AK Series Actuator Manual

V1.0.9

CONTENT

Notice

1. Ensure that the circuit is normal and the interface is correctly connected as required.

2. The driver board will be hot when outputing, please use it carefully to avoid burns.

3. Please Check whether the parts are in good condition before use. If any parts are missing or aging, please stop using and contact technical support in time.

4. Different optional control modes can’t be switched when driver board is working, and different control mode match different communication protocol. If you need to switch, please reboot the power to the diver board then to change. Using the wrong protocol control may burn the driver board.

5. Please use it strictly in accordance with the working voltage, current, temperature and other parameters specified in this article, otherwise it will cause permanent damage to the product.

Feature

The AK series actuators' driver board adopts the driver chip with high-performance, uses the Field Oriented Control (FOC) algorithm, and is equipped with advanced active disturbance rejection control technology to control the speed and angle. It is matched with our modular motor to form a powerful power package. It can be used with CubeMars Tool assistant software for parameter setting and firmware upgrade.

Disclaimer

Thank you for purchasing the AK series actuators. Before using, please read this statement carefully. Once used, it is deemed to be an endorsement and acceptance of the entire content of this statement. Please strictly abide by the product manual and related laws, regulations, policies and guidelines to install and use the product. In the process of using the product, the user promises to be responsible for his actions and all consequences arising therefrom.

CubeMars will not be liable for any losses caused by improper use, installation, or modification by the user.

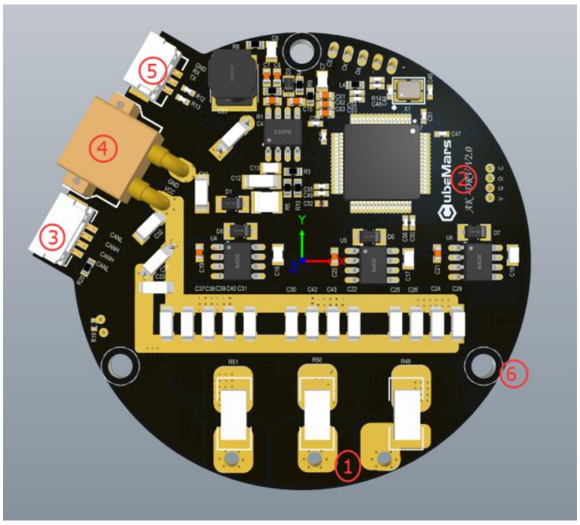
CubeMars is a trademark of JIANGXI XINTUO ENTERPRISE CO., LTD and its affiliates. The product names, brands, etc. appearing in this article are the trademarks of their respective companies. This product and manual are copyrighted by JIANGXI XINTUO ENTERPRISE CO., LTD. Without permission, it is not allowed to reproduce in any form. The final interpretation right of the disclaimer belongs to JIANGXI XINTUO ENTERPRISE CO., LTD.

Version Change Record

Date	Version	Change content
2021.9.1	Ver. 1.0.0	create version
2021.10.08	Ver.1.0.2	Code update of 5.1 and 5.2
2021.10.29	Ver.1.0.3	data definitions update of 5.1,5.2 and 5.3
2021.11.15	Ver.1.0.4	Message acceptance added of 5.2 and 5.3
2021.11.24	Ver.1.0.5	UART protocol update of 5.2
2021.11.30	Ver.1.0.6	Information added of 5.3
2022.01.20	Ver.1.0.7	Data change of AK60-6 motor speed in 5.3
2022.02.24	Ver.1.0.8	Servo Mode Serial Message Protocol added in 5.2
2022.05.17	Ver.1.0.9	Explain of the Servo Mode Serial Message Protocol added in 5.2

1.Drive Product Information

1.1 Introduction of Drive' appearance &Specifications



①Three-phase wires connection port

②Hardware version

③CAN communication connection port

④DC power port

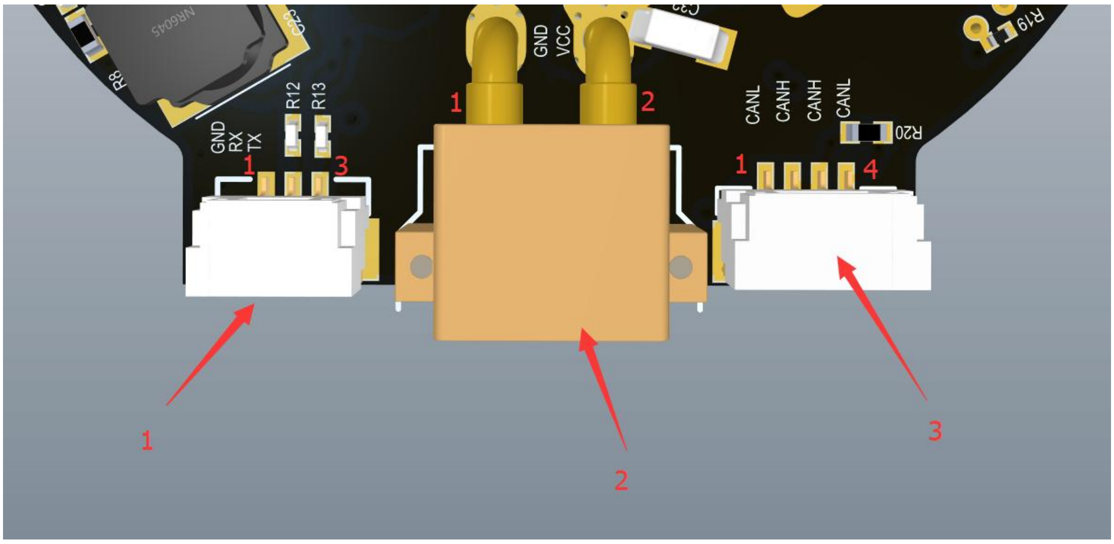
⑤Serial communication connection port

⑥Mounting holes

Specifications	
Rated Voltage	24V-48V (12V-24V for only AK60-6)
Peak Voltage	52V
Rated current	20A
Peak current	60A
Power consumption	≤50mA
Can bus bit rate	1Mbps (no change recommended)
Size	62mm×58mm
Working Environment temperature	-20℃-65℃
Maximum allowable temperature of driver board	100℃
Encoder Accuracy	14bit(single turn absolute)

1.2 Drive connector and Definition

1.2.1 Drive connector Diagram



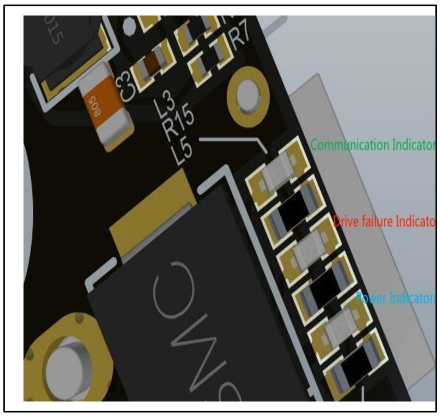
1.2.2 The Brand and Model of Drive connector

No.	Onboard interface model	Brand	Wire connector model	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 Drive connector pin Definition

No.	connector function	Pin	Explanation
1	**Serial communication**	1	Serial signal ground（GND）
		2	Serial signal output （TX）
		3	Serial signal input（RX）
2	**POWER INPUT**	1	Negative pole（-）
		2	Positive pole（+）
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 Drive Indicator Definition



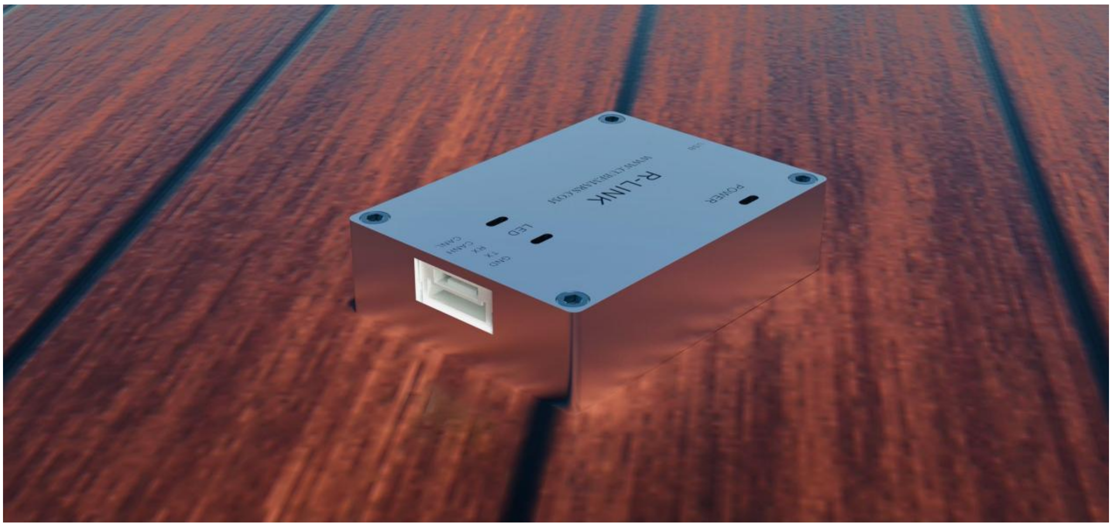
Indicator definition	
1.Power indicator(when blue light is on）	The power indicator is used to show the power supply of the driver board. Normally, it will light up blue when the power is plugged in. If the blue light is not on, please remove the power immediately and dont turn the power on again.
2.Communication Indicator（ when green light is on）	The communication indicator is used to show the communication status of the driver board. normally the driver board will light up green when the driver board communicates normally. If the green light is not on, please check whether the CAN communication wiring is normal.
3.Drive failure indicator （when red light is on）	The drive failure indicator is used to show the failure of the drive board. normally, it will light up in red only when the drive board fails, and it will usually go off under normal circumstances. When the drive failure indicator lights up, it means that the drive board has been damagedto some extent. The power supply should be turned off and no operation is allowed.

