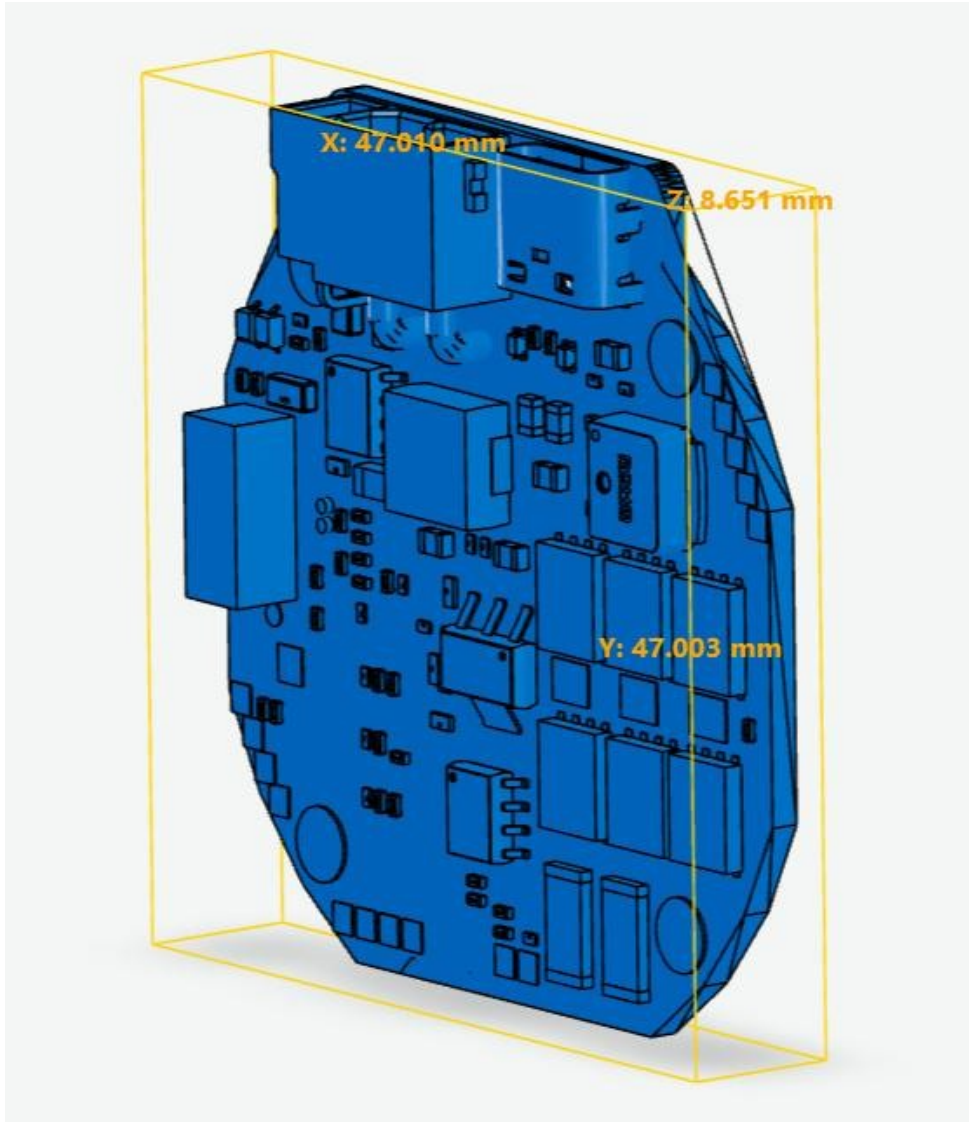


1.3 mechanical property

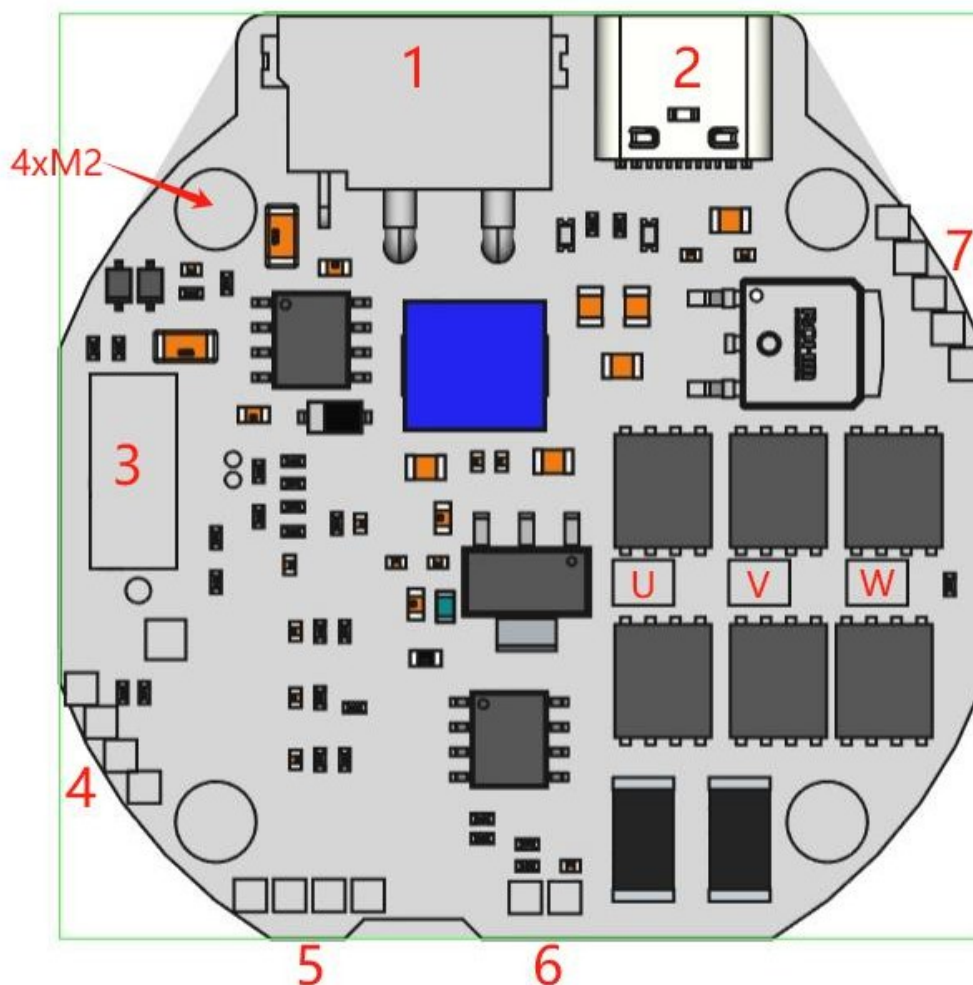
weights	388g±3
polar logarithm	14 pairs
phase (math.)	3 Phase
Driver Type	FOC
Deceleration Contrast Ratio	8:1

2 Drive Information

2.1 Appearance and three-dimensional dimensions



2.2 Interface Overview



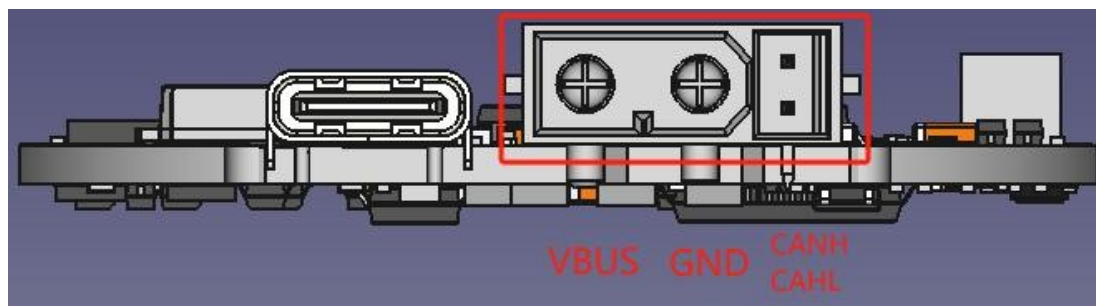
interface number	define
1	15~60V power supply and CAN communication integrated terminal
2	Type-C debugging interface and host computer communication interface
3	Interface Expansion Slot (Expandable RS485, EtherCAT, Model Air, Pulse Orientation, Oil) (interfaces/protocols for door control, etc.)
4	SWD Debug and Download Interface
5	Second encoder interface (I2C and UART support)
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	120A
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 supply and CAN communication terminals



