AK Series Module Driver Manual

V1.0.17

Catalogue

Precautions

1.Ensure that there are no short circuits in the circuit and that interfaces are connected correctly as required.

2. ⚠危险 The driver board will heat up during output; please use it carefully to avoid burns.

3. ⚠警告 Before use, please check if all parts are intact. If any parts are missing or aged, please stop using it and contact technical support in time.

4. ⚠注意 Please strictly follow the working voltage, current, temperature, and other parameters specified in this document; otherwise, it will cause permanent damage to the product!

Product Features

The AK series motor driver board adopts high-performance drive chips in the same class, uses Field Oriented Control (FOC) algorithm, and is equipped with advanced self-disturbance control technology for speed and angle control. It can be used with CubeMarsTool parameter setting software for parameter setting and firmware upgrades. In terms of hardware, the inner loop uses a 16-bit high-precision encoder, supporting up to 21 bits (custom firmware required), and the CAN communication uses a safer isolated CAN interface, along with a more reliable plug, greatly enhancing the reliability of the product's use and communication; in terms of software, the upper computer CubeMarsTool has been fully upgraded, and there is no need to switch between servo mode and force control mode, the control interface is more concise, and a large number of simplifications have been made in the operation, fully improving the customer's experience.

Disclaimer

Thank you for purchasing the AK series modular motor. Before using it, please read this statement carefully. Once used, it is considered as recognition and acceptance of all the contents of this statement. Please strictly follow the product manual and relevant laws, regulations, policies, and guidelines for the installation and use of the product. During the use of the product, the user promises to be responsible for their own actions and all consequences arising therefrom.

Any losses caused by improper use, installation, or modification of the product by the user, CubeMars will not assume legal responsibility.

CubeMars is a trademark of Nanchang Kude Intelligent Technology Co., Ltd. and its affiliated companies. The product names and brands mentioned in this document are trademarks of the companies. This product and manual are copyrighted by Nanchang Kude Intelligent Technology Co., Ltd. No copying or reprinting is allowed without permission. The final interpretation of the disclaimer belongs to Nanchang Kude Intelligent Technology Co., Ltd.

Version Change Record

Date	Version	Change
2021.09.01	Ver. 1.0.0	Create
2021.10.08	Ver.1.0.1	5.1 & 5.2 Code Changes
2021.10.29	Ver.1.0.2	5.1, 5.2 and 5.3 Data definition updates
2021.11.15	Ver.1.0.3	CAN message reception definition
2021.11.24	Ver.1.0.4	UART protocol update in 5.2
2021.11.30	Ver.1.0.5	Addition of information in 5.3
2022.01.20	Ver.1.0.6	5.3 Changing the speed of the AK60-6 motor in the center
2023.07.19	Ver.1.0.10	1. Red light description explanation 2.New 80-8 60KV MIT parameters added
2023.08.29	Ver.1.0.12	1. Modify the position and speed loop routine code.

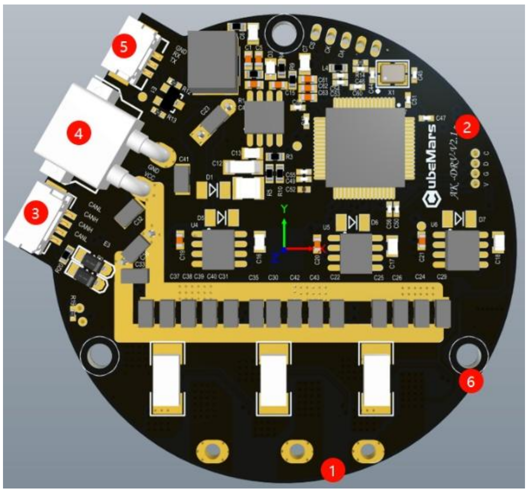
		2. Modify the servo mode byte order to indicate
2023.12.11	Ver.1.0.13	Error reporting refinements, origin mode and send code changes
2023.12.28	Ver.1.0.14	5.1.6 Origin Mode Code Modification 5.1.7 Position Velocity Mode Code Modification 5.1 Chapter Position Loop Velocity Position Loop Description Modification 4.2.1, 4.2.2 and 4.4 Add video link and description
2024.01.19	Ver.1.0.15	1.3 Fixed the red light being slightly illuminated 5.1.6 Explained setting permanent zero point 5.3 Deleted AK80-80 motor parameters and added AK80-64 motor parameters
2024.07.29	Ver.1.0.16	1.1 Change the maximum working voltage to the allowable working voltage range; 1.4 Changed the wire for the serial and can communication cables to 30AWG; 5.2.1 explained frames 0x09, 0x29 and 0x2C of the reply frame;5.2.1(where speed is int16 type, range -32000~32000 represents -320000~320000 electrical speed) This sentence deletes rpm; The example code of can communication between servo and MIT mode is modified in its entirety. 5.2.2 Correct the motor outer ring position

		formula
2025.03.12	Ver.1.0.17	1.1 Updated drive product specifications 5.3 Add AK45-36, AK45-10 and AK40-10 motor parameters 5.2.2 Optimize servo mode serial port telegram protocol format 5.3.1 Add motion control serial port debugging instruction description

1. Driver Product Information

aw6s

1.1Driver Appearance Introduction&Product Specifications



①Three-phase wires connection port

②Hardware version

③CAN Connection port

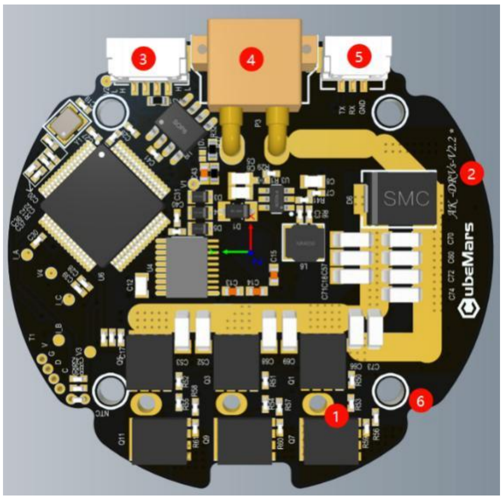
④DC Power interface

⑤Serial port

⑥Mounting holes

Production specifications(AK-DRV-V2.1 small size)	
Rated working voltage	48V
Allowable working voltage	18-52V
Rated working current	20A
Maximum current	60A
Standby power consumption	≤50mA
CAN bus bit rate	1Mbps(No change recommended)

Size	62mm×58mm
Working environment temperature	-20℃ to 65℃
Maximum allowable temperature for control board	100℃
Encoder precision	14bit（single turn absolute)



①Three-phase wire connection port

②Hardware version

③CAN Communication Connection Port

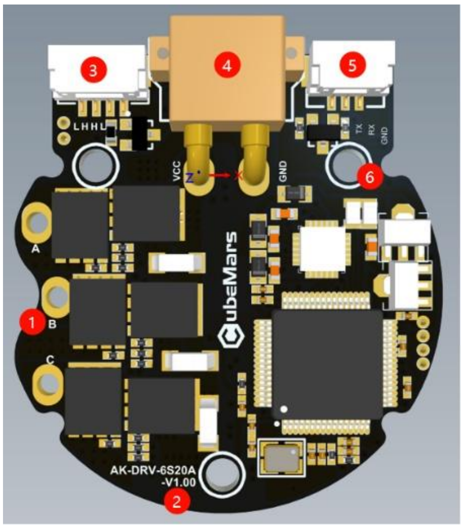
④DC Power interface

⑤Serial port

⑥Mount holes

Product Specifications(AK-DRV-V2.2 small size)	
Rated working voltage	24V
Allowable working voltage	18-28V
Rated working current	10A
Maximum current	30A

Standby power consumption	≤50mA
CAN bus bit rate	1Mbps（No change recommended）
Size	54mm×50mm
Operation environment temperature	-20℃ to 65℃
Maximum allowable temperature for control board	100℃
Encoder precision	14bit（single turn absolute）



①Three-phase wire connection port

②Hardware version

③CAN Communication Connection Port

④DC Power interface

⑤Serial port

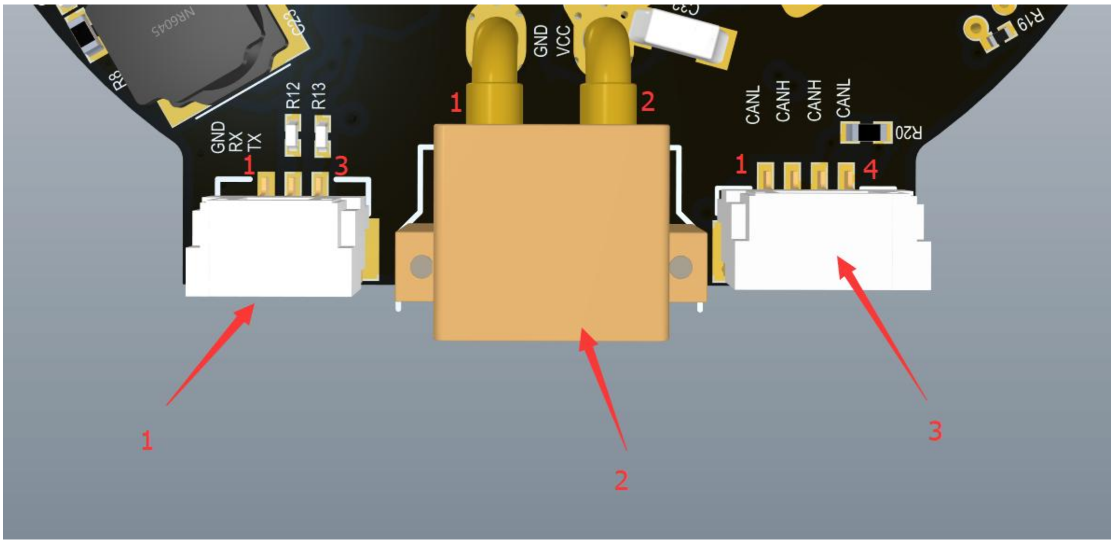
⑥Mount holes

Product specifications(AK-DRV-V1.0 mini size)	
Rated working voltage	24V
Allowable working voltage	16-28V
Rated working current	10A
Maximum current	20A
Standby power consumption	≤60mA
CAN bus bit rate	1Mbps (No change recommended)

Size	42mm×39mm
Operation environment temperature	-20℃ to 65℃
Maximum allowable temperature for control board	100℃
Encoder precision	14bit（single turn absolute）

1.2 Driver Interface and Definition

1. 2. 1 Driver Interface Diagram



1. 2. 2 Recommended Brands and Models for Driver Interface

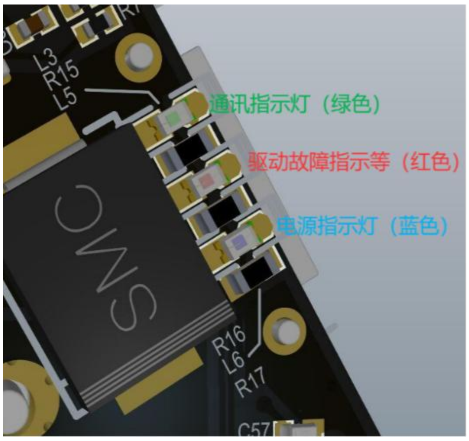
No.	Onboard Interface Model	Brand	End-of-Line Interface Models	Brand
1	A1257WR-S-3P	CJT	A1257H-3P	CJT

2	XT30PW-M	AMASS	XT30UPB-F	AMASS
3	A1257WR-S-4P	CJT	A1257H-4P	CJT

1.2.3 Driver Interface Pin Definitions

No.	Interface function	pin	clarification
1	**Serial Communication**	1	Serial Signal Ground（GND）
		2	Serial Signal Output（TX）
		3	Serial Signal Input（RX）
2	**Power Input**	1	Power Negative（-）
		2	Power Positive（+）
3	**CAN Communication**	1	CAN communication low side（CAN_L）
		2	CAN communication high side（CAN_H）
		3	CAN communication high side（CAN_H）
		4	CAN communication low side（CAN_L）