1.4 Main Accessories and Specifications

NO.	Item	Specification		QTY	Remark
1	Serial communication line	wire rod	24AWG-300MM-Teflon silver-plated wire-black yellow green	Each 1PCS	±2MM
		pin	A1257H-3P	1PCS	
			A2541H-3P	1PCS	
2	power wire	wire rod	16AWG-200MM-Silicone wire-red and black	Each 1PCS	±2MM
		pin	XT30UPB-M	1PCS	
			XT30UPB-F	1PCS	
3	CAN communication wire	wire rod	24AWG-300MM-Teflon silver-plated wire-white and blue	Each 1PCS	±2MM
		pin	A1257H-4P	2PCS	
			A2541H-2P	1PCS	
4	Thermistor	MF51B103F3950-10K-3950		2PCS	
5	Electrolytic capacitor	120Uf-63V-10x12MM		2PCS	AK10-9 V2.0 or above is standard equipment
6	MOS power	BSC026N08NS5-80V-2.6mΩ TPH2R608NH-75V-2.6mΩ		12PCS	Random

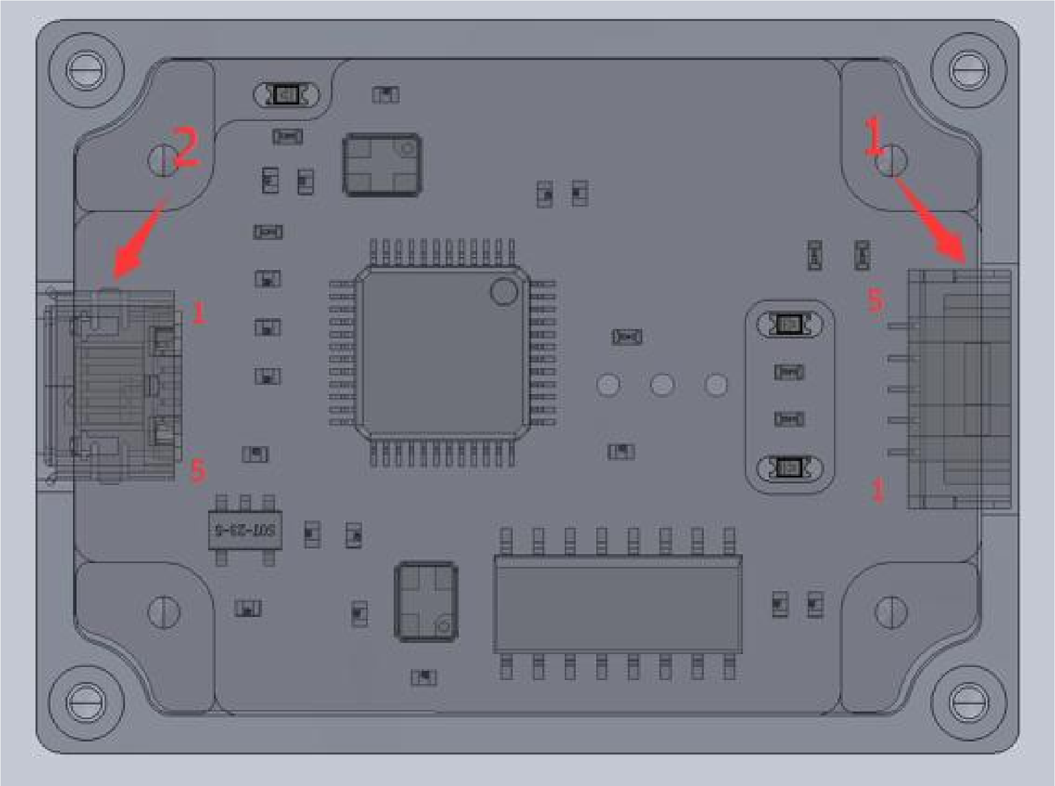
2. R-link produce information

2.1 Introduction of R-link' appearance&Specifications



Specification	
Rated Voltage	5V
Power consumption	≤30mA
Size	39.2x29.2x10MM
Working Environment temperature	-20℃-65℃
Maximum allowable temperature of board	85℃

2.2 R-link connector and Definition

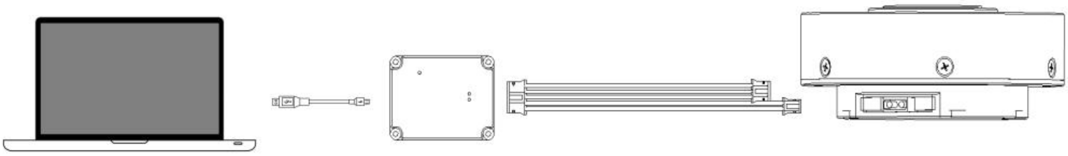


No.	connector function	Pin	Definition
1	Communication connector	1	CAN low (CAN_L)
		2	CAN high (CAN_H)
		3	Serial signal input（RX）
		4	Serial signal output （TX）
		5	Serial signal ground（GND）
2	USB connector	1	VBUS
		2	D-
		3	D+
		4	ID
		5	GND

2.3 R-link Indicator Definition

No.	Color	Definition
1	GREEN	The power indicator is used to indicate the power status of the R-link. Under normal circumstances, it will light up green when the power is plugged in. If the green light does not light up when the power is plugged in, please remove the power immediately and dont turn on the power again
2	BLUE	Serial communication output (TX), always off, flashes when there is data output from the R-link serial port.
3	RED	Serial communication output (TX), always off, flashes when there is data input from the R-link serial port.

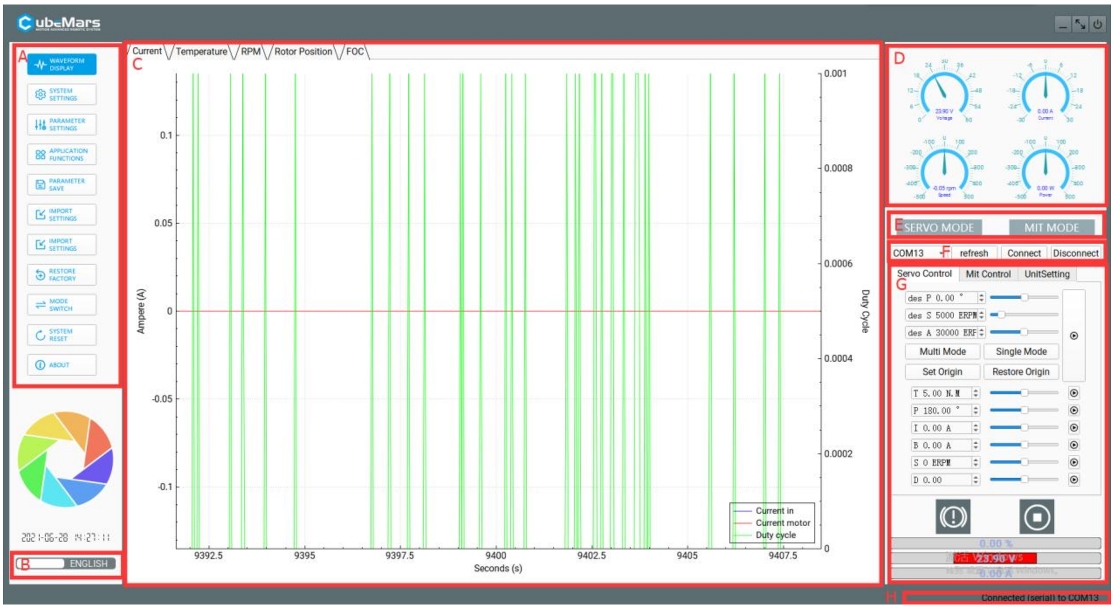
3. Actuator and R-link Connection and Notices



Connection instructions: Connect the USB cable to the PC and R-Link, the 5-Pin port to the R-Link 5 pin port, the 4-Pin port to the CAN port of the motor, and the 3-Pin to the UART port of the motor.