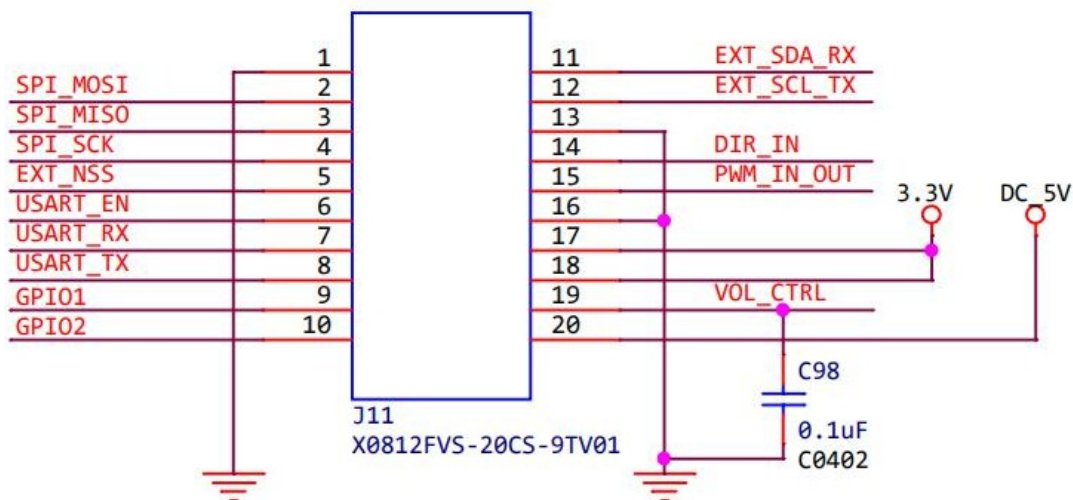
On-board terminal model XT30PB(2+2)-M, wire end model XT30(2+2)-F, brand name AMASS.

2.4.2 Type-C Debugging Interface

Type-C uses a standard cable specification and is compatible with commonly used PC or cell phone Type-C cables.

2.4.3 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:



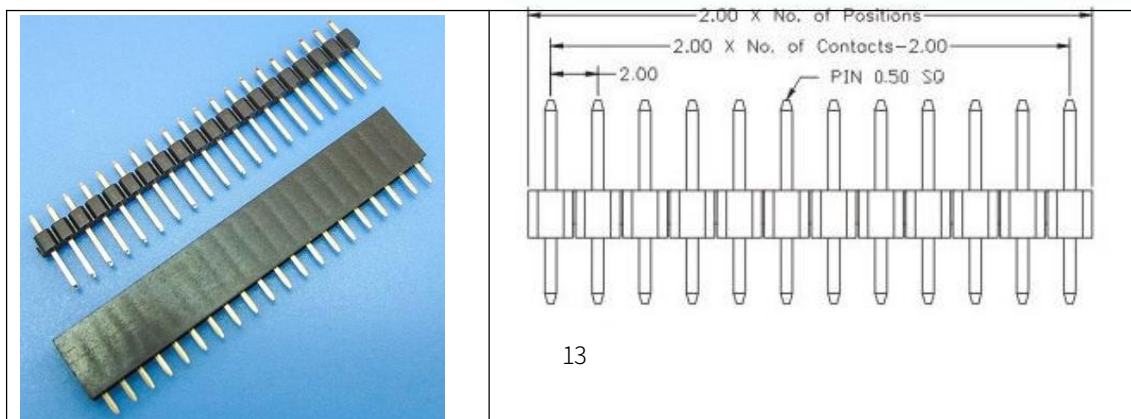
A third party can interact with the driver via SPI, USART, I2C, PWM, ADC, GPIO, etc. to realize various extended functions.

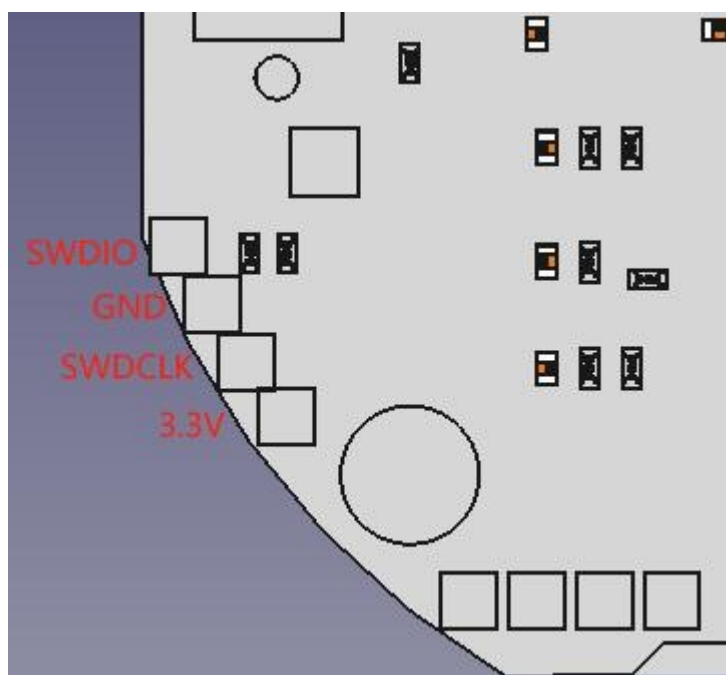
The onboard slot model number is X0812FVS-20CS-9TV01 (female), the expansion board slot model number is X0812WVS-20AS-9TV01 (male), and the brand name is Xingkun.

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

2.4.4 SWD Debug Interface

The 2mm spaced pin holes allow the user to solder 2mm inline single row pins as shown below:

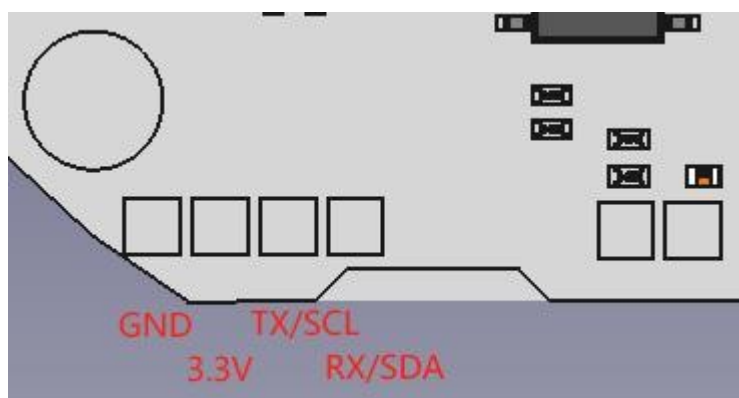




2.4.5 Second encoder interface

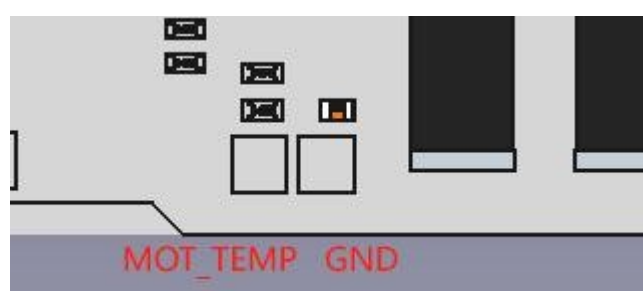
2mm spaced pin holes, user solderable 2mm inline single row pins, see 2.4.4.

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



2.4.6 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.