1.3 Driver Indicator Light Definitions



Indicator Light	
1.Power Indicator Light (Blue when lit)）	Power indicator, used to indicate the power supply of the driver board, under normal circumstances when the power supply is plugged in will light up blue, if the blue light does not light up when the power supply is plugged in, please remove the power supply immediately, do not power up again.
2.Communication indicator (green when lit)	Communication indicator, used to indicate the driver board communication, under normal circumstances the driver board will light up green only when the normal communication, if the green light does not light up, please first check the CAN communication wiring is normal.
3.Drive Fault Indicator Light (Red when lit)	Drive fault indicator, used to indicate the failure of the drive board, under normal circumstances only when the drive board failure will light up red, usually often out. When the drive failure indicator light is on, it means that the drive board has produced some damage, you should turn off the power, do not operate.。

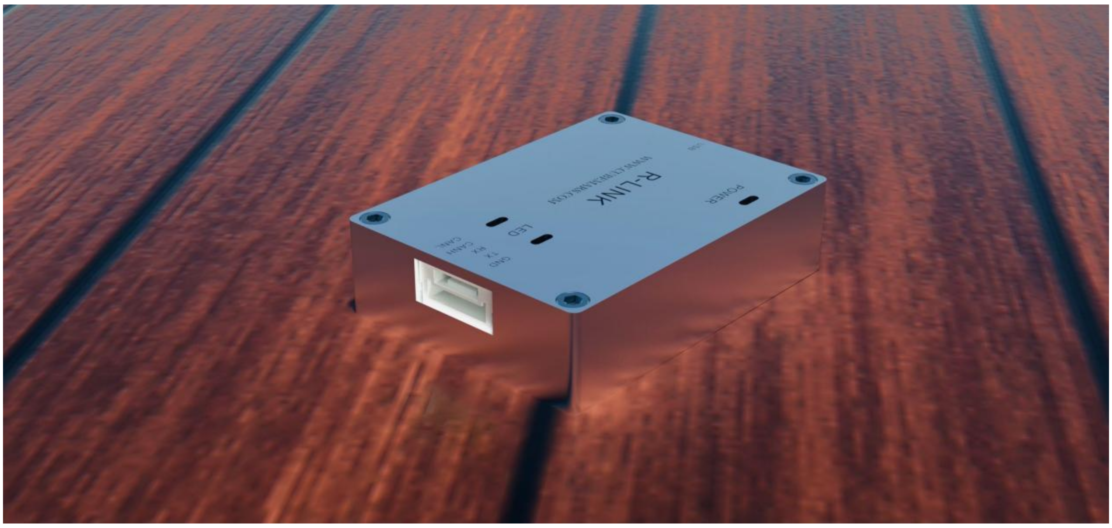
1.4 Main Accessories and Specifications

No.	Item		specifications	Quantity	Remarks
1	Serial port cable	cable	30AWG-300MM-Teflon Silver Plated Wire-Black Yellow Green	Each 1PCS	±2MM
		plug	A1257H-3P	1PCS	

			A2541H-3P	1PCS	
2	Power cable	cable	16AWG-200MM-silicone thread-Red Black	Each 1PCS	±2MM
		plug	XT30UPB-M	1PCS	
			XT30UPB-F	1PCS	
3	CAN cable	cable	30AWG-300MM-Teflon Silver Plated Wire-White Blue	Each 1PCS	±2MM
		plug	A1257H-4P	2PCS	
			A2541H-2P	1PCS	
4	Thermistor	MF51B103F3950-10K-3950		2PCS	
5	Electrolytic capacitor	120Uf-63V-10x12MM		2PCS	AK10-9 V2.0 standard
6	Power MOS	BSC026N08NS5-80V-2.6mΩ TPH2R608NH-75V-2.6mΩ		12PCS	random

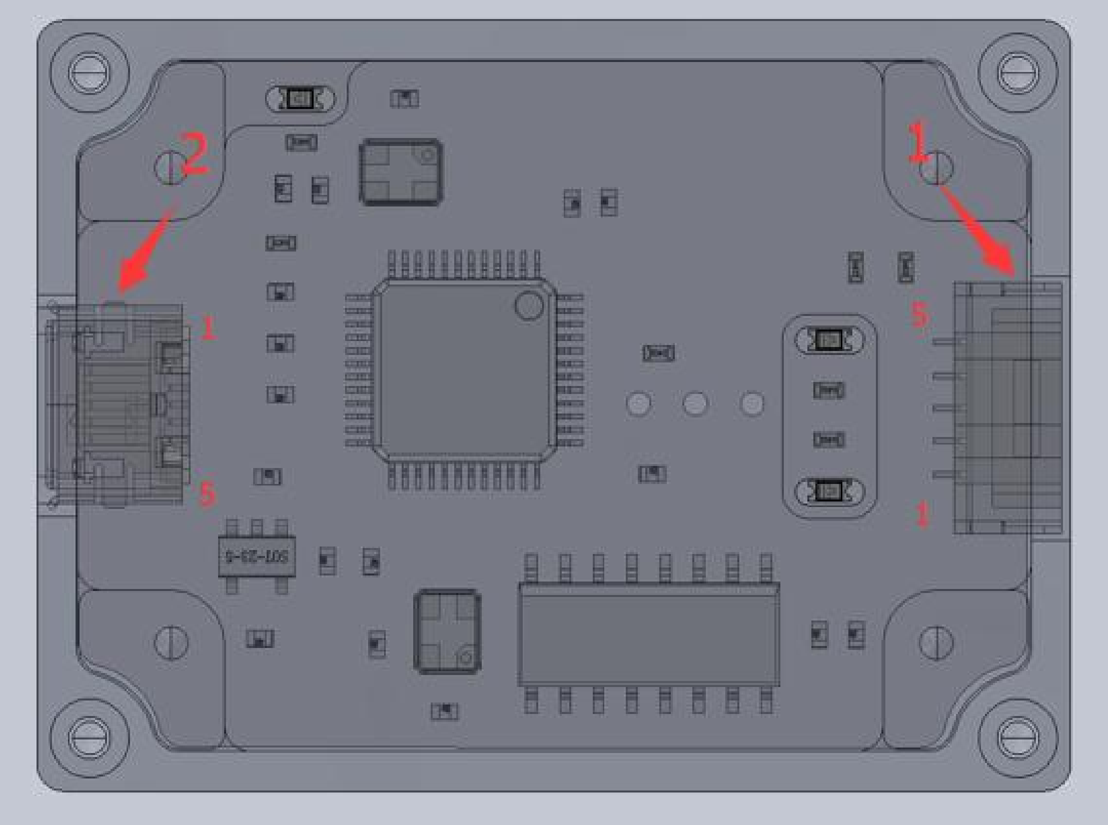
2. R-link Product Information

2.1 R-link Appearance Introduction&Product Specifications



Product Specifications	
Rated working voltage	5V
Standby power consumption	≤30mA
Size	39.2x29.2x10MM
Working environment temperature	-20℃ to 65℃
Maximum Allowable Temperature for Control Board	85℃

2.2 R-link Interface and Definition



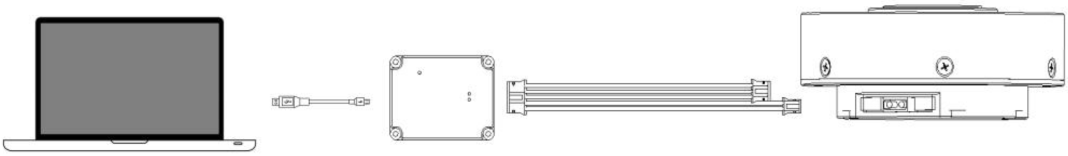
No.	Interface function	Pin	Clarification
1	Communication Interface	1	CAN communication low side（CAN_L）
		2	CAN communication high side（CAN_H）
		3	Serial Signal Input（RX）
		4	Serial Signal Output（TX）
		5	Serial Signal Ground（GND）
2	USB Interface	1	VBUS
		2	D-
		3	D+
		4	ID

No.	Interface function	Pin	Clarification
		5	GND

2.3 R-link Indicator Light Definitions

No.	Color Definition	Clarification
1	Green	Power indicator, used to indicate the R-link power situation, under normal circumstances when the power supply is plugged in will light up green, if the green light does not light up when the power supply is plugged in, please remove the power supply immediately, do not power up again.
2	Blue	Serial communication output (TX), normally off, blinking when there is data output from the R-link serial port.
3	Red	Serial communication output (RX), normally off, blinking when there is data input from the R-link serial port.

3. Driver and R-link Connection and Precautions



USB cable on R-Link ----> PC side

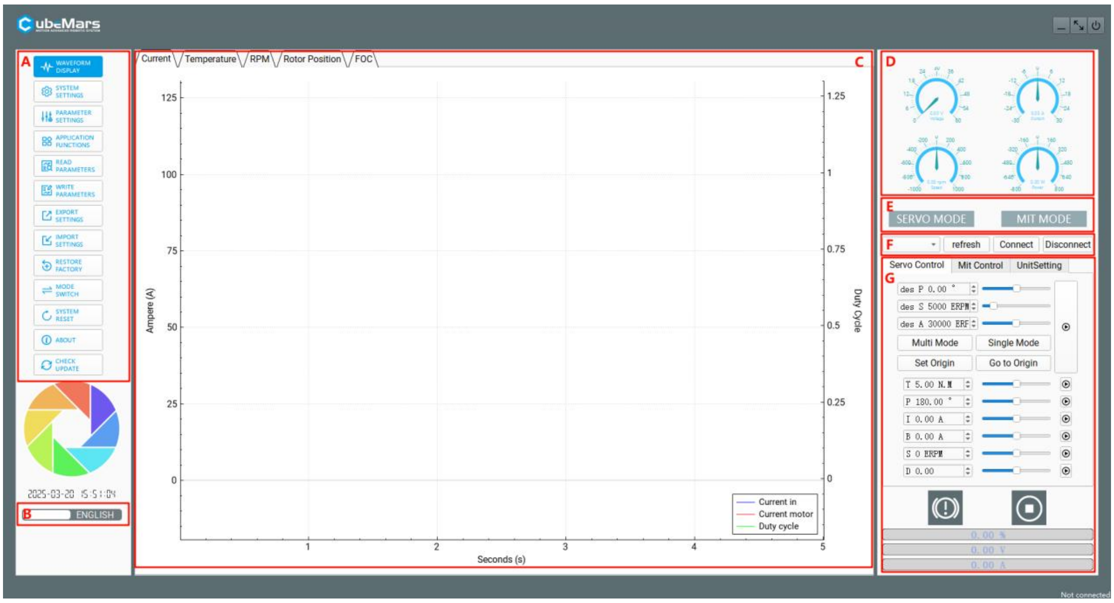
5Pin port ----> R-Link 5Pin port terminal