4. Instructions for the use of the upper computer

4.1 GUI interface and instruction



COM13

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

0.00 %
23.90 V
0.00 A

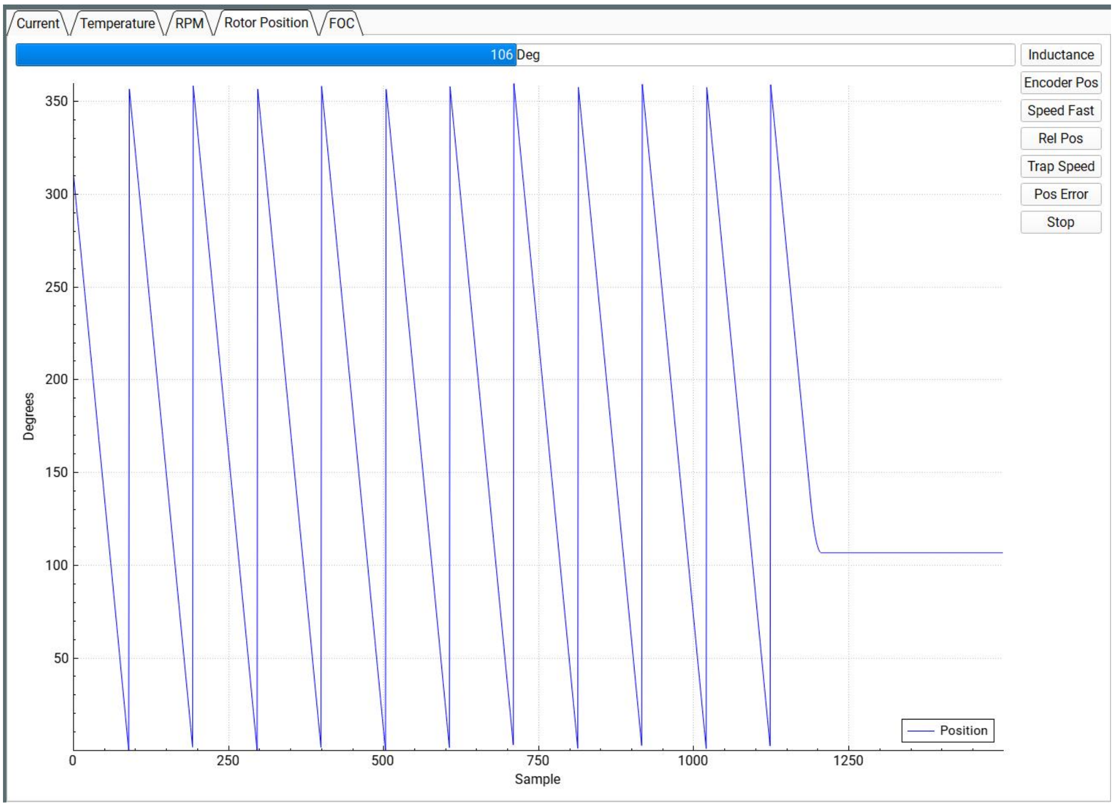
Connected (serial) to COM13

A. Home
B. Chinese and English Switching
C. Main page
D. Implement data display
E. Current mode
F. Serial port selection
G. control parameter

4.1.1 Home menu

4.1.1.1 waveform display

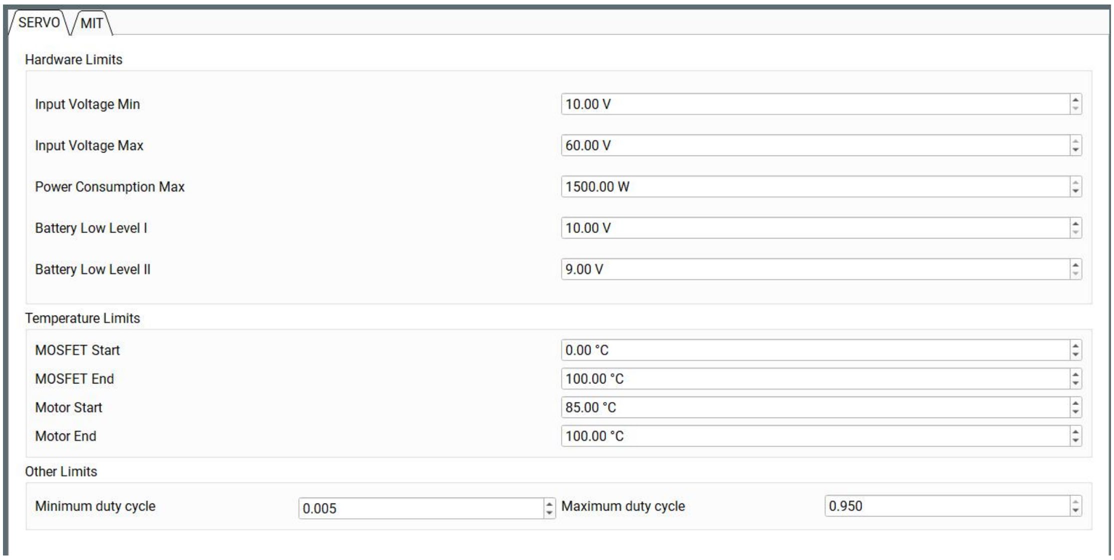
This page allows viewing real-time data feedback and drawing images. Data includes: motor current, temperature, real-time speed, inner encoder position, outer encoder position, high-frequency speed, rotor position, path planning, position deviation, DQ current, etc.



4.1.1.2 System Setting

This page is mainly for changing the hardware limitations of the drive board such as voltage, current, power, temperature, duty ratio, etc. It mainly protects the drive board and motors.

⚠：Please use it strictly in accordance with the specified voltage, current, power, and temperature. Our company will not assume any legal responsibility if the operation of this product cause any injury to the human body or irreversible damage to the drive board and motor by any violation of regulations.



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

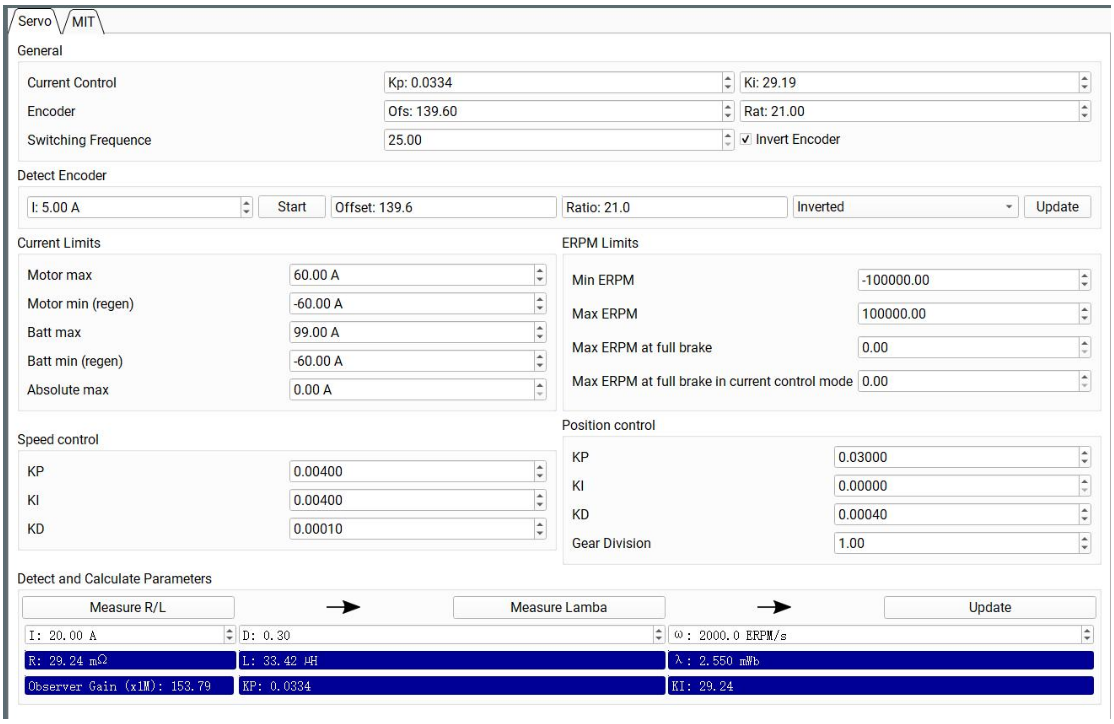
Other Limits

Minimum duty cycle	0.005	Maximum duty cycle	0.950

4.1.1.3 Parameter Setting

This page is mainly about adjusting the parameters of the drive board, including but not only restrict to current loop Kp-Ki, encoder paranoia, maximum and minimum current, maximum and minimum speed, speed loop Kp-Ki-KD, reduction ratio and other parameters, as well as encoder calibration and motor parameter confirm.

⚠：Please use it strictly in accordance with the specified voltage, current, power, and temperature. Our company will not bear any legal responsibility if the operation of this product cause any injury to the human body or irreversible damage to the drive board and motor by any violation of regulations.



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 | Offset: 139.6 | Ratio: 21.0 | 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

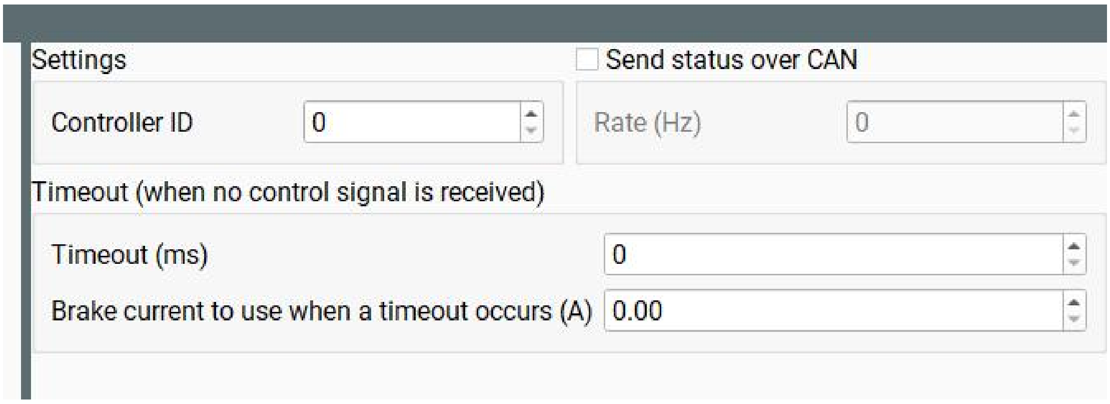
KP	0.03000
KI	0.00000
KD	0.00040
Gear Division	1.00

Detect and Calculate Parameters

I: 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 Functions

This page is mainly about CAN ID setting, CAN communication rate and CAN communication settings of sudden interruption.