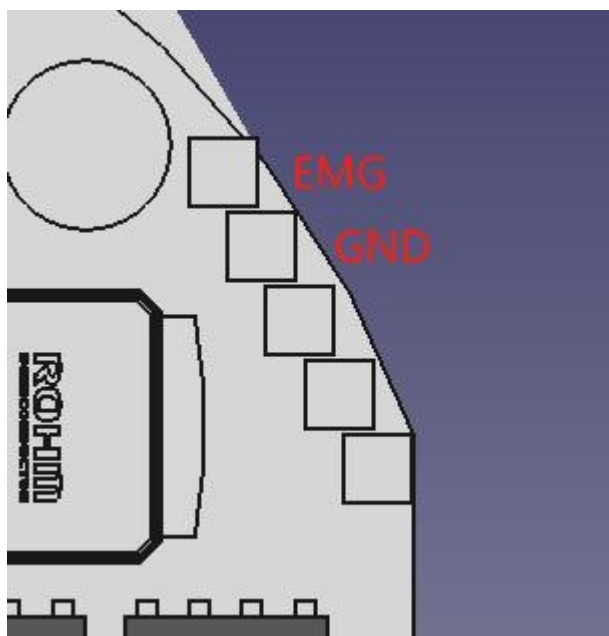


2.4.7 Brake/brake resistor interface

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

When it is the holding brake interface, the driver continuously transfers a current to this interface when power is applied so that the holding brake is open and the motor is able to operate normally. If the driver is powered off, this current stops, the holding brake locks, and the motor will lock in the power-off position.

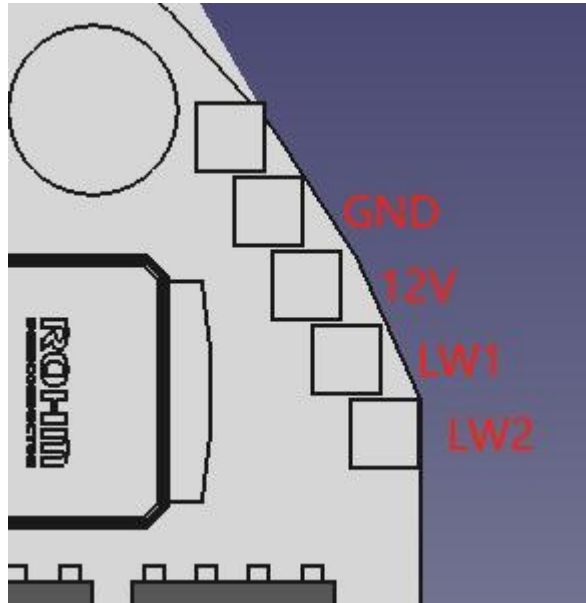
When the interface is a brake resistor, an external brake resistor (or bleeder resistor) can be connected to bleed current through this brake resistor when the reverse electromotive force is high above the threshold voltage, preventing emergency braking or damage to the drive by the reverse electromotive force.



2.4.8 Limit Switch Interface

The driver provides two limit switch ports and 12V power supply for 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. 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's core	FD6288Q	1
3	Magnetic encoder core	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 requirements on power supply voltage. A regulated power supply or a battery is recommended. The question often asked by users is, how do I choose a power supply? The following are some simple suggestions for reference only:

A few points for choosing a power supply:

◆ Current Requirement

In general, at least 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, RPM_{max}. The maximum value of the required supply voltage can be found in the formula:

$$V_a = \frac{P_a}{I_a} \times 1.25$$

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, I_{max}, at the most high speed, as shown in Eq:

$$P_a = V_a \times I_a \times 1.25$$

- ✓ Power + Communication Interface Cables

6010-8 with 2+2 power communication socket, please contact the after-sales service to recommend a suitable 2+2 cable. Please refer to 2.4.1 and **make sure to connect the right cable.**

Power supply positive and negative poles, otherwise there is a risk of burning the drive, because the drive has no anti-reverse connection capability. Meanwhile, **please pay attention to the definition order of the two communication lines, such as the order of CANH/CANL,** the wrong connection will lead to abnormal communication.

warnings

- ◇ Be sure to avoid touching the communication bus with your hands to prevent static electricity from damaging the drive's interface cores, especially in dry areas and dry seasons!
- ◇ Be sure not to unplug the power terminals with electricity!

- ✧ Avoid using a blank switch for the power supply, there is a risk of destroying the power core on the drive!
- ✧ Do not exceed 72V supply voltage!

✓ Type-C cable

Early in the tuning process, it is highly recommended to test the motor with a USB Type-C cable. The most commonly used cell phone Type-C cable can be used, not the 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 Start with odrivetool

The drive is compatible with odrive (<https://github.com/odriverobotics/odrive.git>), so use odrivetool as the upper unit for tuning.

Follow the steps below to install odrivetool:

➤ Windows (computer)

1. 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 Microsoft Microsoft Store.

2. Installation of visual c++ generation tools

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



3. Install odrivetool

Run Windows PowerShell using Administrator, run `pip install odrive` in it and enter to install. If

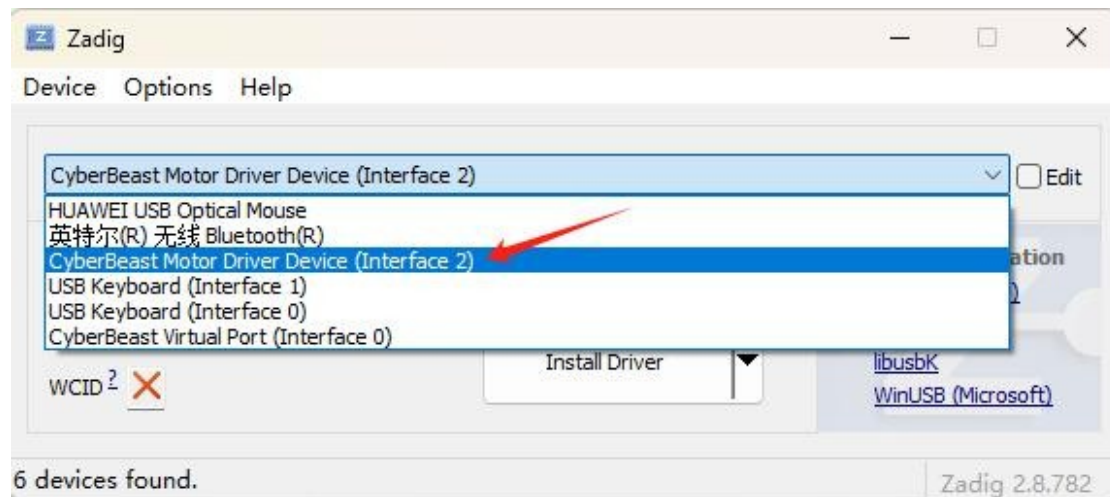
halfway through

🔧 error, please retry. If 🔧 error again and again, please restart your computer and try again.

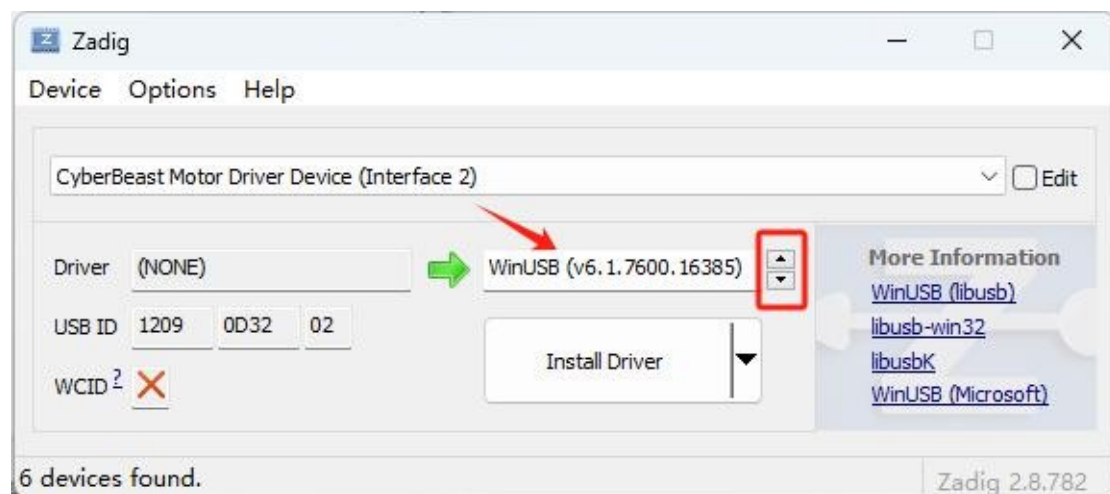
4. Installing the USB Driver

Go to <https://zadig.akeo.ie> and download the USB driver tool Zadig, connect the driver to the computer with the Type-C cable, the power light of the driver will be on, open Zadig and select "CyberBeast Motor Driver Device" from the drop-down box.