4Pin terminal (CAN port) ----> 4Pin port (CAN) on motor

3Pin terminal (UART port) ----> 3Pin port (UART) on motor

4.Upper Computer Instruction

4.1Upper Computer Interface and Explanation



A.Main Menu Bar

B.Chinese and English switching

C.Main page

D.Implementation data display

E.Current mode

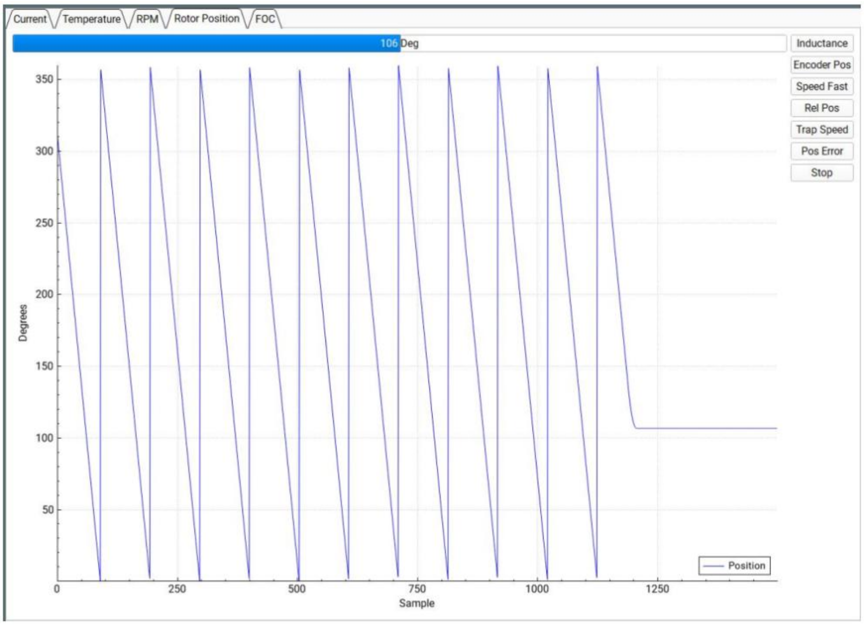
F.Serial port selection

G.Control Parameters

4.1.1 Main Menu Bar

4.1.1.1 Waveform Display

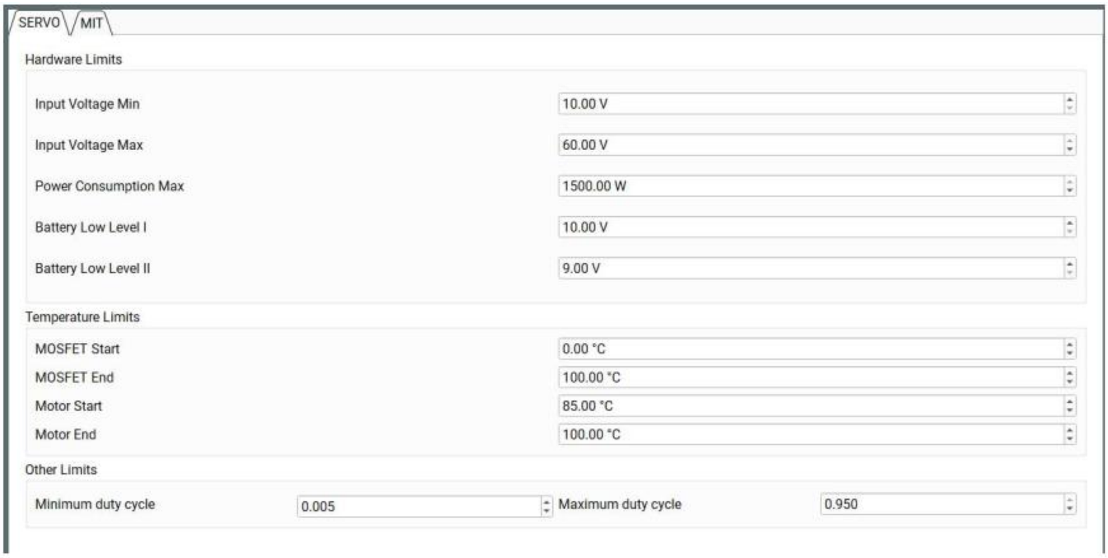
This page supports viewing real-time data feeds and plotting images. The data includes: motor current, temperature, real-time speed, internal encoder position, external encoder position, high frequency speed, rotor position, path planning, position deviation, DQ current, and more.



4.1.1.2 System Settings

This page is mainly for changing the hardware limitations of the driver board such as voltage, current, power, temperature, duty cycle, etc. It mainly serves to protect the driver board and motor.

⚠： Please be sure to use the product in strict accordance with the specified voltage, current, power and temperature. Our company will not bear any legal responsibility for any harm caused to human body or irreversible damage to the driver board and motor as a result of illegal operation of this product.



Hardware Limits

Input Voltage Min	10.00 V
Input Voltage Max	60.00 V
Power Consumption Max	1500.00 W
Battery Low Level I	10.00 V
Battery Low Level II	9.00 V

Temperature Limits

MOSFET Start	0.00 °C
MOSFET End	100.00 °C
Motor Start	85.00 °C
Motor End	100.00 °C

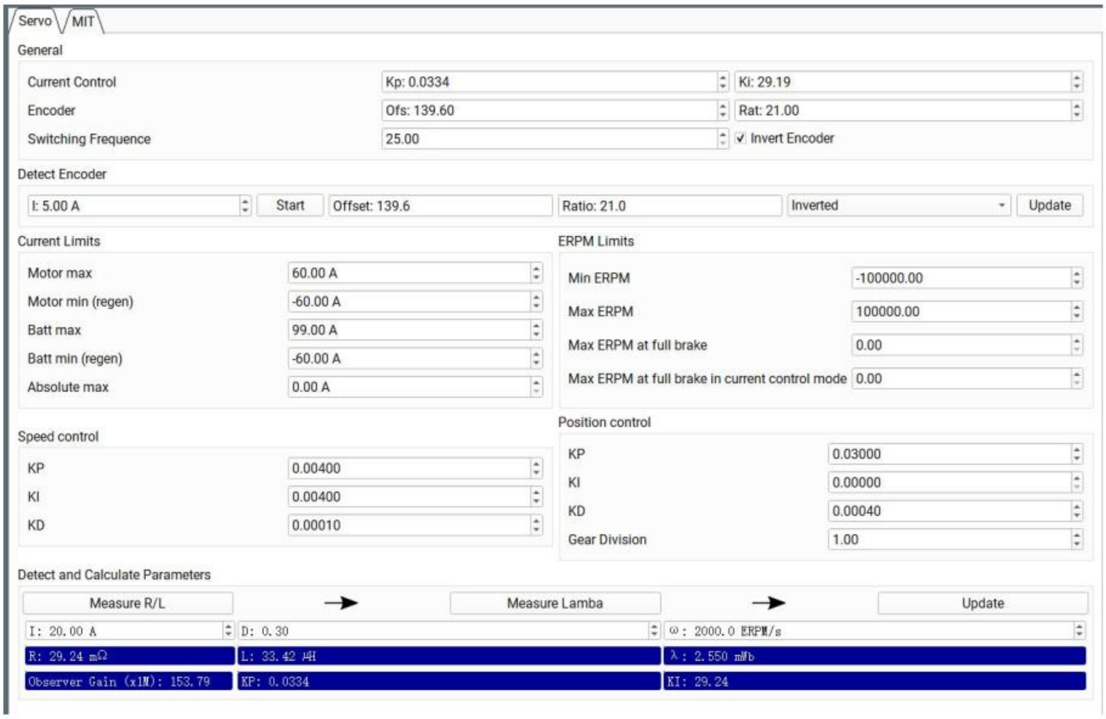
Other Limits

Minimum duty cycle	0.005	Maximum duty cycle	0.950

4.1.1.3 Parameters Setting

This page is for adjusting drive board parameters, including but not limited to current loop Kp -Ki, encoder bias, current max-min, RPM max-min, speed loop Kp-Ki-Kd, reduction ratio, and other parameters, as well as encoder calibration and motor parameterization.

⚠： Please be sure to use the product in strict accordance with the specified voltage, current, power and temperature. Our company will not bear any legal responsibility for any harm caused to human body or irreversible damage to the driver board and motor as a result of illegal operation of this product.



Servo / MIT

General

Current Control	Kp: 0.0334	Ki: 29.19
Encoder	Ofs: 139.60	Rat: 21.00
Switching Frequence	25.00	☑ Invert Encoder

Detect Encoder

I: 5.00 A | Start | Offset: 139.6 | Ratio: 21.0 | Inverted | Update

Current Limits

Motor max	60.00 A
Motor min (regen)	-60.00 A
Batt max	99.00 A
Batt min (regen)	-60.00 A
Absolute max	0.00 A

ERPM Limits

Min ERPM	-100000.00
Max ERPM	100000.00
Max ERPM at full brake	0.00
Max ERPM at full brake in current control mode	0.00

Speed control

KP	0.00400
KI	0.00400
KD	0.00010

Position control

KP	0.03000
KI	0.00000
KD	0.00040
Gear Division	1.00

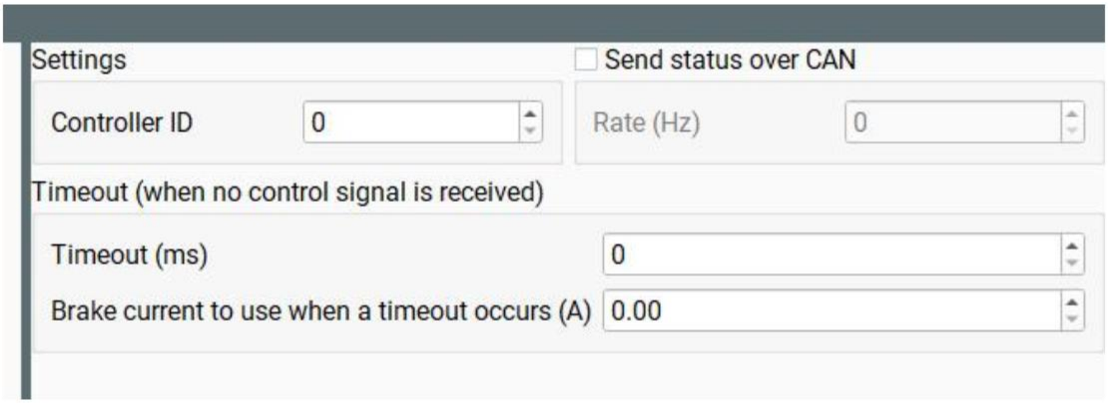
Detect and Calculate Parameters

Measure R/L	Measure Lamba	Update
I: 20.00 A	D: 0.30	ω: 2000.0 ERPM/s
R: 29.24 mΩ	L: 33.42 µH	λ: 2.550 mWb
Observer Gain (x1M): 153.79	KP: 0.0334	KI: 29.24

4.1.1.4 Application Function

APPLICATION FUNCTIONS

This page is for setting the CAN ID, CAN communication rate, and CAN sudden interrupt settings.



Settings

□ Send status over CAN

Controller ID	0
Rate (Hz)	0

Timeout (when no control signal is received)

Timeout (ms)	0
Brake current to use when a timeout occurs (A)	0.00

4.1.1.5 Read Parameters

READ PARAMETERS

Save the current parameters of the motor to the host computer，⚠： Whenever you rewrite the parameters of the motor, please make sure to click this button first, otherwise the other parameters of the motor will be out of order. If this happens, please go to the official website to download the default APP parameters of the corresponding motor, and then write the default parameters of the motor into the motor through the “Import Settings”！

4.1.1.6 Save Parameters

Save the upper computer parameters to the motor.

4.1.1.7 Export Settings

Save the upper computer parameters as two files with the suffixes “.McParams” and “.AppParams” to your computer.

The “.McParams” files are：

The “.AppParams” files are：

4.1.1.8 Import Settings

Upload the parameters from your computer with the suffixes “.McParams” and “.AppParams” to the upper computer.