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 Parameter Save

Save current motor parameters to the the upper computer. Attention: Please ensure hit this button every time you change the parameters, or other parameters will result in error. If it happens, please download the proper default app parameters from our official website and allows default parameter to be written in the motor by clicking the “import setting’’

4.1.1.6 Export Settings

Save the upper computer parameters as two files with the suffixes ".McParams" and ‘‘AppParams’’ to the computer.

The ".McParams" file is:

The ".AppParams" file is:

4.1.1.7 Import Settings

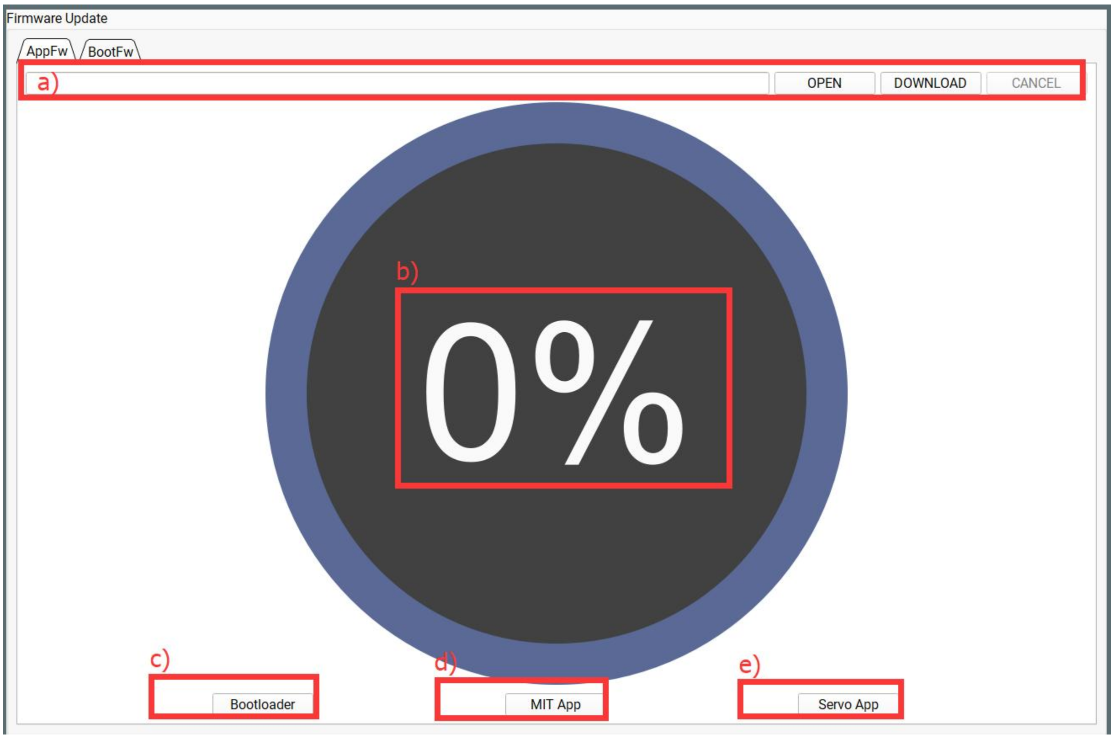
Upload the parameters of the two files with the suffix ".McParams" and ".AppParams" on the computer to the upper computer.

4.1.1.8 Restore Factory

This feature is not currently enabled.

4.1.1.9 Mode Switch

This page is mainly about switching the control mode of the drive board, including "guide mode", "servo mode" and "MIT power mode", and driver board firmware update.



A).Import firmware area： It can import files with the suffix ".bin" in the computer.

B). Firmware update progress bar

C). Enter boot mode

D). Enter MIT power mode

E). Enter servo mode

4.1.1.10 System reset

Stop the actuator and restart.

4.1.1.11 About

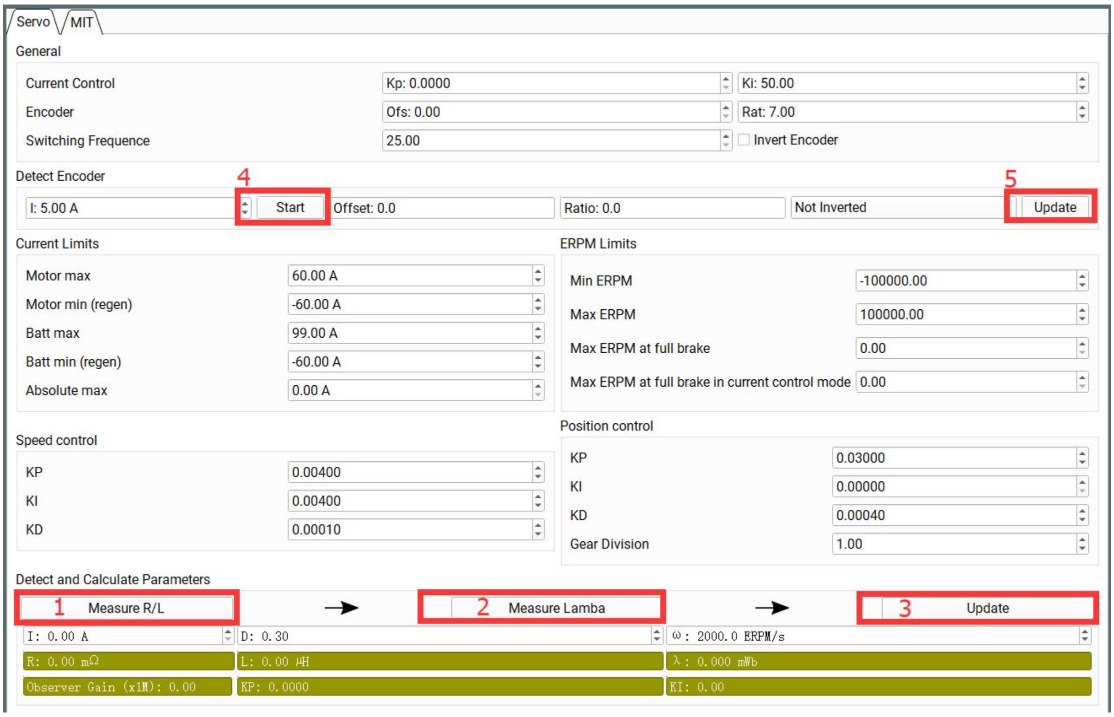
About the version number of the host computer and the official website of the company https://www.cubemars.com/

4.2 Driver board calibration

After you reinstall the driver board on the motor, or change the line sequence of the motor's three-phase line, or update the firmware, calibration is required. The motor can be used normally after calibration.

4.2.1 Servo mode

Confirm 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 upper computer, enter the system setting page, and click 1"Electrical Parameters", 2"Identification Parameters", 3"Update Parameters", 4 "Start calibration", 5 "Update parameters" successively.



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

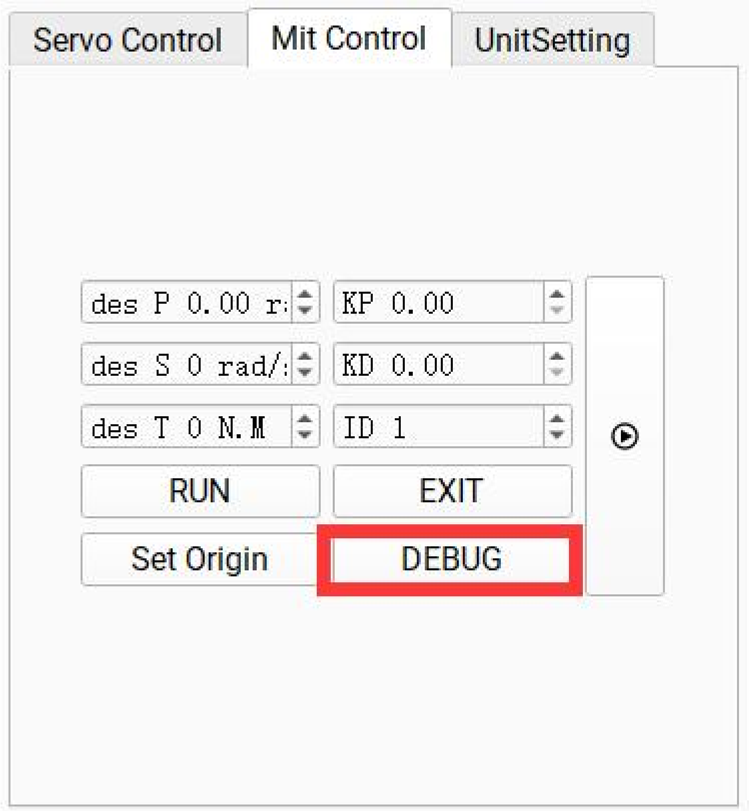
I: 0.00 A D: 0.30 ω: 2000.0 ERPM/s

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 power mode

Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in MIT mode, after successfully connecting with the upper computer, (1)click "Debug Mode" on the "Motion Control" interface, and then input "calibrate" in the input field, Wait for about 30 seconds. At the same time, the output field will scroll the position value of the encoder in real time until the output field prints "Encoder Electrical Offset (rad)", the actuator will reboot and print the message from the driver board. When calibrating, the voltage is about 1A at 48V (except AK60-6 working in 24V), and the current is back to about 0.02A after the calibration.