(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 installed WSL2, go to the WSL2 command line and follow the steps below (assuming the user WSL2).

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

(Ubuntu is installed):

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 uplink.

2) Connecting drives to WSL

Insert a type-C cable into Windows, by default Windows will load the driver for this USB port, but not WSL. To load this USB port into the WSL, please refer to the Microsoft document (<https://learn.microsoft.com/zh-cn/windows/wsl/connect-usb>) operation.

➤ Ubuntu

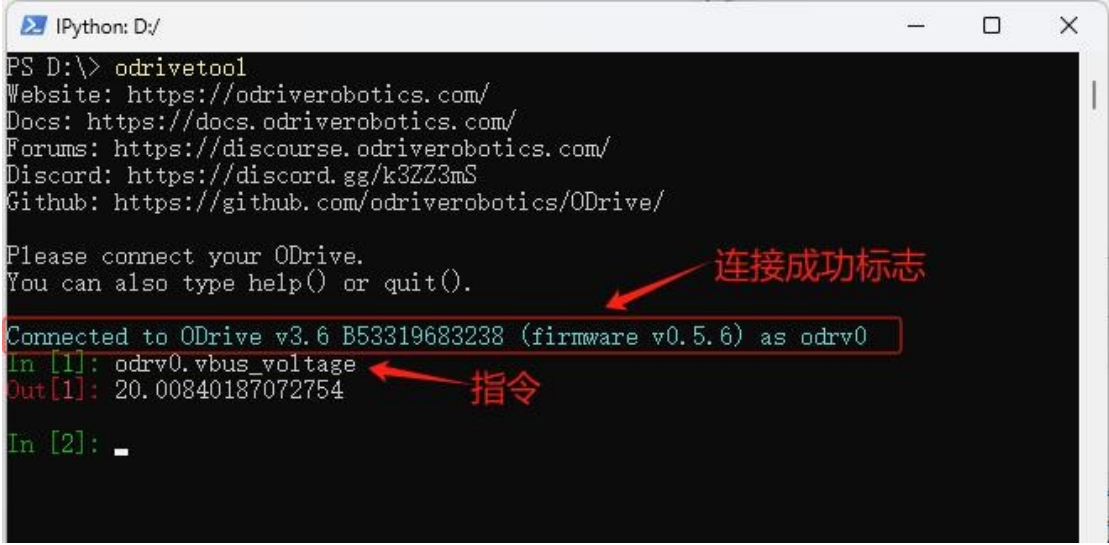
The installation process under Ubuntu is very similar to that under WSL, see the previous subsection.

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 odrivetool as described in the previous section, power up the motor and connect the USB Type-C cable, and then run the shell in the (Run odrivetool (type in odrivetool and enter) in Windows PowerShell or Linux Terminal, and the following figure shows a successful connection under Windows (the connection information is shown in green):



```
IPython: D:/
PS D:\> odrivetool
Website: https://odriverobotics.com/
Docs: https://docs.odriverobotics.com/
Forums: https://discourse.odriverobotics.com/
Discord: https://discord.gg/k3ZZ3mS
Github: https://github.com/odriverobotics/ODrive/

Please connect your ODrive.
You can also type help() or quit().

Connected to ODrive v3.6 B53319683238 (firmware v0.5.6) as odrv0
In [1]: odr0.vbus_voltage
Out[1]: 20.00840187072754
In [2]: _
```

Instruction example: `odr0.axis0.controller.input_vel`, `odr0` represents the current connected motor, by default the first connected motor is called `odr0`, the second one is called `odr1`, 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 in linux, which can be selected with the up, down, left, and right keys.

Instructions;

- ◆ The up and down keys will display the history of commands
- ◆ When a command is entered, the history of similar commands will be prompted, and the right-click will directly complete the command.

➤ **Setting critical thresholds (limits)**


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

- current threshold

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 6010-8, 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 electrical

The current temperature of the machine also affects the Q-axis current.

◆ Drive board temperature

If drive board temperature protection is enabled

(`odrv0.axis0.motor.fet_thermistor.config.enabled=1`), the drive

The current temperature of the 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:

$$i' = \frac{T}{T_{lim}} \times i$$

where i' is the final effective current threshold, i is the configured current threshold, and T is the current temperature (motor temperature or board temperature), is the set lower temperature limit

(`motor_thermistor.config.temp_limit_lower` and

`fet_thermistor.config.temp_limit_lower`), is the set upper temperature limit

(`motor_thermistor.temp_limit_upper` and `fet_thermistor.config.`)

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

- speed limit

The system global speed limit is limited to 30turn/s by the above command. Note that by default this speed limit does not work in torque-only mode, but can be enabled by flipping the following switch:

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

- Calibration Current

The default value of this current is 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, no.

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

Then the low voltage alarm will  be present at school on time.

➤ **Setting Key Hardware Parameters**

● 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 value to avoid the power supply being unable to discharge and causing the voltage to be pulled down, or by the reverse electromotive force.

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

Damage. Note, however, that if these two values are set to a value with a small absolute value, an alarm can easily be generated.

● polar logarithm

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

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

There are calibration alarms.

● torque constant


The torque constant is the torque produced by the motor divided by the Q-axis current, which is related to the motor Kv value by $\tau =$

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

8.27/.

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 for 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, allow the driver to control the output  based on temperature.

```
# Motor temperature protection
odrv0.axis0.motor.motor_thermistor.config.enabled = 1
odrv0.axis0.motor.motor_thermistor.config.temp_limit_lower = 20
odrv0.axis0.motor.motor_thermistor.config.temp_limit_upper = 100 #
Drive board temperature protection
odrv0.axis0.motor.fet_thermistor.config.enabled = 1
odrv0.axis0.motor.fet_thermistor.config.temp_limit_lower = 20
odrv0.axis0.motor.fet_thermistor.config.temp_limit_upper = 100 #
Get temperature
odrv0.axis0.motor.motor_thermistor.temperature #motor
temperature odrv0.axis0.motor.fet_thermistor.temperature
#drive board temperature
```

current (torque), thus protecting the drive and motor.

➤ PID Adjustment

```
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
```

The following procedure provides a reference for the user to adjust the PID parameters:

1. Setting the PID initial value
2. Set `vel_integrator_gain` to 0


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

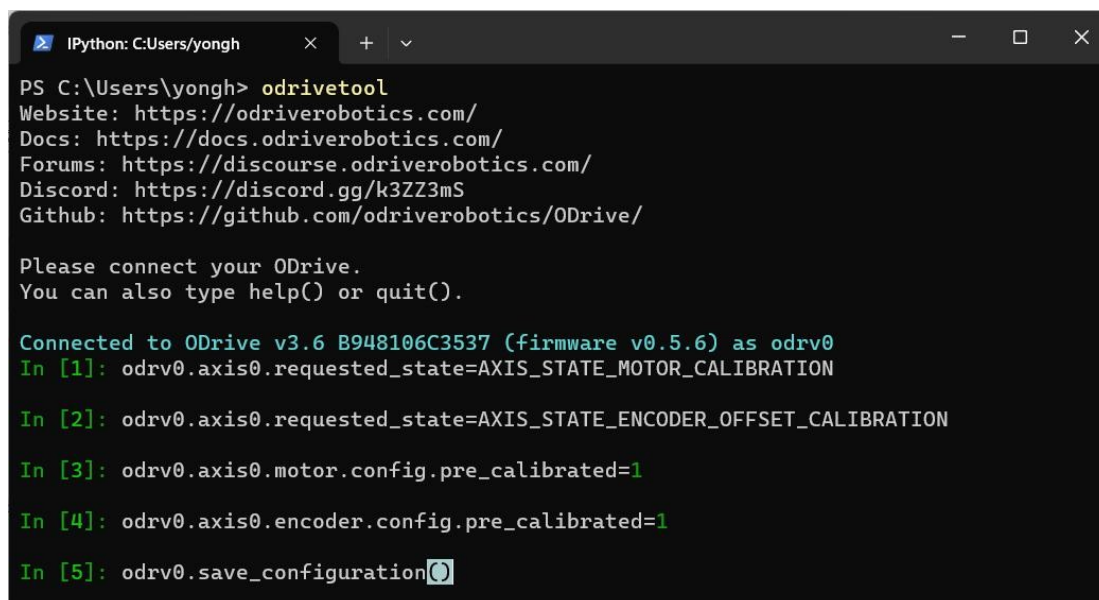
3. Adjust the `vel_gain` method:
 - 1) Rotate the motor in speed control mode, if the rotation is not smooth, jerky or vibrating, reduce `vel_gain` until the rotation is smooth.
 - 2) Next, increase `vel_gain` by about 30% each time until 🌀 noticeable jitter is present!
 - 3) At this point, reduce `vel_gain` by about 50% to stabilize it
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 the position control 🌀 shows significant overshoot (i.e., each time the position control motor overshoots 🌀 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 steps, you can set `vel_integrator_gain` to $0.5 \times \text{bandwidth} \times \text{vel_gain}$, where bandwidth is the system control bandwidth. What is control bandwidth? For example, if the time from the user setting the target position to the motor reaching the target position is 10ms, the control bandwidth is 100Hz, then $\text{vel_integrator_gain} = 0.5 \times 100 \times \text{vel_gain}$.

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 misperception of the naked eye.

Difference.

3.1.4 Power-Up Calibration

When users use the micro motor 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 and lose  axis no load, the calibration process is as follows:



```

IPython: C:\Users\yongh
PS C:\Users\yongh> odrivetool
Website: https://odriverobotics.com/
Docs: https://docs.odriverobotics.com/
Forums: https://discourse.odriverobotics.com/
Discord: https://discord.gg/k3ZZ3mS
Github: https://github.com/odriverobotics/ODrive/

Please connect your ODrive.
You can also type help() or quit().

Connected to ODrive v3.6 B948106C3537 (firmware v0.5.6) as odrv0
In [1]: odrv0.axis0.requested_state=AXIS_STATE_MOTOR_CALIBRATION

In [2]: odrv0.axis0.requested_state=AXIS_STATE_ENCODER_OFFSET_CALIBRATION

In [3]: odrv0.axis0.motor.config.pre_calibrated=1

In [4]: odrv0.axis0.encoder.config.pre_calibrated=1

In [5]: odrv0.save_configuration()
    
```

```

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

The 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 you will hear a sharp "beep". The results of the phase resistance and phase inductance measurements can be seen in the following table.

```

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

The above command is viewed:


- Step 2: Check the error code

Check the system error code after the first step, if  any red error code is present, 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 encoder calibration step, check the system for error codes. Typically  the error that occurs is `ERROR_CPR_POLEPAIRS_MISMATCH`, which indicates that the CPR of the encoder is set incorrectly, or that the number of pole pairs of the motor is set incorrectly.

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

Please check/set it by the following command:

- Step 5: Write in the motor calibration success flag
- Step 6: Write encoder calibration success flag
- Step 7: Store calibration results and reboot

3.1.5 Storage and backup parameters

After any parameter changes have been made, be sure to store them, otherwise the changes made will become invalid after a power failure or reboot. Driver after storing parameters

```
odrv0.save_configuration()
```

The machine will reboot.

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:

3.1.6 Three control modes

After the above preparation and parameterization, you can try to control the motor in different modes of rotation. 6010-8 supports position control, speed control, torque control, and motion control modes.

In position control mode, Filtered Position Control, Trapezoidal Position Control are supported.

(Trajectory Control), Circular Position Control;

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 combination of position, velocity and torque control mode, usually used in scenarios that require a strong instantaneous burst of force, such as robotic knee joints. Some users in the industry also refer to this as MIT control mode, which is derived from the MIT Open Source Mechanical Dog because it uses this motion control mode to control the motors.

In the subsequent detailed description of each control mode, the USB control commands are used as examples in this document, but the same control can also be done with communication protocols (e.g., CAN), and the logic is consistent.

➤ **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 smoothly articulates 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, it is necessary to adjust the filtering bandwidth according to the frequency of transmitting commands, and a better experience is to set the bandwidth to

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

is half of the command frequency (in Hz), e.g., if the command

is sent at 50 Hz, then: Enable filter position control:

Position control is then performed:

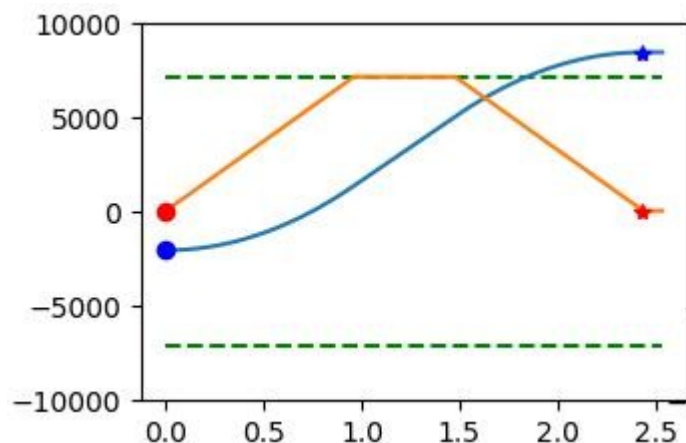
```
odrv0.axis0.controller.config.control_mode = 3
```

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

```
odrv0.axis0.controller.input_pos = 10 # units 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 move 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 #Glide speed maximum in turn/s
odrv0.axis0.trap_traj.config.accel_limit #Acceleration maximum in turn/s^2
odrv0.axis0.trap_traj.config.decel_limit # deceleration max in turn/s^2
odrv0.axis0.controller.config.inertia # inertia in Nm/(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 of the motor. The above four values are all greater than or equal to 0. Also note that the current threshold and velocity threshold mentioned above will still work globally, for example, if the maximum value of the glide speed mentioned above is set to vel_limit higher than the system level, it will work globally as vel_limit.

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

To enable trapezoidal curve

control mode, first: then position

control:

➤ **Circular Position Control (CPC)**

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

This mode is suitable for continuous incremental position control, such as when the robot hub rotates towards an orientation connection for a period of time, or when the conveyor track keeps running, and if the usual position control mode is used, the target position gradually increases to a large value, thus having the error of inaccurate localization due to floating-point precision 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 will be converted to a value within a single lap. If the user wants a single step to be more than a single lap, the following parameter can be set to a number greater than 1:

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

➤ **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 # unit turn/s
```

➤ **Ramped Velocity Control (RVC)**

Ramp speed control mode refers to gradually increasing the speed to the target value according to a certain slope, which is more favorable than the direct speed control mentioned above.

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

Add easing and enable the following:

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

The acceleration is controlled
by adjusting the slope: the
target velocity is then entered
for control:

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

- **Direct torque control (Torque 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:

Torque control is in Nm and current is in A in the driver firmware, so the torque constant also needs to be set to allow the driver to convert Nm to current to drive the motor to deliver torque on demand.

```
# The moment constant is approximately equal to 8.23
odrv0.axis0.motor.config.torque_constant = 8.23/12.3
```

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

The target speed is then entered for control:

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

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

➤ Ramped Torque Control (RTC)

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

Ramp torque control is quite similar to ramp speed control, 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:

$$arg = \times_i + \times_i +$$

$$i = \frac{arg}{39} - urr$$

$$i = \arg \text{urr}$$

Where τ_{arg} is the target moment, e_i is the position error, \dot{e}_i is the velocity error, K_p is the position control gain, K_d is the velocity control gain (or damping coefficient), and τ_{ff} is the feedforward moment.