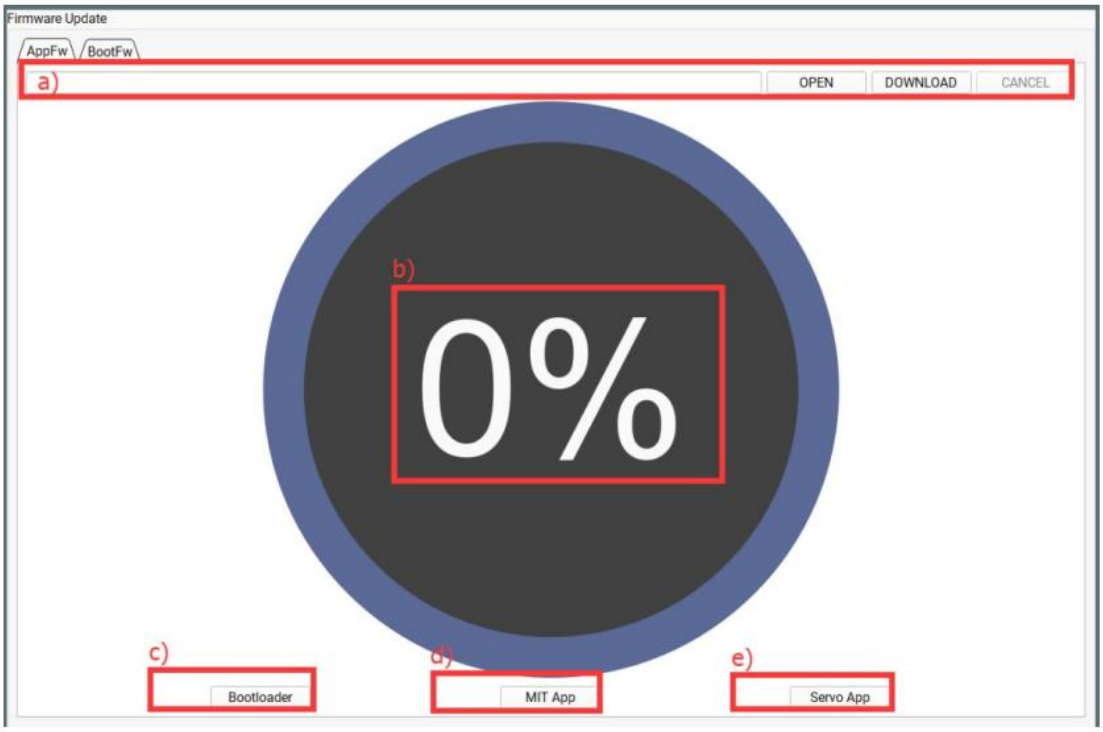
4.1.1.9 Restore Factory

This feature is not open at this time.

4.1.1.10 Mode Switch

This page is mainly for switching the control mode of the driver board, including “Bootloader”, “servo mode”, “MIT control mode”.

As well as updating the firmware of the driver board.



a). Import Firmware Area: Can import files with “.bin” extension from computer.

b). Firmware update progress bar

c). Enter Bootloader

d). Entering MIT mode

e). Enter Servo mode

4.1.1.11 System Reset

Stop and restart the motor.

4.1.1.12 About

The version number and official homepage of the current upper computer.

4.2 Driver Board Calibration

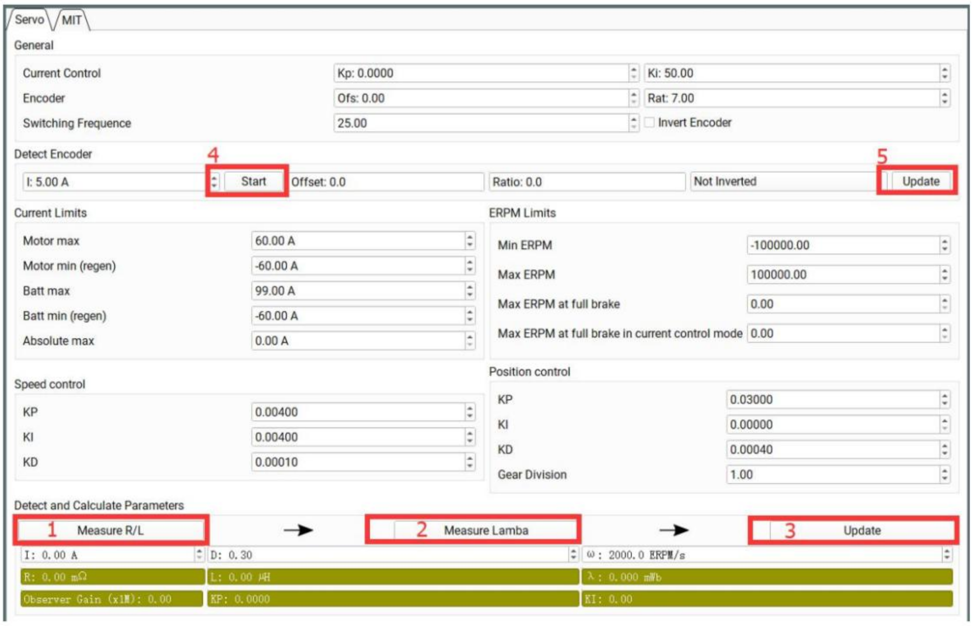
Calibration is required after you have reinstalled the driver board on the motor, changed the wiring sequence of the motor's three-phase wires, or updated the firmware. Once calibrated, the motor can be used normally.

4.2.1 Servo Mode

After confirming that the motor input power is stable, the R-LINK connection is normal, and the motor is in servo mode, after successfully connecting with the host computer, enter the system setup page, and then click “Measure R/L”, “Measure Lamba”, “Update”, “Start”, and “Update” in turn.

⚠: For your convenience, please follow the step-by-step instructions provided in the video to avoid any accidental errors:

https://www.bilibili.com/video/BV1p84y1L7wM/?spm_id_from=333.999.0.0



General

Current Control Kp: 0.0000 Ki: 50.00
Encoder Ofs: 0.00 Rat: 7.00
Switching Frequence 25.00 □ Invert Encoder

Detect Encoder

I: 5.00 A Offset: 0.0 Ratio: 0.0 Not Inverted

Current Limits

Motor max 60.00 A
Motor min (regen) -60.00 A
Batt max 99.00 A
Batt min (regen) -60.00 A
Absolute max 0.00 A

ERPM Limits

Min ERPM -100000.00
Max ERPM 100000.00
Max ERPM at full brake 0.00
Max ERPM at full brake in current control mode 0.00

Speed control

KP 0.00400
KI 0.00400
KD 0.00010

Position control

KP 0.03000
KI 0.00000
KD 0.00040
Gear Division 1.00

Detect and Calculate Parameters

I: 0.00 A D: 0.30 ω: 2000.0 ERPM/s
R: 0.00 mΩ L: 0.00 µH λ: 0.000 mWb
Observer Gain (x1M): 0.00 KP: 0.0000 KI: 0.00

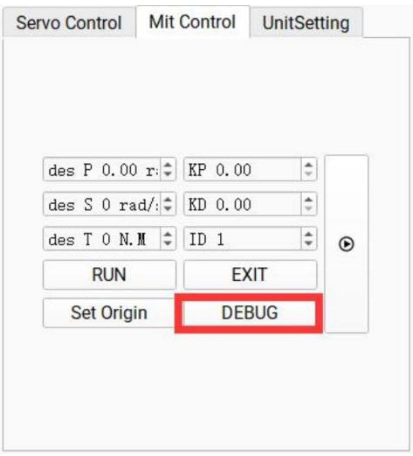
4.2.2 MIT Mode

Confirm that the motor input power is stable, the R-LINK connection is normal, and the motor is in the operation control mode, after successfully connecting with the host computer, click “Debug Mode” in the “MIT Control” interface, and then input “calibrate” in the input field. Then input “calibrate” in the input field, wait for about 30 seconds, at the same time, the output field will scroll the encoder position value in real time, until the output field prints “Encoder Electrical Offset (rad)”, the motor will restart automatically, and then the serial interface will be connected to the host computer. When “Encoder Electrical Offset (rad)” is printed on the output bar, the motor will restart automatically and the serial port will print the drive information. During calibration, the current is about 1A at 48V, after calibration, the current returns to about 0.02A.

⚠:For your convenience, please follow the step-by-step instructions provided in the video to avoid any accidental errors:

MIT（9:53）：

https://www.bilibili.com/video/BV1mY4y1o7jL/?spm_id_from=333.999.0.0&vd_source=c119356dd515eff93cf5c0b8d51671a3



des P 0.00 r:

KP 0.00

des S 0 rad/:

KD 0.00

des T 0 N.M

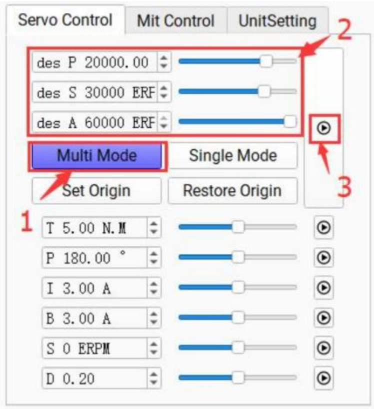
ID 1

4.3 Control Demo

4.3.1 Servo Mode

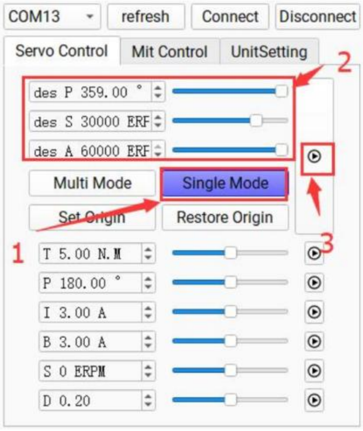
4.3.1.1 Multi-loop position velocity mode

Confirm that the input power of the motor is stable, the R-LINK connection is normal, and the motor is in servo mode, after successfully connecting with the host computer, click “Multi-loop Mode” in the interface of “Servo Control”, and input the desired position (at this time, the position is ±100 loops, that is, -36000° -36000°), the desired speed and acceleration. -36000° -36000°), desired speed and acceleration, the motor will move according to the desired speed until the desired position.



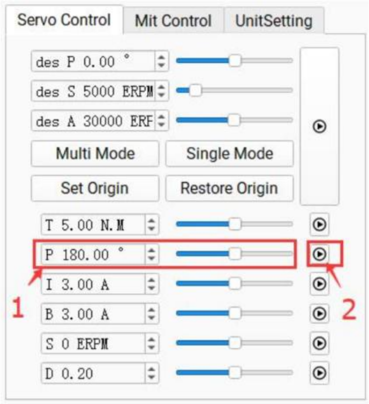
4.3.1.2 Single-loop position-velocity mode

Ensure that the motor's input power is stable, the R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the host computer, navigate to the "Servo Control" interface and click on "Single Loop Mode." Enter the desired position (limited to one full rotation, i.e., 0° – 359°), desired speed, and acceleration. The motor will then move at the specified speed until it reaches the desired position.



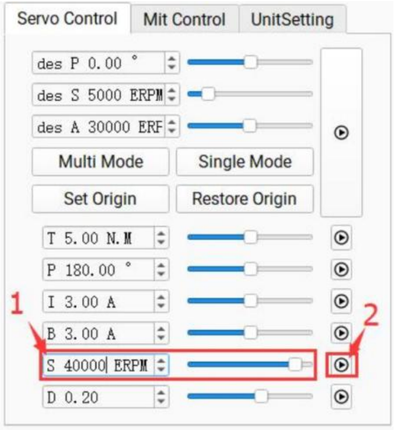
4.3.1.3 Position Mode

Ensure that the motor's input power is stable, the R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the host computer, enter the desired position in the "Servo Control" interface. The motor will then reach the desired position at its maximum speed.



4.3.1.4 Velocity Mode

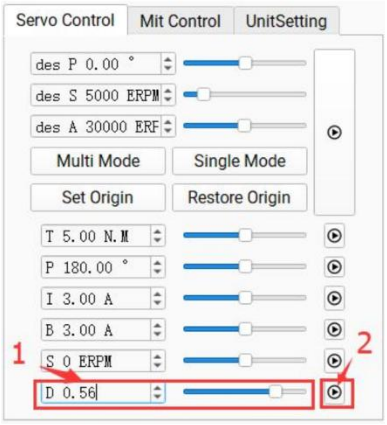
Ensure that the motor's input power is stable, the R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, enter the desired speed (±50000 ERPM) in the "Servo Control" interface. The motor will then operate at the specified speed.



des P 0.00 °
des S 5000 ERPM
des A 30000 ERF
T 5.00 N.M
P 180.00 °
I 3.00 A
B 3.00 A
S 40000 ERPM
D 0.20

4.3.1.5 Duty Cycle Mode

Ensure the motor's input power is stable, the R-LINK connection is normal, and the motor is in servo mode. After connecting to the upper computer, enter the desired duty cycle (default 0.005 – 0.95) in the "Servo Control" interface. The motor will then operate at the specified duty cycle.