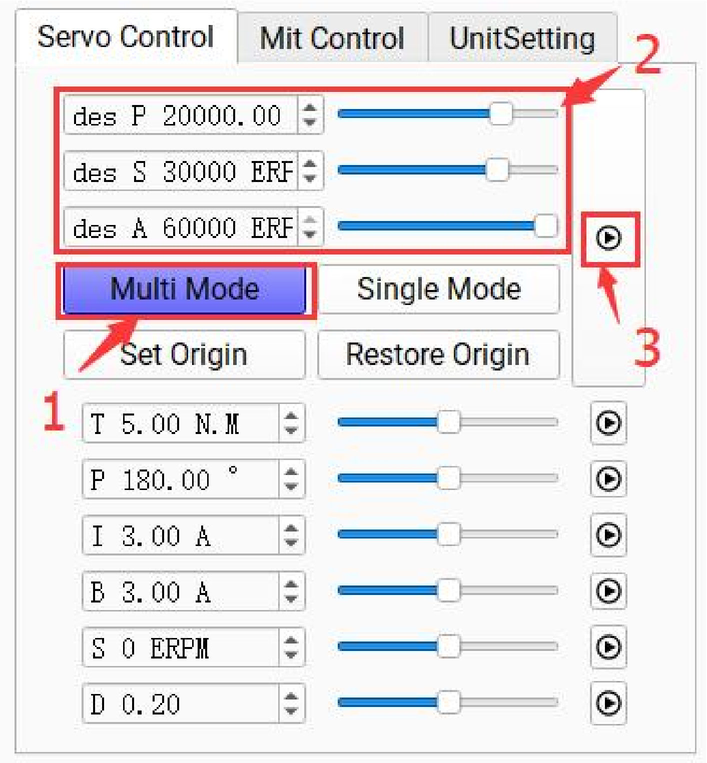
des P 0.00 r:
KP 0.00
des S 0 rad/:
KD 0.00
des T 0 N.M
ID 1

4.3 Control demo

4.3.1 Servo mode

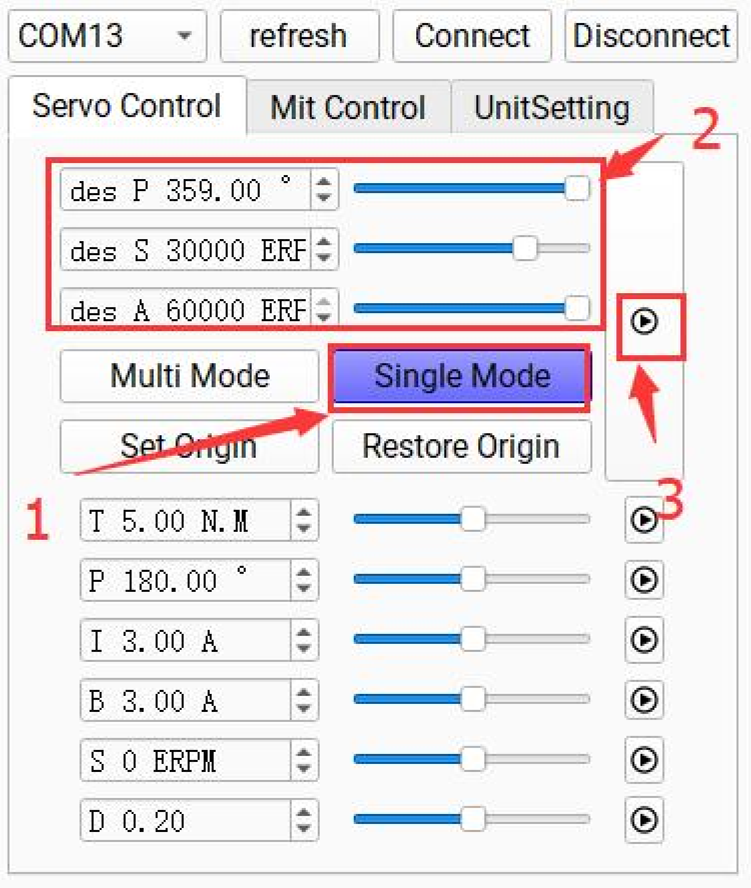
4.3.1.1 Multi-loop position velocity mode

Confirm that 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 upper computer, click (1)"multi-loop mode" on the "servo control" interface, and (2) input the expected position (the position at this time is ± 100 loops, is from -36000°to 36000°), (3) the motor will move at the expected speed until it reaches the expected position.



4.3.1.2 Single loop position velocity mode

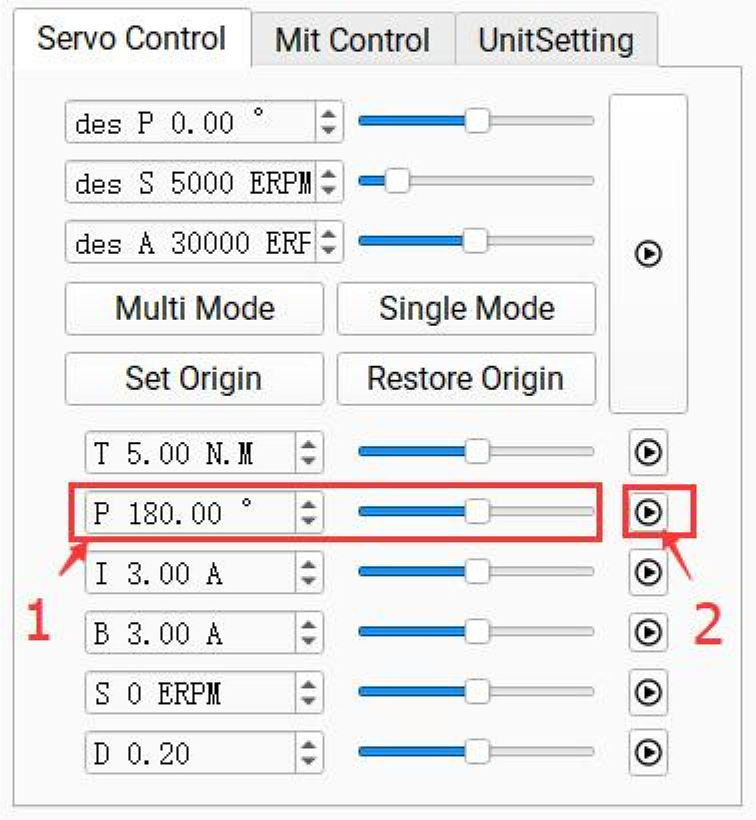
Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in servo mode, after successfully connecting with the upper computer, (1) click "single-loop mode" on the "servo control" interface, and after (2) inputting the expected position (there is only one circle at this time, is from 0°to 359°), (3) the motor will move at the expected speed until it reaches the expected position.



des P 359.00 °
des S 30000 ERF
des A 60000 ERF
T 5.00 N.M
P 180.00 °
I 3.00 A
B 3.00 A
S 0 ERPM
D 0.20

4.3.1.3 Position mode

Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in servo mode, (1) input the expected position in the "Servo Control" interface after connecting with the upper computer successfully, and (2) the motor will reach the expected position at the maximum 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

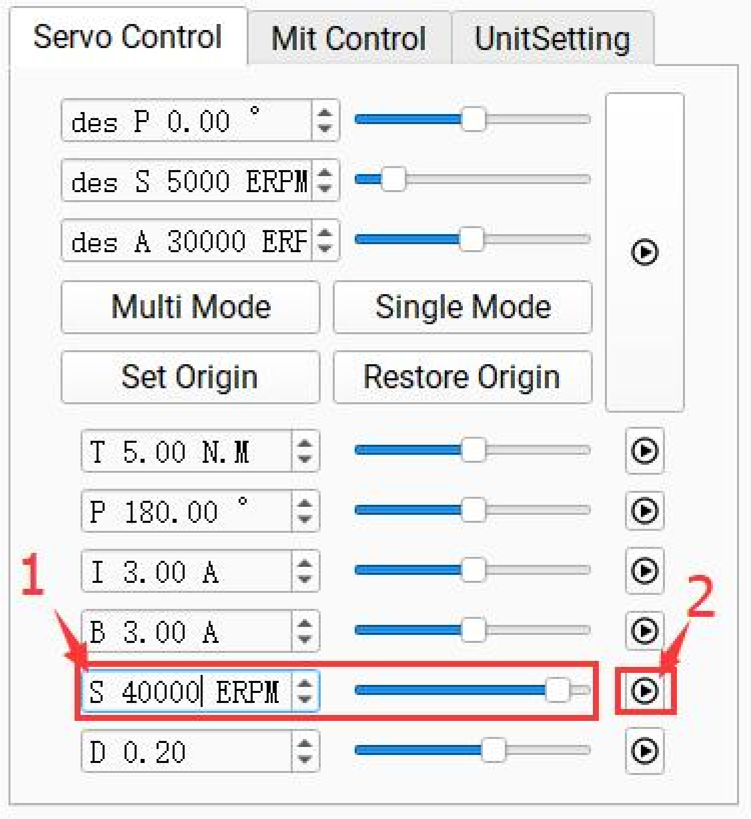
B 3.00 A

S 0 ERPM

D 0.20

4.3.1.4 Velocity mode

Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in servo mode, after connecting with the upper computer successfully, input the expected speed (±50000ERPM) in the "Servo Control" interface, and the motor will move at the desired 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