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

Motion control mode

enable as follows:

adjusts the gain:

The motion is then controlled by inputting input_pos, input_vel, input_torque:

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

odrv0.axis0.controller.input_pos = 5 # units turns
odrv0.axis0.controller.input_vel = 30 # units turn/s
odrv0.axis0.controller.input_torque = 2 # units Nm
```




3.1.7 advanced

1 List of commonly used instructions

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()	After modifying the parameters, or after the motor automatically recognizes the parameters or is calibrated, be sure to execute this command to store the modifications, otherwise the motor will not be able to operate. Then all modifications are lost after 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 Master Version Number, 6010-8 The current master version number is 3
	odrv0.hw_version_minor	Hardware minor version number, 6010-8 current minor version number is 8

	odrv0.hw_version_variant	Different model numbers for the same hardware configuration, 6010-8 vs. The corresponding type number is 1
	odrv0.can.config.r120_gpio_num	Controls the 120R matching resistor switch of the CAN interface GPIO number
	odrv0.can.config.enable_r120	120R Matching Resistor Switch to Control CAN Interface
	odrv0.can.config.baud_rate	Baud rate setting for CAN
parameters configuration	odrv0.config.dc_bus_undervoltage_trip_level	Low Voltage Alarm Threshold (V)
directives	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_calibration_max_voltage	Maximum voltage value for motor parameter recognition, generally this value is slightly less than half of the supply voltage, such as 24V supply, can be set. 10
	odrv0.axis0.motor.config.calibration_current	Maximum current value for motor parameter recognition, this value can generally be Set to 2~5A, not too large 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 least significant bit of the The IO number is set to 1, and the IO number for the maximum limit is set 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 bits The control target values are all based on this user zero point.

	odrv0.axis0.motor.motor_thermistor.config	Configure the motor temperature sensor:
	odrv0.axis0.motor.fet_thermistor.config	enabled: enabled or not temp_limit_lower: lower temperature limit temp_limit_upper: upper temperature limit
	odrv0.axis0.motor.motor_thermistor.temperature	Motor temperature
	odrv0.axis0.motor.fet_thermistor.temperature	Drive Temperature
Calibration Instructions	odrv0.axis0.requested_state=4	The motor is parameterized, including phase resistance, phase inductance, and  calibration of the three-phase current balance. This process will take 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. error and confirm that there is no error before proceeding with other operations.
	odrv0.axis0.requested_state=7	Perform a calibration of the encoder. Before performing this operation, make sure that there is no load on the motor input  shaft and that the motor is secured by hand or other device. After this operation is performed, 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 confirm that there are no errors error before proceeding with other subsequent steps.
	odrv0.axis0.encoder.config.pre_calibrated=1	Write Pre-calibration Successful indicates that calibration will not be performed each time power is applied. This parameter is only used after the above calibration has been successful. The program can be written in, otherwise it will fail.
	odrv0.axis0.controller.config.load_encoder_axis=0	Make sure that the currently operated motor is the 0th motor. This operation Necessary in BETA only, not in production version.
	odrv0.axis0.requested_state=1	Stop motor, enter idle state
	odrv0.axis0.requested_state=8	Start motor, enter closed loop

control command	odrv0.axis0.motor.config.current_lim	Maximum line current of motor operation (A), exceeding this value will be reported as Overcurrent alarm. Note that this value must not be greater than 100.
	odrv0.axis0.controller.config.vel_limit	Maximum motor running speed (turn/s), motor rotor speed A speed alarm will be reported if the degree exceeds this value.
	odrv0.axis0.controller.config.enable_vel_limit	Velocity limit switch, when True, the above vel_limit is generated. is valid, and no effect when 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 mode in which the user enters the control value. Definite de-control motor operation: 0: Idle 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 speed loop PID control
	odrv0.axis0.controller.config.vel_integrator_gain	I value for speed 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.config.pos_gain	P value for position loop PID control
	odrv0.axis0.controller.input_torque	target for torque control, or speed control/position control for Torque Feed Forward (Nm)
	odrv0.axis0.controller.input_vel	Target for speed control, or speed 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 the integer needs to be	

		less than odrv0.axis0.encoder.config.cpr
	odrv0.axis0.trap_traj.config	Contains three parameters: <ul style="list-style-type: none"> ➤ accel_limit: maximum acceleration (rev/s²) ➤ decel_limit: maximum deceleration rate (rev/s²) ➤ vel_limit: maximum velocity (rev/s) These three parameters are used in the
		odrv0.axis0.controller.config.input_mode Functions when trapezoidal curves are used, adjusting the position control The plus-deceleration effect of the
	odrv0.axis0.controller.config. input_filter_bandwidth	The positional filter bandwidth, a parameter in the odrv0.axis0.controller.config.input_mode Functions for position filtering, adjusts the position control The plus-deceleration effect of the

4 Graphics adjustment and testing

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

1. environmental preparation

```
pip install numpy matplotlib
```

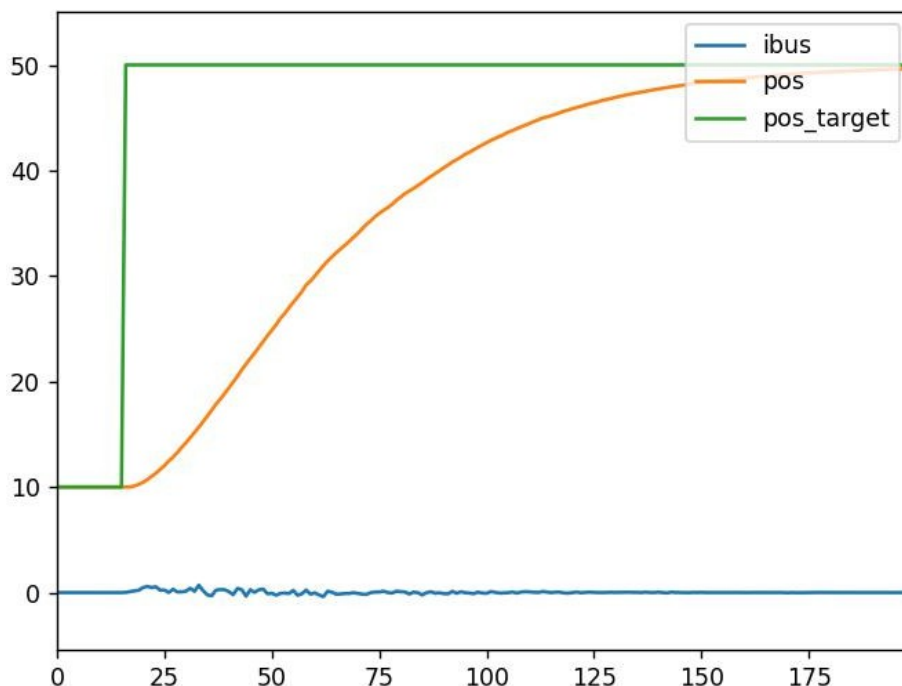
Installation of calculation and graphics libraries:

2. Graphical parameter input 🌀

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

At the odrivetool command line interface, bring up the graphics library and read any motor operation indicators such as:

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



3 ~~USB and CAN compatibility~~

In earlier versions of this product (hardware version less than or equal to 3.7, which can be obtained by the commands `odrv0.hw_version_major` and `odrv0.hw_version_minor` in the next subsection 3.1.7), USB is not compatible with CAN, and it is possible to switch between the two communication modes by the following way (hardware version greater than 3.7 can be USB is not compatible with CAN:)

- Switching to CAN during USB communication

When CAN is disabled, the user can communicate using the Type-C interface, in which case, the following command can be used to switch to CAN:

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

- Switching to USB for CAN communication

When CAN is enabled, the user first switches the motor to the idle state by sending the message `Set_Axis_State` (parameter 1 for the idle (IDLE) state), and then switches to USB by sending the message `Disable_Can` (see 4.1.2).

Please note that regardless of whether you are switching from USB to CAN or CAN to USB, the motor must first be in the idle (IDLE) state or the switchover will fail.

4 ~~CAN Impedance~~

A 120 ohm impedance matching resistor is already on-board the driver and can be turned on or off by the user as needed, example command

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

Below:

§ User Configuration

By default, the user position read from the motor, and the input value when doing position control, is 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 the user's zero point relative to the encoder's zero point:

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

§ Limit switch

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

```
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
```

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 perform the relevant operation at this time.

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

3.2 Firmware Update Download

Firmware can be burned via the SWD connector (2.4.4) or Type-C connector (2.4.2) in the following three ways:

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 the Nation Burner (<https://cyberbeast.cn/filedownload/766844>) and Unzip it to any directory and run it.

Then, connect the Type-C connector, enter odrivetool and execute the following command to put the drive in DFU mode:

```
odrv0.enter_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 odrivetool for testing.