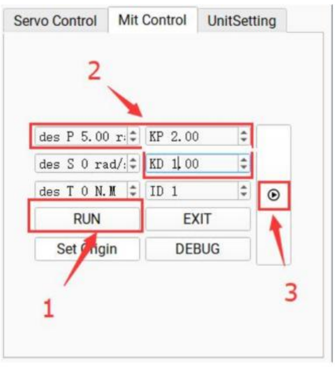


des P 0.00 °
des S 5000 ERPM
des A 30000 ERF
T 5.00 N.M
P 180.00 °
I 3.00 A
B 3.00 A
S 0 ERPM
D 0.56

4.3.2 MIT Mode

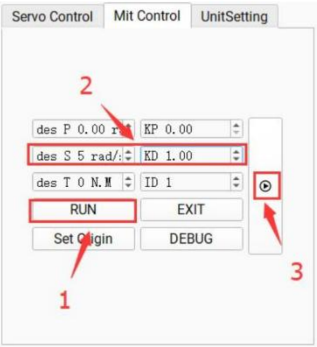
4.3.2.1 Position Mode

Ensure the motor's input power is stable, the R-LINK connection is normal, and the motor is in motion control mode. After successfully connecting to the host computer, enter the corresponding "CAN ID" in the "MIT Control" interface and click "Enable Control" to activate the motor mode. Then, enter the desired position along with KP and KD values. The motor will perform position control with a default speed of 12,000 ERPM and an acceleration of 40,000 ERPM.



4.3.2.2 Velocity Mode

Ensure the motor's input power is stable, the R-LINK connection is normal, and the motor is in motion control mode. After successfully connecting to the host computer, enter the corresponding "CAN ID" in the "MIT Control" interface and click "Enable Control" to activate the motor mode. Then, enter the desired speed and KD value. The motor will then operate in speed control mode.

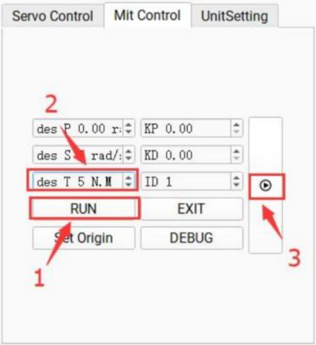


Servo Control | Mit Control | UnitSetting

des P 0.00 r | KP 0.00
des S 5 rad/ | KD 1.00
des T 0 N.M | ID 1

4.3.2.3 Torque Mode

Ensure the motor's input power is stable, the R-LINK connection is normal, and the motor is in motion control mode. After successfully connecting to the host computer, enter the corresponding "CAN ID" in the "MIT Control" interface and click "Enable Control" to activate the motor mode. Then, enter the desired torque. The motor will then operate in torque control mode.

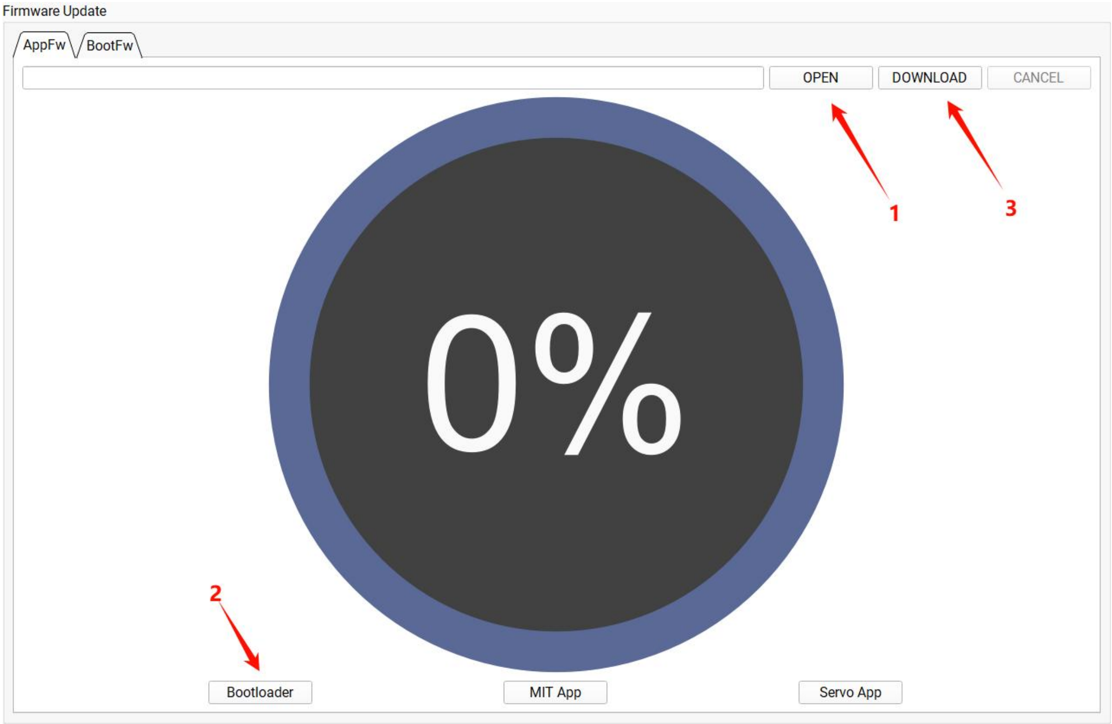


Servo Control | Mit Control | UnitSetting

des P 0.00 r | KP 0.00
des S rad/ | KD 0.00
des T 5 N.M | ID 1

4.4 Firmware Update

1. Click **"Open File"**, select the firmware file, which typically has the **".BIN"** extension.
2. Click **"Bootloader"**.
3. Click **"Download"** and wait for the progress bar to reach **100%**.
4. Restart the power supply to complete the firmware update.



⚠:For your convenience, please follow the step-by-step instructions provided in the video carefully to avoid any accidental errors:

Firmware Installation and Calibration:

https://www.bilibili.com/video/BV1p84y1L7wM/?spm_id_from=333.999.0.0

Please note that in this video demonstration, only the servo mode firmware has been uploaded. If your firmware contains both servo and MIT firmware, please upload both firmware separately to ensure proper use of both modes.

In addition, if the firmware upload progress bar appears to be stuck and not moving, please follow the steps below:

Step 1: Ensure stable power supply and correct connection.

Step 2: Enter the mode switching interface, click the “Open” button and select the firmware of your motor.

Step 3: Continuously click on the Jump Guide button. At the same time, please use your other hand to turn off the power, and then turn on the power again.

After performing these steps, you can see the progress bar start to move. Once the firmware has been reinstalled, please perform the default parameter import and calibration, and the motor will resume normal operation.

5. Driver Board Communication Protocol and Description

5.1Servo Mode Control Mode and Descriptions

Servo mode includes six control modes:

1. **Duty Cycle Mode**: Sets a specified duty cycle voltage for the motor, similar to square wave drive control.
2. **Current Loop Mode**: Sets a specified Iq current for the motor. Since the output torque is Torque = Iq × KT, this mode can be used as a torque control loop.
3. **Current Brake Mode**: Applies a specified braking current to hold the motor in its current position (monitor motor temperature during use).
4. **Velocity Mode**: Sets a specified operating speed for the motor.
5. **Position Mode**: Sets a specified target position, and the motor will move to this position at maximum speed.
6. **Position-Velocity Loop Mode**: Sets a specified position, speed, and acceleration, allowing the motor to move to the target position with the given acceleration and speed constraints.

The servo motor protocol is the can protocol, which uses the extended frame format as follows

Can ID bits	[28]-[8]	[7]-[0]
Field name	Control mode	Source node ID

Control mode has {0,1,2,3,4,5,6} 7 characteristic values corresponding to 7 control modes.

Duty cycle mode: 0

Current loop mode: 1

Current brake mode: 2

RPM mode: 3

Position mode: 4

Set Origin Mode: 5

Position velocity loop mode: 6

Examples of various modes of motor control are provided below

The following is a list of example library functions and macro definitions.

```
typedef enum {
    CAN_PACKET_SET_DUTY = 0,            //Duty Cycle Mode
    CAN_PACKET_SET_CURRENT,             //Current Loop Mode
    CAN_PACKET_SET_CURRENT_BRAKE,       // Current Brake Mode
    CAN_PACKET_SET_RPM,                 // RPM Mode
    CAN_PACKET_SET_POS,                 // Position Mode
    CAN_PACKET_SET_ORIGIN_HERE,         //Set Origin Mode
    CAN_PACKET_SET_POS_SPD,             //Position-Velocity Loop Mode
    CAN_PACKET_SET_mit=8,            //MIT Mode
} CAN_PACKET_ID;
```

```
void comm_can_transmit_eid(uint32_t id, const uint8_t *data, uint8_t len) {
        uint8_t i=0;
        CanTxMsg TxMessage;
        if (len > 8) {
                len = 8;
        }
        TxMessage.StdId = 0;
        TxMessage.IDE = CAN_ID_EXT;
        TxMessage.ExtId = id;
        TxMessage.RTR = CAN_RTR_DATA;
        TxMessage.DLC = len;
        for(i=0;i<len;i++)
                TxMessage.Data[i]=data[i];
        CAN_Transmit(CANx, &TxMessage);    //CAN send TxMessage data
}

void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index) {
        buffer[(*index)++] = number >> 24;
        buffer[(*index)++] = number >> 16;
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}

void buffer_append_int16(uint8_t* buffer, int16_t number, int16_t *index) {
        buffer[(*index)++] = number >> 8;
        buffer[(*index)++] = number;
}
```

5.1.1 Duty Cycle Mode

Duty Cycle Mode transmit data definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
Range	0~0xff	0~0xff	0~0xff	0~0xff
corresponding variable	Duty cycle 25-32bit	Duty cycle 17-24bit	Duty cycle 9-16bit	Duty cycle 1-8bit

```
void comm_can_set_duty(uint8_t controller_id, float duty) {
        int32_t send_index = 0;
        uint8_t buffer[4];
        buffer_append_int32(buffer, (int32_t)(duty * 100000.0), &send_index);
        comm_can_transmit_eid(controller_id |
                        ((uint32_t)CAN_PACKET_SET_DUTY << 8), buffer, send_index);
}
```

5.1.2 Current Loop Mode

Current loop mode transmit data definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
Range	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	Current 25-32bit	Current 17-24bit	Current 9-16bit	Current 1-8bit

Where the current value is of type int32 and the value -60000-60000 represents -60-60A.

Current Loop Mode Send Routine

```
void comm_can_set_current(uint8_t controller_id, float current) {
        int32_t send_index = 0;
        uint8_t buffer[4];
        buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);
        comm_can_transmit_eid(controller_id |
                                ((uint32_t)CAN_PACKET_SET_CURRENT << 8), buffer, send_index);
}
```

5.1.3 Current Brake Mode

Current Brake Mode Send Data Definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
Range	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	Brake current 25-32bit	Brake current 17-24bit	Brake current 9-16bit	Brake current 1-8bit

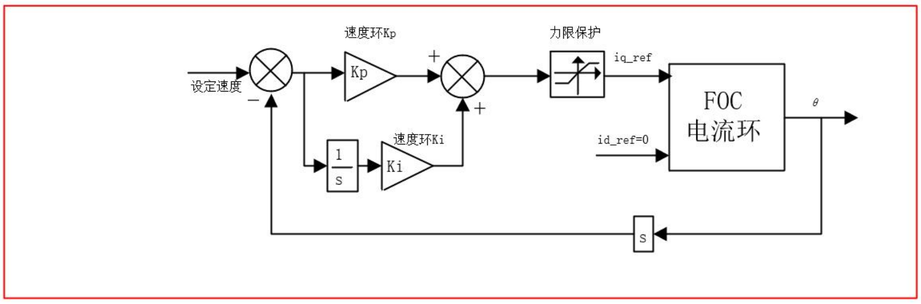
Where the brake current value is of type int32 and the value 0-60000 represents 0-60A.