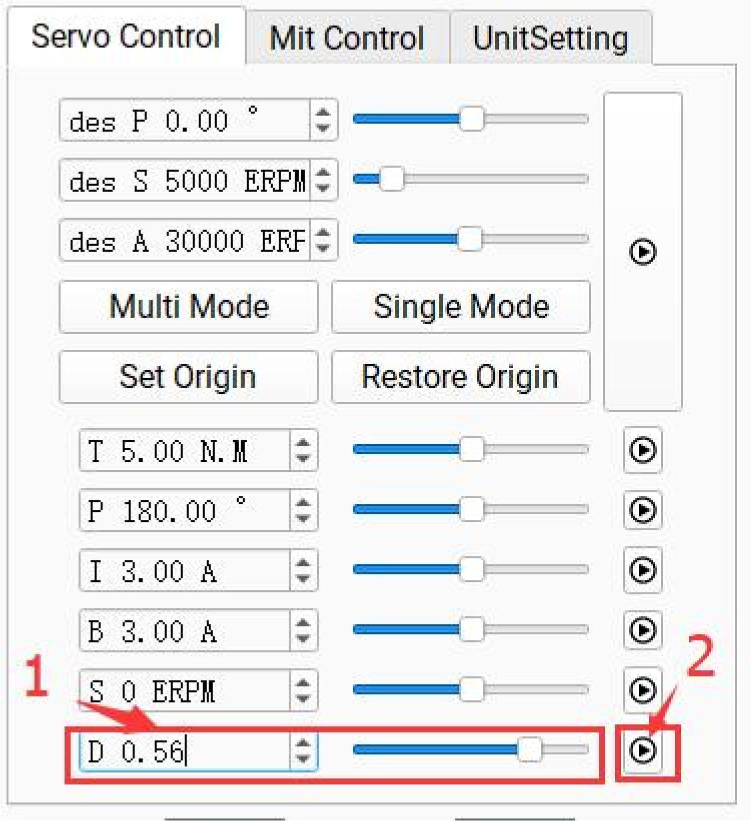
B 3.00 A

S 40000 ERPM

D 0.20

4.3.1.5 Duty cycle mode

Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in servo mode, input the expected duty ratio(default 0.005-0.95) in the "Servo Control" after connecting with the upper computer, the motor will work at the desired duty ratio.



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

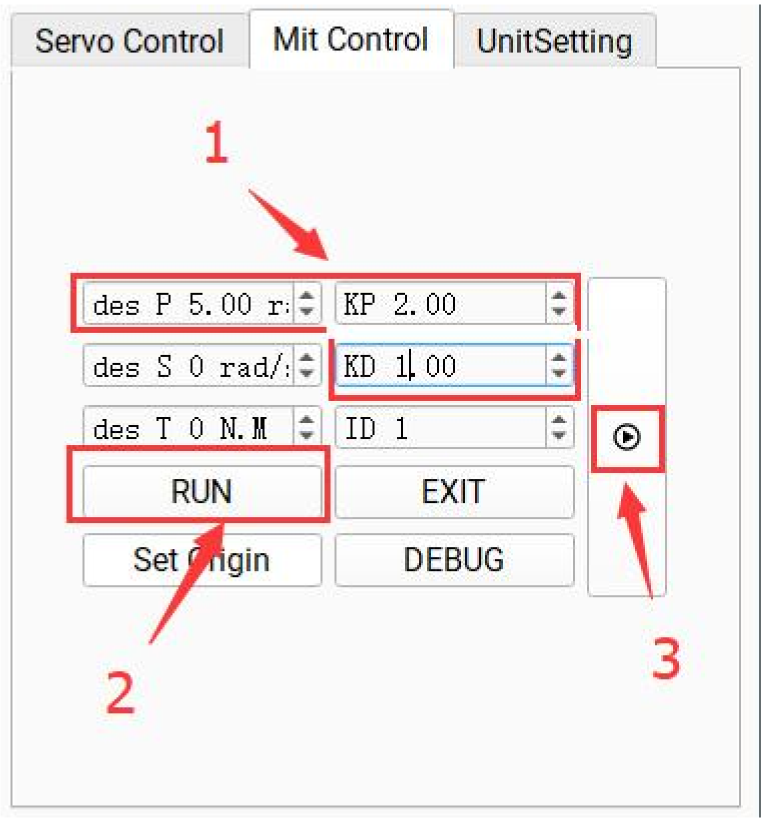
S 0 ERPM

D 0.56

4.3.2 MIT power Mode

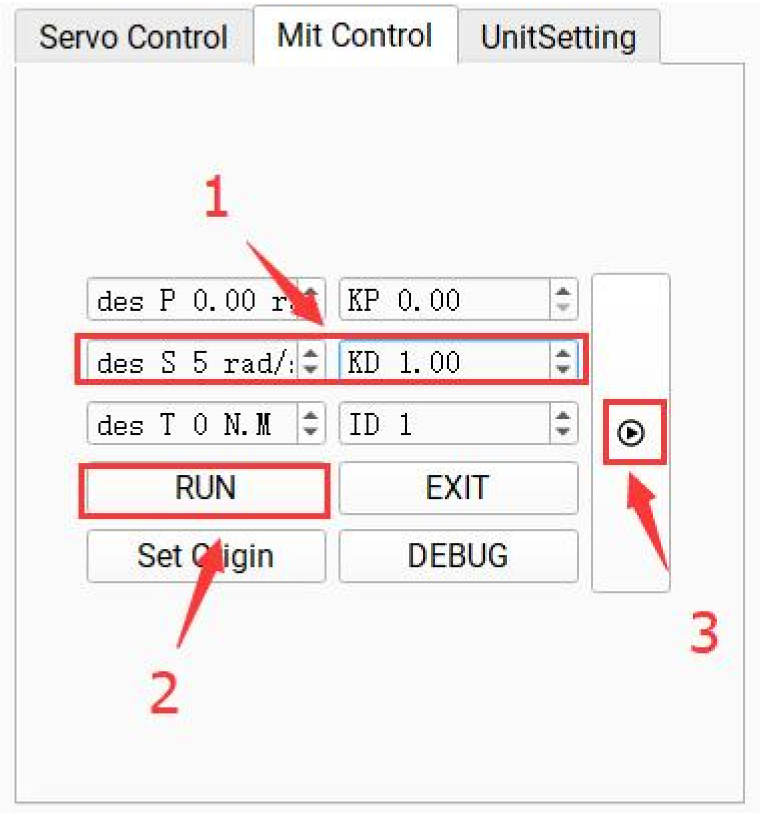
4.3.2.1 Position Mode

Confirm that input power of the motor is stable, the R-LINK connection is normal, and the motor is in MIT mode, after connecting with the upper computer successfully, (1)input matched “CAN ID” in the “Mit Control” interface and then (2)click “RUN”， you can enter the motor mode. The motor will do position movement (default speed 12000erpm, acceleration 40000erpm) after inputting desired position, KP and KD.



4.3.2.2 Velocity mode

Confirm that input power of the motor is stable, R-Link connection is well, and the motor is in MIT mode. After the motor successfully connect with the upper computer, (1)enter the matched "CAN ID" on the "Mit Control" interface and (2)click "RUN" to enter the motor mode. After the expected speed and KD are input, the motor will running at certain speed.



des P 0.00 r

KP 0.00

des S 5 rad/

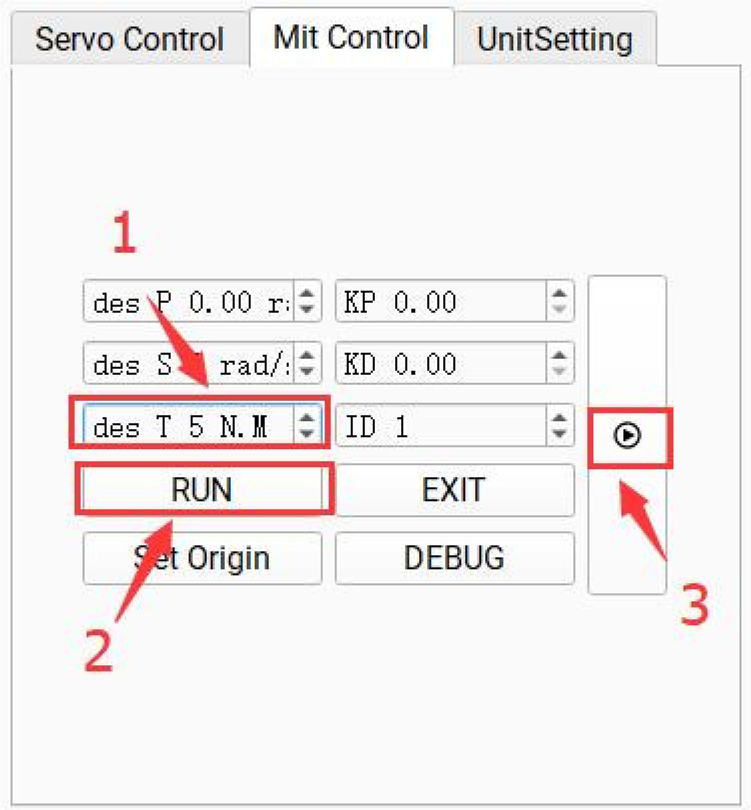
KD 1.00

des T 0 N.M

ID 1

4.3.2.3 Torque mode

Confirm that input power of the motor is stable, R-Link connection is normal, and the motor is in MIT mode. After the motor successfully connect with the upper computer, （1） enter the corresponding "CAN ID" on the "Mit Control" interface and (2) click "RUN" to enter the motor mode. After the expected torque is input, the motor will running according to the torque.