2. SWD (JLink or DAP) writing

Using SWD for multiple downloads 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 supports common debugging tools such as STLink, JLink, DAP, etc. for erase, burn, reset, etc. **Please note that the drive must be connected using the SWD interface. Please refer to 2.4.4 for the wire sequence of SWD. There is a 3.3V power supply in the SWD connector, so please do not connect the wires in the wrong sequence to avoid damaging the driver!**

```
pip install pyocd
```

1. mounting
2. burn

```
pyocd list
```

First, list  connected debugging tools:

```

管理员: Windows PowerShell
PS D:\projects\cheetah\ODrive\Firmware\keil\Objects> pyocd list
# Probe/Board Unique ID Target
-----
0 STM32 STLink 6000470018000037544B524E n/a
PS D:\projects\cheetah\ODrive\Firmware\keil\Objects>
    
```

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

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

```

管理员: Windows PowerShell
PS D:\projects\cheetah\ODrive\Firmware\keil\Objects> pyocd load . \ODrive_N32G455.bin -a 0x8000000
0000477 W Generic 'cortex_m' target type is selected by default; is this intentional? You will be able
to debug most devices, but not program flash. To set the target type use the '--target' argument or
'target_override' option. Use 'pyocd list --targets' to see available targets types. [board]
0000529 I Loading D:\projects\cheetah\ODrive\Firmware\keil\Objects\ODrive_N32G455.bin at 0x08000000 [l
oad_cmd]
[=====] 100%
0004543 I Erased 0 bytes (0 sectors), programmed 230112 bytes (0 pages), skipped 0 bytes (0 pages) at
56.02 kB/s [loader]
PS D:\projects\cheetah\ODrive\Firmware\keil\Objects>
    
```

3.2.3 Motor Wizard (coming soon)

4 Communication protocols and examples

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 default rate of 500Kbps. **Note: USB is not compatible with CAN in earlier hardware releases (less than or equal to 3.7), see section 3) in 3.1.7. How to switch from USB to CAN.**

4.1.1 PF format

CAN communication uses the 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, 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 (test encoding can be done 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|Input_Pos starts with 4 bytes at the beginning of the 0th byte and is encoded as C3 F5 48 40 (Floating point 3.14 is encoded as a 32-bit number using the IEEE 754 standard 0x4048f5c3), Vel_FF starts with 2 bytes at the beginning of the 4th byte and is encoded as E8 03 (1000=0x03E8), Torque_FF starts with 2 bytes at the beginning of the 6th byte and is encoded as 88 13 (5000=0x1388), then the 8 bytes of communication data is as follows.
1000=0x03E8), Torque_FF in the 2 bytes at the beginning of the 6th byte, encoded as 88 13 (5000=0x1388), then 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)	Orientation	parameters
0x001	Heartbeat	Motor → Main unit	Axis_Error Axis_State Motor_Flag Encoder_Flag Controller_Flag
0x002	Estop	Host → Motor	
0x003	Get_Error	Motor → Main unit	Error_Type
0x004	RxSdo	Motor → Main unit	
0x005	TxSdo	Motor → Main unit	
0x006	Set_Axis_Node_ID	Host → Motor	Axis_Node_ID
0x007	Set_Axis_State	Host → Motor	Axis_Requested_State
0x008	Mit_Control	Mainframe → motor	
0x009	Get_Encoder_Estimates	Motor → Main unit	Pos_Estimate Vel_Estimate
0x00A	Get_Encoder_Count	Motor → Main unit	Shadow_Count Count_In_Cpr

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

Detailed descriptions of all the messages are given below:

➤ Heartbeat

CMD ID: 0x001 (motor → host)

start byte	name (of a thing)	typology	odrivetool access
0	Axis_Error	uint32	odrv0.axis0.error
4	Axis_State	uint8	odrv0.axis0.current_state

5	Motor_Flag	uint8	1: odrv0.axis0.motor.error is not 0 0: odrv0.axis0.motor.error is 0
6	Encoder_Flag	uint8	1: odrv0.axis0.encoder.error is not 0 0: odrv0.axis0.encoder.error is 0
7	Controller_Flag	uint8	bit7: odrv0.axis0.controller.trajectory_done bit0: 1: odrv0.axis0.controller.error is not 0
			0: odrv0.axis0.controller.error is 0

➤ Estop

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

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

➤ Get_Error

CMD ID: 0x003 (motor → host)

Enter (host → motor):

start byte	name (of a thing)	typology	clarification
0	Error_Type	uint8	0: Get motor abnormality 1: Get Encoder Exception 2: Acquisition of Degenerate Anomalies 3: Get Controller Exception

Transmission 🌀 (motor → host):

start byte	name (of a thing)	typology	odrivetool access
0	error	uint32	Different inputs Error_Type: 0: odrv0.axis0.motor.error 1: odrv0.axis0.encoder.error 2: odrv0.axis0. 3: odrv0.axis0.controller.error

➤ 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 a JSON file of all parameters and IDs corresponding to interface functions: https://cyberbeast.cn/filedownload/784822
3	reserve	uint8	
4	Value	uint8[4]	This varies depending on the Endpoint_ID, as described in the JSON above. If Endpoint_ID corresponds to a read-write float value, then the 4 bytes here are the IEEE encoded float value, which will be set to 1 when opcode=1. Write the float value.

Lose 🌀 (when opcode=0 above):

start byte	name (of a thing)	typology	clarification
0	opcode	uint8	Fixed to 0
1	Endpoint_ID	uint16	Please download a JSON file of all parameters and IDs corresponding to interface functions: https://cyberbeast.cn/filedownload/784822
3	reserve	uint8	
4	Value	uint8[4]	This 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 the little endian Byte-ordered uint32.

➤ TxSdo

CMD ID: 0x005 (motor → host)

The usage is the same as RxSdo when 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>).



✓ Host → Motor

CAN number frame rate	hidden meaning	clarification

BYTE0	<p>Position: 16 bits total, BYTE0 is high 8 bits, BYTE1 is 8 bits lower</p>	<p>The actual position is of type double and needs to be converted to 16-bit int type, the conversion process is:</p>
BYTE1		
	<p>Multiturn position of the input  axis in radians (RAD)</p>	$\text{pos_int} = (\text{pos_double} + 12.5) * 65535 / 25$
BYTE2	<p>Speed: 12 bits total, with BYTE2 being the high 8 bits.</p> <p>BYTE3[7-4] (high 4 bits) is its low 4 bits. Indicates the angular velocity of the output  axis in RAD/s.</p> <p>KP value: 12 bits total, BYTE3[3-0] (low 4) BYTE4 is the high 4 bits and BYTE4 is the low 8 bits.</p>	<p>The actual speed is of double type and needs to be converted to 12-bit int type, the conversion process is:</p> $\text{vel_int} = (\text{vel_double} + 65) * 4095 / 130$ <p>The KP value is actually of double type and needs to be converted to 12-bit int type by the conversion procedure:</p> $\text{kp_int} = \text{kp_double} * 4095 / 500$
BYTE3		
BYTE4		
BYTE5	<p>KD value: 12 bits total, BYTE5 is the high 8 bits and BYTE6[7-4] (high 4 bits) is the low 4 bits.</p> <p>Torque: 12 bits total, BYTE6[3-0] (lower 4 bits) is the high 4 bits and BYTE7 is the low 8 bits. The unit is N.m.</p>	<p>The KD value is actually of double type and needs to be converted to 12-bit int type, the conversion process is:</p> $\text{kd_int} = \text{kd_double} * 4095 / 5$ <p>The actual torque is of double type and needs to be converted to 12-bit int type, the conversion process is as follows:</p> $\text{t_int} = (\text{t_double} + 50) * 4095 / 100$ <p>Torque constant in N.m/A</p>
BYTE6		
BYTE7		