Current Brake Mode Send Routine

```
void comm_can_set_cb(uint8_t controller_id, float current) {
        int32_t send_index = 0;
        uint8_t buffer[4];

        buffer_append_int32(buffer, (int32_t)(current * 1000.0), &send_index);

        comm_can_transmit_eid(controller_id |

                        ((uint32_t)CAN_PACKET_SET_CURRENT_BRAKE << 8), buffer, send_index);

}
```

5.1.4 Velocity-loop Mode

Velocity Loop Abbreviated Control Block Diagram



Velocity loop mode transmit data definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
Range	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	Velocity 25-32bit	Velocity 17-24bit	Velocity 9-16bit	Velocity 1-8bit

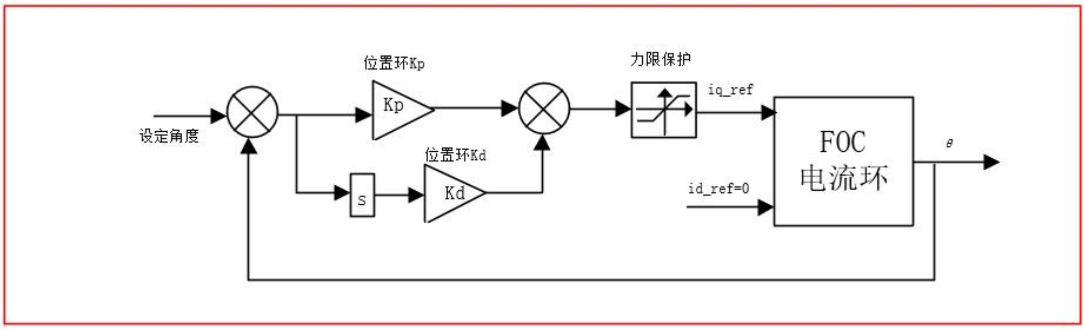
Where the speed value is of type int32 and the range -100000-100000 represents -100000-100000 ERPM.

Velocity Loop send routine

```
void comm_can_set_rpm(uint8_t controller_id, float rpm) {

        int32_t send_index = 0;

        uint8_t buffer[4];

        buffer_append_int32(buffer, (int32_t)rpm, &send_index);

        comm_can_transmit_eid(controller_id |

                        ((uint32_t)CAN_PACKET_SET_RPM << 8), buffer, send_index);
}
```

5.1.5 Position loop Mode

Position Loop Abbreviated Control Block Diagram



Position Loop Mode Transmit Data Definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
Range	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	Position 25-32bit	Position 17-24bit	Position 9-16bit	Position 1-8bit

where position is of type int32 and the range -360000000-360000000 represents position -36000° ~36000° ;

Position loop send routine

```
void comm_can_set_pos(uint8_t controller_id, float pos) {
    int32_t send_index = 0;
    uint8_t buffer[4];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    comm_can_transmit_eid(controller_id |
                ((uint32_t)CAN_PACKET_SET_POS << 8), buffer, send_index);
}
```

5.1.6 Set Origin Mode

Data bit	Data[0]
Range	0~0x02
Corresponding Variable	Set command

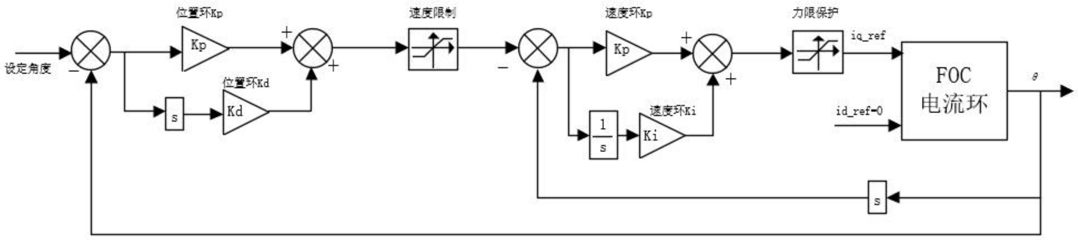
Among them, the setup instruction is of uint8_t type, with 0 representing the setting of the temporary origin (power failure elimination) and 1 representing the setting of the permanent zero point (dual encoder models only).

Position loop send routines

```
void comm_can_set_origin(uint8_t controller_id, uint8_t set_origin_mode) {
        int32_t send_index = 1;
        uint8_t buffer;
        buffer=set_origin_mode;
        comm_can_transmit_eid(controller_id |
                        ((uint32_t) CAN_PACKET_SET_ORIGIN_HERE << 8), &buffer, send_index);
}
```

5.1.7 Position-velocity Mode

Abbreviated block diagram of position-velocity loop



Position Velocity Loop Mode Transmit Data Definition

Data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Range	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	Position 25-32bit	Position 17-24bit	Position 9-16bit	Position 1-8bit	Speed high 8 bits	Speed low 8 bits	Acceleration high 8 bits	Acceleration low 8 bits

The position is of type int32, with a range of -360000000 to 360000000, corresponding to a position range of -36000° to 36000° .
The speed is of type int16, with a range of -32768 to 32767, corresponding to an electrical speed range of -327680 to 327680 ERPM.
The acceleration is of type int16, with a range of 0 to 32767, corresponding to 0 to 327670, where 1 unit equals 10 electrical RPM/s^2 .

```
void comm_can_set_pos_spd(uint8_t controller_id, float pos,int16_t spd, int16_t RPA ) {
    int32_t send_index = 0;
    int16_t send_index1 = 4;
    uint8_t buffer[8];
    buffer_append_int32(buffer, (int32_t)(pos * 10000.0), &send_index);
    buffer_append_int16(buffer,spd/10.0, & send_index1);
    buffer_append_int16(buffer,RPA/10.0, & send_index1);
    comm_can_transmit_eid(controller_id |
            ((uint32_t)CAN_PACKET_SET_POS_SPD << 8), buffer, send_index1);
}
```

5.2 Servo Mode Motor Message Protocol

5.2.1 Servo mode CAN upload message protocol

The motor CAN message in servo mode uses the timed upload mode, the upload frequency can be set to 1~500HZ, and the upload byte is 8 bytes.

Servo Mode Timing Feedback Data Definition

Identifier: Function ID + Motor ID Frame Type: Extended Frame

Frame format: DATA DLC: 8 bytes

Function ID：

Frame 0x09 represents that the motor is in jump start state;

Frame 0x2C is the start frame sent by the motor to enter servo mode and the reply message is fixed to 0xFA 0xFB 0xFC 0xFD;

Frame 0x29 represents the servo mode real-time feedback of the current state of the motor and its data bits.

Data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Range	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding Variable	position high 8 bits	Position low 8 bits	Speed high 8 bits	Speed low 8 bits	Current high 8 bits	Current low 8 bits	Motor temperature	error code

The position is of type int16, with a range of -32000 to 32000, representing a position range of -3200° to 3200° .

The speed is of type int16, with a range of -32000 to 32000, representing an electrical speed range of -320000 to 320000 ERPM.

The current is of type int16, with a range of -6000 to 6000, representing a current range of -60 to 60A.

The temperature is of type int8, with a range of -20 to 127, representing the drive board temperature range of -20° C to 127° C.

The error code is of type uint8, where:

- 0 indicates no fault,
- 1 indicates motor over-temperature fault,
- 2 indicates over-current fault,
- 3 indicates over-voltage fault,
- 4 indicates under-voltage fault,
- 5 indicates encoder fault,
- 6 indicates MOSFET over-temperature fault,
- 7 indicates motor stall.

The following is an example of message acceptance

```
void motor_receive(float* motor_pos,float*

motor_spd,float* motor_cur,int8_t* motor_temp,int8_t* motor_error,CanRxMsg *RxMessage)

  {

       int16_t pos_int =((RxMessage)->Data[0] << 8 | (RxMessage)->Data[1]);

       int16_t spd_int = ((RxMessage)->Data[2] << 8 | (RxMessage)->Data[3]);

       int16_t cur_int = ((RxMessage)->Data[4] << 8 | (RxMessage)->Data[5]);

      *motor_pos= (float)( pos_int * 0.1f); //motor position

      *motor_spd= (float)( spd_int * 10.0f);//motor velocity

      *motor_cur= (float) ( cur_int * 0.01f);//motor current

      *motor_temp= (RxMessage)->Data[6] ;//motor temperature

      *motor_error= (RxMessage)->Data[7] ;//motor error code

  }
```

5.2.2 Servo Mode Serial Message Protocol

The protocol for sending and receiving messages from the servo mode serial port is as follows

Header	Data length	Data frame	Data bit	Checksum bit		Tail
0x02	Without header, footer and parity bits	See data frame definition		High 8 bits	Low 8 bits	0x03

Data frame definition:

```
typedef enum {
            COMM_FW_VERSION = 0,                // Firmware version
            COMM_JUMP_TO_BOOTLOADER,             // Jump to bootloader
            COMM_ERASE_NEW_APP,                  // Erase new application
            COMM_WRITE_NEW_APP_DATA,             // Write new application data
            COMM_GET_VALUES,                    // Get motor operating parameters
            COMM_SET_DUTY,                      // Run motor in duty cycle mode
            COMM_SET_CURRENT,                   // Run motor in current loop mode
            COMM_SET_CURRENT_BRAKE,             // Run motor in current brake mode
            COMM_SET_RPM,                       // Run motor in speed loop mode
            COMM_SET_POS,                       // Run motor in position loop mode
            COMM_SET_HANDBRAKE,                  // Run motor in handbrake current loop mode
            COMM_SET_DETECT,                    // Motor real-time feedback of current position
command
            COMM_ROTOR_POSITION = 22,          // Motor feedback current position
            COMM_GET_VALUES_SETUP = 50,        // Motor single or multiple parameter fetch
command
     COMM_SET_POS_SPD = 91,            // Run motor in position-speed loop mode
     COMM_SET_POS_MULTI = 92,          // Set motor movement to single-turn mode
```

```
    COMM_SET_POS_SINGLE = 93,        // Set motor movement to multi-turn mode, range ±100 turns
            COMM_SET_POS_UNLIMITED = 94,    // Reserved
    COMM_SET_POS_ORIGIN = 95,        // Set motor origin
} COMM_PACKET_ID;
```

Serial checksum：

```
const unsigned short crc16_tab[] = { 0x0000, 0x1021, 0x2042, 0x3063, 0x4084,
    0x50a5, 0x60c6, 0x70e7, 0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad,
    0xe1ce, 0xf1ef, 0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7,
    0x62d6, 0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485, 0xa56a,
    0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d, 0x3653, 0x2672,
    0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4, 0xb75b, 0xa77a, 0x9719,
    0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc, 0x48c4, 0x58e5, 0x6886, 0x78a7,
    0x0840, 0x1861, 0x2802, 0x3823, 0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948,
    0x9969, 0xa90a, 0xb92b, 0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50,
    0x3a33, 0x2a12, 0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b,
    0xab1a, 0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49, 0x7e97,
    0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70, 0xff9f, 0xefbe,
    0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78, 0x9188, 0x81a9, 0xb1ca,
    0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f, 0x1080, 0x00a1, 0x30c2, 0x20e3,
    0x5004, 0x4025, 0x7046, 0x6067, 0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d,
         0xd31c, 0xe37f, 0xf35e, 0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214,
         0x6277, 0x7256, 0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c,
         0xc50d, 0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
         0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c, 0x26d3,
         0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634, 0xd94c, 0xc96d,
         0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab, 0x5844, 0x4865, 0x7806,
         0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3, 0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e,
         0x8bf9, 0x9bd8, 0xabbb, 0xbb9a, 0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1,
         0x1ad0, 0x2ab3, 0x3a92, 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b,
         0x9de8, 0x8dc9, 0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0,
         0x0cc1, 0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
         0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0};


unsigned short crc16(unsigned char *buf, unsigned int len) {
          unsigned int i;
          unsigned short cksum = 0;
          for (i = 0; i < len; i++) {
           cksum = crc16_tab[(((cksum >> 8) ^ *buf++) & 0xFF)] ^ (cksum << 8);
          }
          return cksum;
}
```

5.2.2.1 Get Parameters command

1、Get motor parameters

Command：

02 01 04 40 84 03

Respond：

02 49 04 01 66 FC D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF F3 00 F6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 16 D7 AD 00 0A 6F 19 40 7E 00 00 00 00 00 00 00 00 00 00 00 00 00 04 4D 53 03 02 05 16 00 1A B6 03 C9 B5 03

Clarification：

Get the motor parameters and feedback the motor status once after the motor receives it

parameter parsing：

02 (Frame Header) + 49 (Data Length) + 04 (Data Frame) + MOS Temperature (2 bytes) + Motor Temperature (2 bytes) + Output Current (4 bytes) + Input Current (4 bytes) + Id Current (4 bytes) + Iq Current (4 bytes) + Motor Throttle Value (2 bytes) + Motor Speed (4 bytes) + Input Voltage (2 bytes) + Reserved Value (24 bytes) + Motor Status Code (1 byte) + Motor Outer Loop Position Value (4 bytes) + Motor Control ID (1 byte) + Temperature Reserved Value (6 bytes) + Vd Voltage Value (4 bytes) + Vq Voltage Value (4 bytes) + CRC + 03 (Frame Tail).