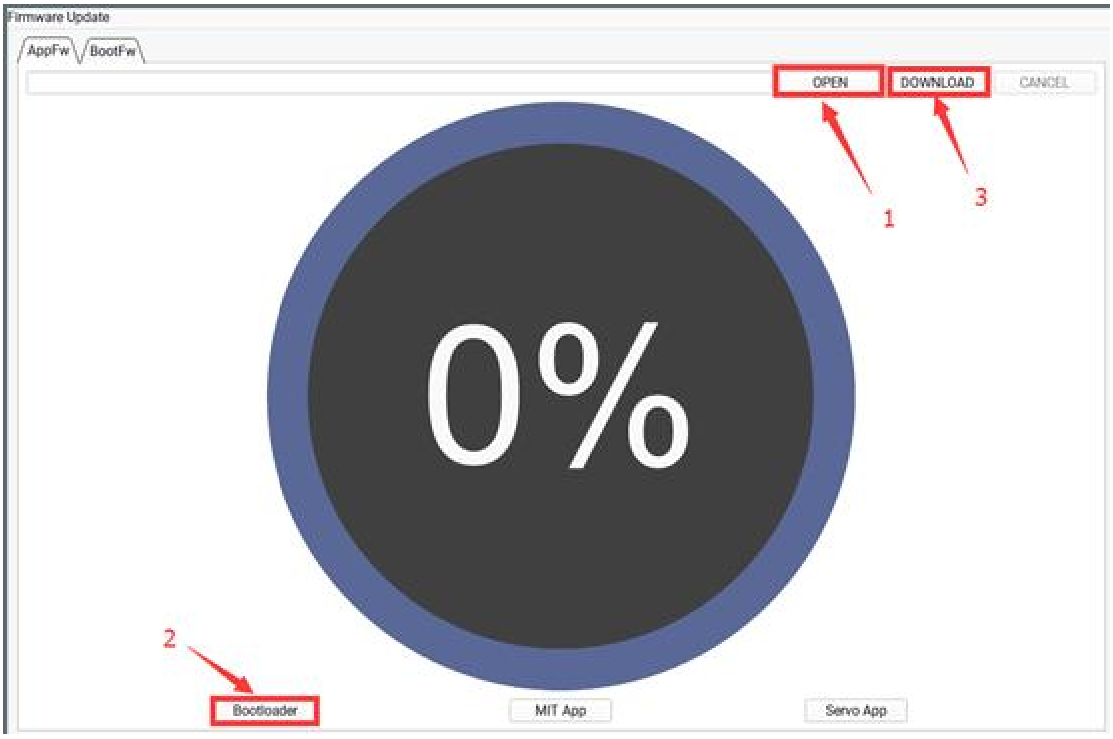

Servo Control
Mit Control
UnitSetting
1
des P 0.00 r:
KP 0.00
des S 0 rad/:
KD 0.00
des T 5 N.M
ID 1
RUN
EXIT
Set Origin
DEBUG
2
3

4.4 Firmware update

1. Click Open File and select the firmware. The firmware name suffix commonly end with is “bin”.

2. Click Bootloader.

3. Click download and wait for the progress bar reach to 100%. Then restart the power supply.



5. Driver board communication protocol and description

5.1 Servo mode and control mode description

Servo mode has six control modes

Duty cycle mode: a certain duty cycle voltage to motor.
Current loop mode: a Iq current to motor, the motor output torque = Iq *KT, so it can be used as a torque loop
Current brake mode: a certain brake current to motor, in order to fasten the motor at the current position (pay attention to the motor temperature when using)
Velocity mode: a certain motion speed to motor
Position mode: a certain position to motor, the motor will run to the specified position, (default speed 12000erpm acceleration 40000erpm)
Position velocity loop mode: a certain position, speed and acceleration to motor. The motor will run at a given acceleration and maximum speed to a specified position.

The servo motor protocol is CAN protocol, and the extended frame format is shown below

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

Control mode contain {0,1,2,3,4,5,6,7} Seven eigenvalues correspond to seven control modes respectively

Duty cycle mode: 0
Current loop mode: 1
Current brake mode: 2
Velocity mode: 3
Position mode: 4
Set origin mode:5
Position velocity loop mode: 6

Examples of various mode control motors are provided below
The following are library functions and macro definitions for each instance

```
typedef enum {
    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,                    //Velocity 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_ID;


void comm_can_transmit_eid(uint32_t id, const uint8_t *data, uint8_t len) {
        uint8_t i=0;
        if (len > 8) {
                len = 8;
        }
    CanTxMsg TxMessage;
    TxMessage.StdId = 0;
        TxMessage.IDE = CAN_ID_EXT;
        TxMessage.ExtId = id;
        TxMessage.RTR = CAN_RTR_DATA;
        TxMessage.DLC = len;
        //memcpy(txmsg.data8, data, len);
        for(i=0;i<len;i++)
    TxMessage.Data[i]=data[i];
    CAN_Transmit(CHASSIS_CAN, &TxMessage);
}

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 sends data definitions

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

```
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 sends data definition

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

Among them, the current value is of int32 type, and the value -60000-60000 represents -60-60A.

Current loop mode sending 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 sending data definition

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

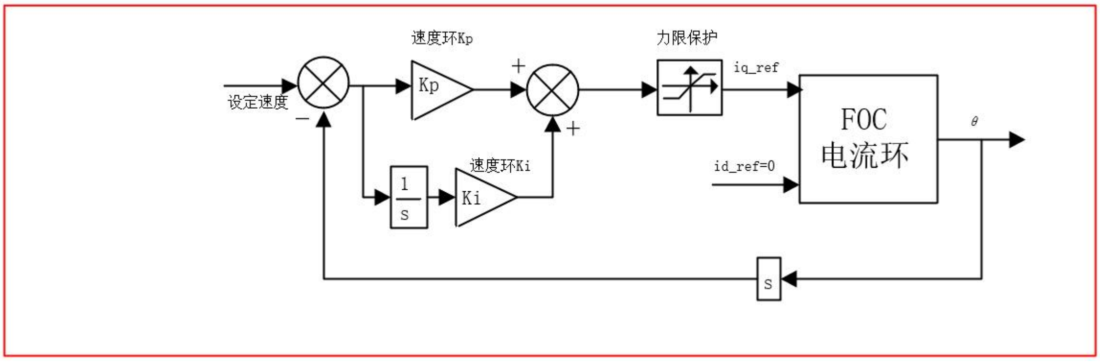
Among them, the braking current value is of int32 type, and the value 0-60000 represents 0-60A.

Current brake mode sending 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 mode

Velocity loop simple control block diagram



Velocity loop mode sending data definition

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

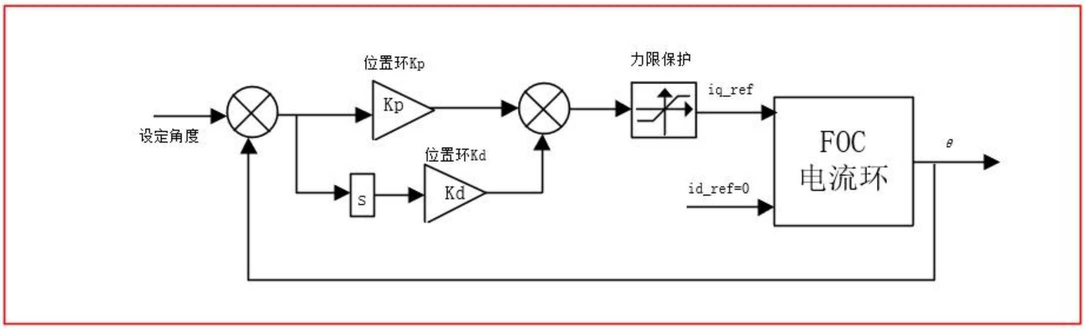
Among them, the speed value is int32 type, and the range -100000-100000 represents -100000-100000 electrical speed.

Velocity loop sending 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 brief control block diagram



Position loop mode sending data definitions

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

Position loop sending routine, position as int32 type， range-360000000~360000000 represents position -36000° ~36000°

```
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 * 1000000.0), &send_index);
        comm_can_transmit_eid(controller_id |
                        ((uint32_t)CAN_PACKET_SET_POS << 8), buffer, send_index);
}
```

5.1.6 Set origin mode

Date bits	Data[0]
Range	0~0x02
Corresponding variable	Set instruction

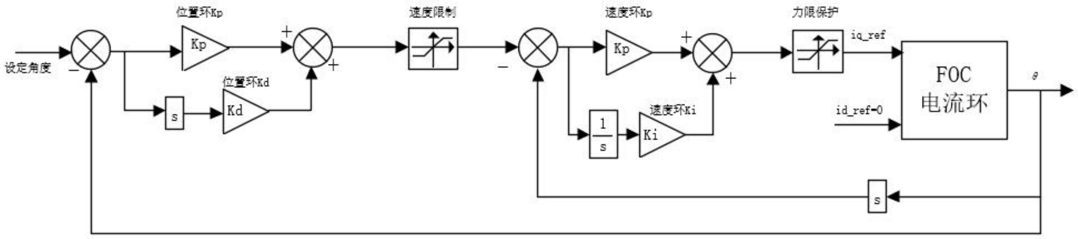
Among them, the setting command is uint8_t type, 0 means setting the temporary origin (power failure elimination), 1 means setting the permanent zero point (automatic parameter saving), 2 means restoring the default zero point (automatic parameter saving);

Position loop sending routine

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

5.1.7Position and Velocity Loop Mode

Simplified block diagram of position velocity loop



Position velocity loop mode sending data definition

Data bits	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 variables	Position 25-32 bit	Position 17-24 bit	Position 9-16 bit	Position 1-8 bit	Speed 8 bit high	Speed 8 bit low	Accelerated speed 8 bit high	Accelerated speed 8 bit low

Among them, the position is int32 type, and the range -360000000-360000000 represents the position -36000°-36000°;

Among them, the speed is int16 type, and the range -32768-32767 represents -327680-327680electrical speed;

Among them, the acceleration is int16 type, and the range 0-32767 represents 0-327670 electrical speed/s². 1 unit equals 10 electrical speed /s².