✓ Motor → Main unit

CAN number frame rate	hidden meaning	clarification
BYTE0	node id	Drive node id

BYTE1	<p>Position: 16 bits total, BYTE1 is high 8 bits, BYTE2 is 8 bits lower</p> <p>Multiturn position of the input  axis in radians (RAD)</p>	<p>The actual position is of double type, which needs to be converted from 16-bit int type, and the conversion process is as follows:</p> $\text{pos_double} = \text{pos_int} * 25 / 65535 - 12.5$
BYTE2		
BYTE3	<p>Speed: 12 bits total, with BYTE3 being the high 8 bits.</p> <p>BYTE4[7-4] (high 4 bits) is its low 4 bits.</p> <p>Indicates the angular velocity of the output -axis in RAD/s</p> <p>Torque: 12 bits total, BYTE4[3-0] (lower 4 bits) is the high 4 bits and BYTE5 is the low 8 bits. Units is N.m.</p>	<p>The actual speed is of type double, which needs to be converted from 12-bit int, and the conversion process is:</p> $\text{vel_double} = \text{vel_int} * 130 / 4095 - 65$ <p>The actual torque is of double type, which needs to be changed from 12-bit int.</p> <p>The type is converted over, and the conversion process is:</p> $\text{t_double} = \text{t_int} * 100 / 4095 - 50$ <p>Torque constant in N.m/A</p>
BYTE4		
BYTE5		

➤ 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 measure)	odrivetool access
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)

for torque 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.traj_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.traj_traj.config.accel_limit
4	Traj_Decel_Limit	float32	rev/s ²	odrv0.axis0.traj_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	typology	unit (of	odrivetool access
------------	------------	----------	----------	-------------------

	thing)	y	measur e)	
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)

Clears all errors and exceptions.

➤ Set_Linear_Count

CMD ID: 0x019 (host → motor)

Sets the encoder absolute 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 Action

4.1.3.1 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 Parameter: 7 Calibration of the encoder

4.1.3.2 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 parameter: 2/2 Set control mode to speed control, input mode is the velocity ramp
0x007	data frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 8 Entering closed-loop control
0x00D	data frame	00 00 20 41 00 00 00 00	Message: Set_Input_Vel Parameter: 10/0 Set the target speed and torque feedforward, where the target speed is 10 (floating point number: 0x41200000), the Torque feedforward is 0 (floating point number: 0x00000000)


4.1.3.3 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 parameter: 3/3 Set control mode to position control, input mode for positional filtering
0x007	data frame	08 00 00 00 00 00 00 00	Message: Set_Axis_State Parameters: 8 Entering closed-loop control
0x00C	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 torque feedforward, where the target position is 2.2 (floating point

			number: 0x400CCCCD), the torque feedforward and velocity feedforward Feed is 0
--	--	--	---

4.1.4 CANOpen Compatibility


Interworking with CANOpen is possible if the node ID is properly assigned. The following table lists  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. Cycle messages can be turned on/off (a value of 0 means off, other values mean cycle time in ms) through a series of configurations under `odrv0.axis0.config.can`, as shown in the table below:

messages	odrivetool configuration	default value
Heartbeat	<code>odrv0.axis0.config.can.heartbeat_rate_ms</code>	100
Get_Encoder_Estimates	<code>odrv0.axis0.config.can.encoder_rate_ms</code>	10
Get_Motor_Error	<code>odrv0.axis0.config.can.motor_error_rate_ms</code>	0
Get_Encoder_Error	<code>odrv0.axis0.config.can.encoder_error_rate_ms</code>	0
Get_Controller_Error	<code>odrv0.axis0.config.can.controller_error_rate_ms</code>	0
Get_Sensorless_Error	<code>odrv0.axis0.config.can.sensorless_error_rate_ms</code>	0
Get_Encoder_Count	<code>odrv0.axis0.config.can.encoder_count_rate_ms</code>	0
Get_Iq	<code>odrv0.axis0.config.can.iq_rate_ms</code>	0
Get_Sensorless_Estimates	<code>odrv0.axis0.config.can.sensorless_rate_ms</code>	0
Get_Bus_Voltage_Current	<code>odrv0.axis0.config.can.bus_vi_rate_ms</code>	0

By default, the first two cycle messages are turned on at  , so when the user monitors the CAN bus, he or she will see two kinds of dissipation

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

Messages are broadcast with a set period. They can be turned off by the user with the following command:
see 4.1.2 for details of each message.

4.2 Python SDK

Please first install odrivetool (pip install -upgrade odrive) by referring to section 3.1. See section 3.1.7 for python development with all the commands described in that subsection.

The following 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.
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.VELOCITY_CONTROL
odrv0.axis0.controller.config.input_mode=odrive.utils.InputMode.VEL_RAMP
odrv0.axis0.controller.config.vel_ramp_rate=50
odrv0.axis0.requested_state=odrive.utils.AxisState.CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_vel=15
odrive.utils.dump_errors(odrv0)
time.sleep(5)
odrv0.axis0.controller.input_vel=0
```

4.2.2 Practical: speed control

```
import odrive

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

4.2.3 Practical: position control

4.2.4 Practical: data collection

In the process of R&D and integration, users often need to collect motor operation data, such as collecting voltage and current changes, position and speed changes, etc. The Python SDK integrates a powerful data capturing capability, which can realize massive operation data capturing with simple scripts, making R&D and integration easier.

The code in the following section is based on the previous position control example, with the addition of real-time position and current data grabbing, and

```
import odrive
import numpy as np

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

Save the data as a csv file.

In the BulkCapture statement, data_rate represents the sampling frequency in hz, duration represents the sampling time in seconds, and the lambda expression can be used to insert any mathematical operation to facilitate data analysis.

4.3 Arduino SDK

The user can use the Arduino to control the motor via the 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 (by Teensy 4.0 and Teensy 4.1)

Other Arduino-compatible boards can be accessed using the MCP2515-based CAN Expansion Board The following 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 3.1.3, 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 need to be too high, but you can increase this bandwidth value if you use a faster Arduino):

- Configuring CAN

Configure CAN as follows:

- Installing the ODriveArduino Library

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

- 1) Open the Arduino IDE
 - 2) Sketch -> Include Library -> Manage Libraries
 - 3) Enter "ODriveArduino" in the search box.
 - 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 ----- */
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

// frequency 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); // 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 */

#ifndef 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.
odrv0_user_data; 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*&gt;(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); odrv0.onStatus(

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

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

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

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

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

Serial.println("Enabling closed loop control..." );
while (odrv0_user_data.last_heartbeat.Axis_State !
=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
  // The following is an example of
  how to do this: // Immediately,
  but can be delayed. 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(p
  hase), // 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.
  Serial.println(feedback.Vel_Estimate);
}
}
```

4.4 ROS SDK

The following steps have been tested and validated on Ubuntu 23.04 and ROS2 Iron, but are not supported on MAC and Windows platforms and have not been validated on other ROS2 releases, and will 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 that the above `odrive_can_node` node runs in the namespace `odrive_axis0` (which can be `./launch/example_launch.yaml`). Once the node in 4.4.1 is up and running, you can view the published topic messages such as:

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

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

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

5 FAQs and exception codes (to be updated)

5.1 Frequently Asked Questions (FAQ)

5.2 exception code