The parameter conversion formulas for values sent by the motor are as follows:

MOS Temperature = (float)buffer_get_int16(data, &ind) / 10.0

Motor Temperature = (float)buffer_get_int16(data, &ind) / 10.0

Output Current = (float)buffer_get_int32(data, &ind) / 100.0

Input Current = (float)buffer_get_int32(data, &ind) / 100.0

Id Current = (float)buffer_get_int32(data, &ind) / 100.0

Iq Current = (float)buffer_get_int32(data, &ind) / 100.0

Motor Throttle Value = (float)buffer_get_int16(data, &ind) / 1000.0

Motor Speed = (float)buffer_get_int32(data, &ind)

Input Voltage = (float)buffer_get_int16(data, &ind) / 10.0

Motor Outer Loop Position = (float)buffer_get_int32(data, &ind) / 1000.0

Motor ID = data

Motor Vd Voltage = (float)buffer_get_int32(data, &ind) / 1000.0

Motor Vq Voltage = (float)buffer_get_int32(data, &ind) / 1000.0

2、Get the motor position

Command：

02 02 0B 04 9C 7E 03

Respond：

02 05 16 00 1A B6 64 D5 F4 03

Clarification：

Motor receives and sends current position every 10ms

Parameter parsing：

Pos=(float)buffer_get_int32(data, &ind) / 10000.0

3、Get single or multiple parameters of the motor

Command：

02 05 32 00 00 00 01 58 4C 03 //get MOS temperature

Respond：

02 03 32 00 81 2A 6C 03

Clarification：

To retrieve a single or multiple motor parameters, the command data section (4 bytes) determines which parameters are fetched based on the bit. When the corresponding bit is 1, the motor will return the corresponding motor parameter; when the bit is 0, that field is excluded. (The sequence of motor parameters is fetched starting from the first bit.)

The motor parameters corresponding to each bit are shown in the table below:

Data bit	32-19byte	18byte	17byte	16byte	10-15byte	9byte	8byte	7byte
clarification	Reserved value	Motor ID 1byte	Motor position 4byte	Motor Error Sign 1byte	Reserved value	Input voltage2byte	RPM4 byte	Duty cycle 2byte
Data bit	**6byte**	**5byte**	**4byte**	**3byte**	**2byte**	**1byte**		
clarification	Iq current 4byte	Id current 4byte	Input current 4byte	Output current 4byte	Motor temp 2byte	MOS temp 2byte		

Parameter parsing：

The conversion formulas for some parameters sent by the motor are as follows:

MOS Temperature = (float)buffer_get_int16(data, &ind) / 10.0

Motor Temperature = (float)buffer_get_int16(data, &ind) / 10.0

Output Current = (float)buffer_get_int32(data, &ind) / 100.0

Input Current = (float)buffer_get_int32(data, &ind) / 100.0

Motor Throttle Value = (float)buffer_get_int16(data, &ind) / 1000.0

Motor Speed = (float)buffer_get_int32(data, &ind)

Input Voltage = (float)buffer_get_int16(data, &ind) / 10.0

Motor Position = (float)buffer_get_int32(data, &ind) / 1000000.0

Motor ID = data

Motor error status codes

```
typedef enum {
        FAULT_CODE_NONE = 0,
        FAULT_CODE_OVER_VOLTAGE,    // Over-voltage
        FAULT_CODE_UNDER_VOLTAGE, // Under-voltage
        FAULT_CODE_DRV,                    // Driver fault
        FAULT_CODE_ABS_OVER_CURRENT, // Motor over-current
        FAULT_CODE_OVER_TEMP_FET, // MOS over-temperature
        FAULT_CODE_OVER_TEMP_MOTOR, // Motor over-temperature
        FAULT_CODE_GATE_DRIVER_OVER_VOLTAGE, // Driver over-voltage
        FAULT_CODE_GATE_DRIVER_UNDER_VOLTAGE, // Driver under-voltage
        FAULT_CODE_MCU_UNDER_VOLTAGE, // MCU under-voltage
        FAULT_CODE_BOOTING_FROM_WATCHDOG_RESET, // Under-voltage
        FAULT_CODE_ENCODER_SPI, // SPI encoder fault
        FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE, // Encoder out of range
        FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE, // Encoder out of range
        FAULT_CODE_FLASH_CORRUPTION, // FLASH fault
        FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_1, // Current sampling channel 1 fault
        FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_2, // Current sampling channel 2 fault
        FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_3, // Current sampling channel 3 fault
        FAULT_CODE_UNBALANCED_CURRENTS, // Current imbalance
} mc_fault_code;
```

5.2.2.2 Control Command

1、Duty cycle send mode

Command：

02 05 05 00 00 4E 20 29 F6 03 // 0.20 duty cycle

02 05 05 FF FF B1 E0 77 85 03 // -0.20 duty cycle

Parameter parsing：

Duty=(float)buffer_get_int32(data, &ind) / 100000.0) //The value is to receive 4 bits of data/100000.0

2、Current loop send mode

Command：

02 05 06 00 00 13 88 8B 25 03 // 5 A IQ current

02 05 06 FF FF EC 78 E3 05 03 // -5 A IQ current

Parameter parsing：

Current=(float)buffer_get_int32(data, &ind) / 1000.0 //The value is to receive 4 bits of data/1000.0

3、Brake current send mode

Command：

02 05 07 00 00 13 88 21 74 03 // 5A brake current

02 05 07 FF FF EC 78 49 54 03 // - 5A brake current

Parameter parsing：

I_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //The value is to receive 4 bits of data/1000.0

4、Velocity loop send mode

Command：

02 05 08 00 00 03 E8 2B 58 03 // 1000 ERPM ERPM

02 05 08 FF FF FC 18 43 78 03 // - 1000 ERPM ERPM

Parameter parsing：

Speed=(float)buffer_get_int32(data, &ind) //The value is to receive 4 bits of data

5、Position loop send mode

Command：

02 05 09 0A BA 95 00 1E F7 03 //Motor rotates to 180 degrees

02 05 09 05 5D 4A 80 7B 29 03 //Motor rotates to 90 degrees

Parameter parsing：

Pos=(float)buffer_get_int32(data, &ind) / 1000000.0 //The value is to receive 4 bits of data/1000000.0

6、Hand brake current send mode

Command：

02 05 0A 00 00 13 88 00 0E 03 //5A HB current ERPM

02 05 0A FF FF EC 78 68 2E 03 //5A HB current ERPM

Parameter parsing：

HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //The value is to receive 4 bits of data/1000.0

7、Position-velocity loop send mode

Command：

02 0D 5B 00 02 BF 20 00 00 13 88 00 00 75 30 A5 AC 03

Parameter parsing：

position+ velocity + acceleration

180degree 5000ERPM acceleration 30000/S

Pos=(float)buffer_get_int32(data, &ind) / 1000.0) //The value is to receive 4 bits of data/1000.0

Speed=(float)buffer_get_int32(data, &ind) //The value is to receive 4 bits of data

Acc_Speed=(float)buffer_get_int32(data, &ind) //The value is to receive 4 bits of data

8、Set multi loop mode

Command：

02 05 5C 00 00 00 00 9E 19 03

Clarification：

Motor position ring in multi-loop operation mode ±100 loops.

9、Set single loop mode

Command：

02 05 5D 00 00 00 00 34 48 03

Clarification：

Motor position ring for single-loop operation mode 0-360 degrees

10、Set the current position as 0

Command：

02 02 5F 01 0E A0 03

Clarification：

Set the current position ring of the motor as the zero reference point

11、Shortest distance to zero

Command：

02 05 65 00 00 00 00 3A 8B 03

Clarification：

Returning the motor to relative zero for the shortest distance.

data conversion：

```
int16_t buffer_get_int16(uint8_t* buffer, int32_t *index)
{
        Int16_t res = ((uint16_t)buffer[*index])<<8|
                                  ((uint16_t)buffer[*index+1]);
        return res;
}

uint16_t buffer_get_int16(uint8_t* buffer, int32_t *index)
{
        Uint16_t res = ((uint16_t)buffer[*index])<<8|
                                  ((uint16_t)buffer[*index+1]);
        return res;
}

int32_t buffer_get_int32(uint8_t* buffer, int32_t *index)
```

```
{
        int32_t res = ((uint32_t)buffer[*index])<<24|
                        ((uint32_t)buffer[*index+1])<<16|
                        ((uint32_t)buffer[*index+2])<<8|
                        ((uint32_t)buffer[*index+3]);
        return res;
}

uint32_t buffer_get_uint32(uint8_t* buffer, int32_t *index)
{
        uint32_t res = ((uint32_t)buffer[*index])<<24|
                        ((uint32_t)buffer[*index+1])<<16|
                        ((uint32_t)buffer[*index+2])<<8|
                        ((uint32_t)buffer[*index+3]);
        return res;
}

//int16Data Bit Collation
void buffer_append_int16(uint8_t* buffer, int16_t number, int32_t *index) {
                buffer[(*index)++] = number >> 8;
                buffer[(*index)++] = number;
        }

//uint16Data Bit Collation
void buffer_append_uint16(uint8_t* buffer, uint16_t number, int32_t *index) {
```

```
            buffer[(*index)++] = number >> 8;

            buffer[(*index)++] = number;

      }


//int32Data Bit Collation

void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index) {

            buffer[(*index)++] = number >> 24;

            buffer[(*index)++] = number >> 16;

            buffer[(*index)++] = number >> 8;

            buffer[(*index)++] = number;

      }


//uint32Data Bit Collation

void buffer_append_uint32(uint8_t* buffer, uint32_t number, int32_t *index) {

            buffer[(*index)++] = number >> 24;

            buffer[(*index)++] = number >> 16;

            buffer[(*index)++] = number >> 8;

            buffer[(*index)++] = number;

      }



  //Packet collation and sending

void packet_send_packet(unsigned char *data, unsigned int len, int handler_num) {

      int b_ind = 0;

      unsigned short crc;
```

```
    if (len > PACKET_MAX_PL_LEN) {
        return;
    }
    if (len <= 256) {
        handler_states[handler_num].tx_buffer[b_ind++] = 2;
        handler_states[handler_num].tx_buffer[b_ind++] = len;
    } else {
        handler_states[handler_num].tx_buffer[b_ind++] = 3;
        handler_states[handler_num].tx_buffer[b_ind++] = len >> 8;
        handler_states[handler_num].tx_buffer[b_ind++] = len & 0xFF;
    }

    memcpy(handler_states[handler_num].tx_buffer + b_ind, data, len);
    b_ind += len;

    crc = crc16(data, len);
    handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc >> 8);
    handler_states[handler_num].tx_buffer[b_ind++] = (uint8_t)(crc & 0xFF);
    handler_states[handler_num].tx_buffer[b_ind++] = 3;

    if (handler_states[handler_num].send_func) {
        handler_states[handler_num].send_func(handler_states[handler_num].tx_buffer, b_ind);
    }
}
```

5.3 MIT Communication Protocol

special can code

Enter motor control mode {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }

Exit motor control mode {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFD }

Set current motor position to point 0 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFE }

Note: When using CAN communication to control the motor, you must enter the motor MIT mode first!

ps: (If you want to read the current state when there is no state, the command to send is

{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC })

Definition of the data received by the operation MIT mode driver board

Identifier: Motor ID number set (default is 1) Frame type: Standard frame

Frame format: DATA DLC: 8 bytes

data domain	DATA[0]	DATA[1]	DATA[2]	DATA[3]	
data bit	7-0	7-0	7-0	7-4	3-0
data content	Motor position high 8 bits	Motor position low 8 bits	Motor speed high 8 bits	Motor speed low 4 bits	KP value high 4 bits

data domain	DATA[4]	DATA[5]	DATA[6]		DATA[7]
data bit	7-0	7-0	7-4	3-0	0-7
data content	Kp value low 8 bits	KD value high 8 bits	KD value low4 bits	current value high 4 bits	current value low 8 bits

Definition of the data sent by the driver board in MIT mode

Identifier: 0X00+Drive ID Frame Type: Standard Frame

Frame format: DATA DLC: 8 bytes

data domain	DATA[0]	DATA[1]	DATA[2]	DATA[3]	DATA[4]
data bit	7-0	7-0	7-0	7-0	7-4
data content	Drive ID number	Motor position high 8 bits	Motor position low 8 bits	Motor position high 8 bits	Motor position low 4 bits

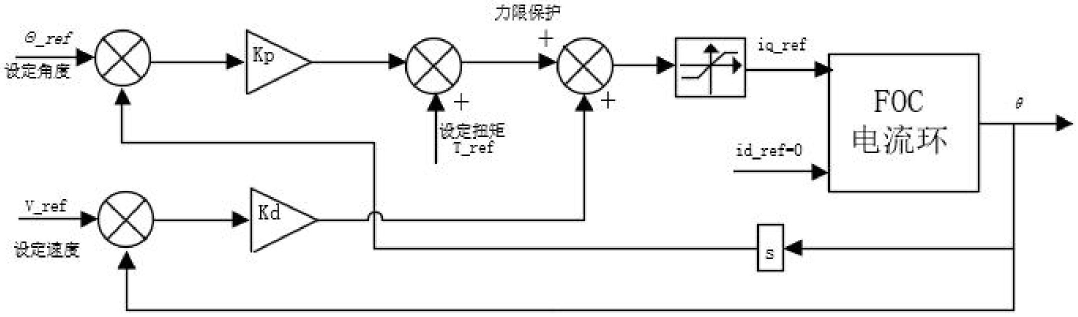
data domain	DATA[4]	DATA[5]	DATA[6]	DATA[7]
data bit	3-0	7-0	7-0	7-0
data content	Current high 4 bits	Current low 8 bits	Motor temperature	Motor Error Sign

CAN rate: 1 MHZ

MIT Control Mode Abbreviated Block Diagram

parameter range

motor	AK10-9	AK60-6	AK70-10	AK80-6	AK80-9	AK80-64	AK80-8	AK45-36	AK45-10	AK40-10
Motor position (rad)	-12.5f-12.5f									
Motor velocity (rad/s)	-50.0f-50.0f	-45.0f-45.0f	-50.0f-50.0f	-76.0f-76.0f	-50.0f-50.0f	-8.0f-8.0f	-37.5f-37.5f	-6.0f-6.0f	-20.0f-20.0f	-45.5f-45.5f
Motor torque (N.M)	-65.0f-65.0f	-15.0f-15.0f	-25.0f-25.0f	-12.0f-12.0f	-18.0f-18.0f	-144.0f-144.0f	-32.0f-32.0f	-34.0f-34.0f	-8.0f-8.0f	-5.0f-5.0f
Kp range	0-500									
Kd range	0-5									



MIT Control Mode Transceiver Code Routine

Transmission routine code:

```
u8 MIT_Can_Send_Msg(u8 stdId, u8 len)
{
    CanTxMsg TxMessage;
    u8 mbox;
    u16 i = 0;
    TxMessage.StdId = 1;                    // Standard identifier
    // TxMessage.ExtId = 0x00;        // Set extended identifier
    TxMessage.IDE = CAN_Id_Standard; // Standard frame
    TxMessage.RTR = CAN_RTR_Data;      // Data frame
    TxMessage.DLC = 8;                         // Data length to be sent

    for(i = 0; i < len; i++)

    mbox = CAN_Transmit(CANx, &TxMessage);
    i = 0;
    while((CAN_TransmitStatus(CAN2, mbox) == CAN_TxStatus_Failed) && (i < 0XFFF)) i++; // Wait for transmission to complete
}
if(i>=0XFFF)return 1;
return 0;
}
void pack_cmd( CanTxMsg *TxMessage,float p_des, float v_des, float kp, float kd, float t_ff){
 float P_MIN =-12.5f;

 float P_MAX =12.5f;

 float V_MIN =-30.0f;

 float V_MAX =30.0f;

 float T_MIN =-18.0f;

 float T_MAX =18.0f;

 float Kp_MIN =0;

 float Kp_MAX =500.0f;

 float Kd_MIN =0;

 float Kd_MAX =5.0f;

 float Test_Pos=0.0f;

int p_int ;

int v_int;

int kp_int ;

int kd_int;

int t_int ;

p_des = fminf(fmaxf(P_MIN, p_des), P_MAX);

v_des = fminf(fmaxf(V_MIN, v_des), V_MAX);

kp = fminf(fmaxf(Kp_MIN, kp), Kp_MAX);

kd = fminf(fmaxf(Kd_MIN, kd), Kd_MAX);

t_ff = fminf(fmaxf(T_MIN, t_ff), T_MAX);



    p_int = float_to_uint(p_des, P_MIN, P_MAX, 16);

  v_int= float_to_uint(v_des, V_MIN, V_MAX, 12);
```

```
        kp_int = float_to_uint(kp, Kp_MIN, Kp_MAX, 12);

        kd_int = float_to_uint(kd, Kd_MIN, Kd_MAX, 12);

        t_int= float_to_uint(t_ff, T_MIN, T_MAX, 12);



TxMessage->Data[0] = p_int >> 8;           // Position high 8 bits

TxMessage->Data[1] = p_int & 0xFF;         // Position low 8 bits

TxMessage->Data[2] = v_int >> 4;           // Speed high 8 bits

TxMessage->Data[3] = ((v_int & 0xF) << 4) | (kp_int >> 8);    // Speed low 4 bits, KP high 4 bits

TxMessage->Data[4] = kp_int & 0xFF;        // KP low 8 bits

TxMessage->Data[5] = kd_int >> 4;          // Kd high 8 bits

TxMessage->Data[6] = ((kd_int & 0xF) << 4) | (t_int >> 8);    // Kd low 4 bits, torque high 4 bits

TxMessage->Data[7] = t_int & 0xFF;         // Torque low 8 bits


MIT_Can_Send_Msg(MitCanId, 8);


}


//All the numbers in the packet are converted to integer numbers by the following function before they are sent to the motor.

int float_to_uint(float x, float x_min, float x_max, unsigned int bits){

          /// Converts a float to an unsigned int, given range and number of bits ///

          float span = x_max - x_min;

          if(x < x_min) x = x_min;

          else if(x > x_max) x = x_max;
          return (int) ((x- x_min)*((float)((1<<bits)/span)));

      }
```

Receive routine code:

```
float postion ;

float speed ;

float torque ;

float Temperature ;

int id;

int p_int;

int v_int;

int i_int;

float T_int;


void unpack_reply(CanRxMsg *RxMessage){

/// unpack ints from can buffer ///

     float P_MIN =-12.5f;

 float P_MAX =12.5f;

 float V_MIN =-30.0f;

 float V_MAX =30.0f;

 float T_MIN =-18.0f;

 float T_MAX =18.0f;

 float Kp_MIN =0;

 float Kp_MAX =500.0f;

 float Kd_MIN =0;
 float Kd_MAX =5.0f;

 float Test_Pos=0.0f;

 id = RxMessage->Data[0]; //driver id number

 p_int = ( RxMessage->Data[1]<<8)| RxMessage->Data[2];                // motor position data

 v_int = ( RxMessage->Data[3]<<4)|( RxMessage->Data[4]>>4);           //motor speed value

 i_int = (( RxMessage->Data[4]&0xF)<<8)| RxMessage->Data[5];     //motor torque value

 T_int =    RxMessage->Data[6] ;

/// convert ints to floats ///

//float p = uint_to_float(p_int, P_MIN, P_MAX, 16);

//float v = uint_to_float(v_int, V_MIN, V_MAX, 12);

//float i = uint_to_float(i_int, -T_MAX, T_MAX, 12);

//float T =T_int;


if(id == 1){

postion =    uint_to_float(p_int, P_MIN, P_MAX, 16);

speed = uint_to_float(v_int, V_MIN, V_MAX, 12);

torque = uint_to_float(i_int, -T_MAX, T_MAX, 12);

Temperature = T_int-40;     //temperature range-40~215

}

}

//All numbers in the packet are converted to floating point by the following function.

float uint_to_float(int x_int, float x_min, float x_max, int bits){

        /// converts unsigned int to float, given range and number of bits ///

        float span = x_max - x_min;

        float offset = x_min;
```

```
    return ((float)x_int)*span/((float)((1<<bits)-1)) + offset;

}
```

5.3.1 MIT Control Serial Protocol

The protocol for sending and receiving telegrams from the MIT mode serial port is as follows：

Header	Data length	Data frame	Data bit	Checksum bit		Tail
0x02	Without header, footer and parity bits	See data frame definition		High 8	Low 8	0x03

Check Digit Calculation Code Reference page 33

Frame definition：

```
typedef enum {

    COMM_FW_VERSION = 0,

    COMM_JUMP_TO_BOOTLOADER,

    COMM_ERASE_NEW_APP,

    COMM_WRITE_NEW_APP_DATA,

    COMM_TERMINAL_CMD=20,

    COMM_PRINT=21,

} COMM_PACKET_ID;
```

5.3.1.1 debugging command

1、Get encoder value

Command：

```
    02 08 14 65 6E 63 6F 64 65 72 B0 4C 03
```

respond：

```
    02 35 15 45 20 41 4E 47 4C 45 20 3A 35 2E 32 34 33 38 31 30 20 20 4D 20 41 4E 47 4C 45 3A
```

20 2D 31 39 2E 37 39 32 35 37 30 20 20 20 52 41 57 3A 20 32 35 34 31 0A 0D 0F 3F 03

clarification：

Send encoder data in real time

Parameter parsing：

02 (frame header) + 35 (data length) + 15 (data frame) + 45 20 41 4E 47 4C 45 20 3A 35 2E 32 34 33 38 31 30 20 20 4D 20 41 4E 47 4C 45 3A 20 2D 31 39 2E 37 39 32 35 37 30 20 20 20 52 41 57 3A 20 32 35 34 31 0A 0D (String motor position + mechanical position + encoder position) + CRC + 03 (end of frame)

String motor position + mechanical position + encoder position

45 20 41 4E 47 4C 45 20 3A 35 2E 32 34 33 38 31 30 20 20 4D 20 41 4E 47 4C 45 3A 20 2D 31 39 2E 37 39 32 35 37 30 20 20 20 52 41 57 3A 20 32 35 34 31 0A 0D

corresponding string： E ANGLE :5.243810 M ANGLE: -19.792570 RAW: 2541

E ANGLE： 0-6.283185

M ANGLE： Unlimited scope

RAW： 0-16383

2、Calibrate encoder

Command：

02 0A 14 63 61 6C 69 62 72 61 74 65 76 A5 03

respond：

02 1B 15 0A 0D 20 43 61 6C 69 62 72 61 74 69 6F 6E 20 63 6F 6D 70 6C 65 74 65 2E 0A 0D 52 F1

clarification：

The motor will automatically rotate and calibrate the encoder and return the calibration data, and return the response data when calibration is complete.

Parameter parsing：

02 1B 15 “\n\r Calibration complete.   Press 'exit' to return to menu\n\r”

02 (frame header) + (data length) + 15 (data frame) + (string calibration feedback) + CRC

3、Exit debugging

Command：

02 05 14 65 78 69 74 96 C3 03

Clarification：

Exit motor debugging