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

5.2 Servo mode of motor message format

5.2.1 Servo mode CAN uploading message protocol

In servo mode, motor messages are uploaded in timing mode. The upload frequency can be set as 1-500Hz, and the upload byte is 8 bytes

Data bits	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 variables	Position 8 bit high	Position 8 bit low	Speed 8 bit high	Speed 8 bit low	Current 8 bit high	Current 8 bit low	Motor temperature	An error code

Among them, the position is int16 type, and the range -32000-32000 represents the position -3200° -3200° ;

Among them, the speed is int16 type, and the range -32000-32000 represents -320000-320000rpm electrical speed;

Among them, the current is of type int16, and the value -6000-6000 represents -60-60A.

Among them, the temperature is int8 type, and the range of -20-127 represents the temperature of the driver board: -20℃-127℃;

Among them, the error code is uint8 type, 0 means no fault, 1 means over temperature fault, 2 means over current fault, 3 means over voltage fault, 4 means under voltage fault, 5 means encoder fault, 6 means phase current unbalance fault (The hardware may be damaged);

The following is an example of message acceptance

```
  void motor_receive(float* motor_pos,float*
motor_spd,float* cur,int_8* temp,int_8* error,rx_message)
   {
        int16_t pos_int = (rx_message)->Data[0] << 8 | (rx_message)->Data[1]);
        int16_t spd_int = (rx_message)->Data[2] << 8 | (rx_message)->Data[3]);
        int16_t cur_int = (rx_message)->Data[4] << 8 | (rx_message)->Data[5]);
      &motor_pos= (float)( pos_int * 0.1f); //motor position
      &motor_spd= (float)( spd_int * 10.0f);//motor speed
```

```
&motor_cur= (float) ( cur_int * 0.01f);//motor current
&motor_temp= (rx_message)->Data[6] ;//motor temperature
&motor_error= (rx_message)->Data[7] ;//motor error mode
}
```

5.2.2 Servo Mode Serial Message Protocol

Servo mode serial port sending and receiving message protocol is as follows:

Frame header（0x02）	Data length	Data Frame	Data bit	Check the high 8 bits	Check the lower 8 bits	End of frame（0x03）

Check digit calculation code refer to page 32

data frame definition

```
typedef enum {
        COMM_FW_VERSION = 0,
        COMM_JUMP_TO_BOOTLOADER,
        COMM_ERASE_NEW_APP,
        COMM_WRITE_NEW_APP_DATA,
        COMM_GET_VALUES,        //Get motor running parameters
        COMM_SET_DUTY,          //Motor runs in duty cycle mode
        COMM_SET_CURRENT,    //Motor runs in current loop mode
        COMM_SET_CURRENT_BRAKE, //Motor current brake mode operation
        COMM_SET_RPM,                   //Motor runs in current loop mode
        COMM_SET_POS,                   //Motor runs in position loop mode
        COMM_SET_HANDBRAKE,      //Motor runs in handbrake current loop mode
        COMM_SET_DETECT,            //Motor real-time feedback current position command
        COMM_ROTOR_POSITION=22,//Motor feedback current position
        COMM_GET_VALUES_SETUP=50,//Motor single or multiple parameter acquisition instructions
        COMM_SET_POS_SPD=91,         //  Motor runs in position velocity loop mode
        COMM_SET_POS_MULTI=92,          // Set the motor movement to single-turn mode
        COMM_SET_POS_SINGLE=93,    //  Set the motor motion to multi-turn mode, the range is ±100 turns
        COMM_SET_POS_UNLIMITED=94, //Save
        COMM_SET_POS_ORIGIN=95, //Set the motor origin
} COMM_PACKET_ID;
```

1. Obtain an example of motor parameters

Serial port command: 02 01 04 40 84 03 // Get the motor parameter command After the motor receives the motor feedback once the motor status

Example of motor return serial port command: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 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

// 02（frame header）+49（Data length）+ 04（Data Frame） + mos temperature（2byte）+motor temperature （2byte）+output current（4byte）+input current（4byte）+Id current（4byte）+Iq current（4byte）+Motor throttle value（2byte）+motor speed（4byte）+input voltage（2byte）+reserved value（24byte）+Motor status code（1byte）+Motor outer ring position value（4byte）+Motor Control ID Number（1byte）+Temperature retention value（6byte）+Vd voltage value（4byte）+Vqvoltage value（4byte）+CRC+03(end of frame)

The conversion formula of some parameters sent by the motor is 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))
Iinput voltage=(float)buffer_get_int16(data, &ind) / 10.0)
Motor outer ring position=(float)buffer_get_int32(data, &ind) / 1000000.0)
Motor ID number=data
Motor Vd voltage=(float)buffer_get_int32(data, &ind) / 1000.0)
Motor Vq voltage=(float)buffer_get_int32(data, &ind) / 1000.0)

Motor feedback position message command
Serial command：02 02 0B 04 9C 7E 03 // The motor sends the current position every 10ms after receiving

Example of sending the motor feedback position value (the feedback position message command needs to be sent to the motor in advance, and the current position is sent every 10ms after the motor receives it)
Serial command：02 05 16 00 1A B6 64 D5 F4 03
Pos=(float)buffer_get_int32(data, &ind) / 1000.0

Single or multiple parameter acquisition instruction instance of motor
Serial command: 02 05 32 00 00 00 01 58 4C 03 // Get motor temperature command
Instruction description: This instruction can acquire single or multiple motor parameters. The acquired parameters are determined by the 4 bytes of the data segment. When the

corresponding bit is 1, the motor will return the motor parameters of the corresponding bit. When it is 0, this field will be removed.

The motor parameters corresponding to the bits are as follows:

32-19 bit	18bit	17bit	16bit	15-10bit	9bit	8bit	7bit
reserved value	Motor ID（1byte）	Motor position（4byte）	Motor Error Flag (1byte)	reserved value	Input voltage（2byte）	Motor speed（4byte）	Duty Cycle (2byte)
6 bit	5bit	4bit	3bit	2 bit	1bit		
Iq current（4 byte）	Id current (4 byte)	input current (4 byte)	output current (4 byte)	Motor temperature (2 byte)	Mosfet temperature（2 byte）		

After the motor receives the command, it will send out the corresponding parameters

Example：02 03 32 00 81 2A 6C 03 // Feedback motor temperature

The conversion formula of some parameters sent by the motor is 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 NUMBER=data

Motor error status code

```
typedef enum {
	FAULT_CODE_NONE = 0,
	FAULT_CODE_OVER_VOLTAGE,// OVER VOLTAGE
	FAULT_CODE_UNDER_VOLTAGE,// UNDER_VOLTAGE
	FAULT_CODE_DRV,// DRIVE FAULT
	FAULT_CODE_ABS_OVER_CURRENT,// OVER_CURRENT
	FAULT_CODE_OVER_TEMP_FET,// MOS OVER TEMPERATURE
	FAULT_CODE_OVER_TEMP_MOTOR,//MOS 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 UNDRE VOLTAGE
	FAULT_CODE_BOOTING_FROM_WATCHDOG_RESET,//UNDREVOLTAGE
	FAULT_CODE_ENCODER_SPI,// SPI ENCODER FAULT
	FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE,//Encoder overrun
```

```
    FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE,//Encoder overrun
    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 1 fault
    FAULT_CODE_UNBALANCED_CURRENTS,// current unbalance

} mc_fault_code;
```

2. Examples of control commands:

Example of Duty Cycle Transmit Mode

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

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

Duty=(float)buffer_get_int32(data, &ind) / 100000.0) //4 bit data accepted /10000.0

Example of current loop transmit mode

Serial command：02 05 06 00 00 13 88 8B 25 03 // 5 A IQ 电流

Serial command：02 05 06 FF FF EC 78 E3 05 03 // - 5 A IQ 电流

Current=(float)buffer_get_int32(data, &ind) / 1000.0 //4 bit data accepted/1000.0

Example of brake current sending mode

Serial command：02 05 07 00 00 13 88 21 74 03 // 5A Braking current

Serial command：02 05 07 FF FF EC 78 49 54 03 // - 5A Braking current

I_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //4 bit data accepted/1000.0

Example of speed loop sending mode

Serial command：02 05 08 00 00 03 E8 2B 58 03 // 1000 ERPM Electrical speed

Serial command：02 05 08 FF FF FC 18 43 78 03 // - 1000 ERPM Electrical speed

Speed=(float)buffer_get_int32(data, &ind) //4 bit data accepted

Example of position loop sending mode

Serial command：02 05 09 0A BA 95 00 1E E7 03 //Motor turns to 180 degrees

Serial command：02 05 09 05 5D 4A 80 7B 29 03 //Motor turns to 90 degrees

Pos=(float)buffer_get_int32(data, &ind) / 1000000.0 //4 bit data accepted/1000000.0

Handbrake current sending mode example

Serial command：02 05 0A 00 00 13 88 00 0E 03 //5A HB current Electrical speed

Serial command：02 05 0A FF FF EC 78 68 2E 03 //5A HB current Electrical speed

HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //4 bit data accepted/1000.0

Handbrake current sending mode example

Serial command：02 05 0A 00 00 13 88 00 0E 03 //5A HB current Electrical speed

Serial command：02 05 0A FF FF EC 78 68 2E 03 //5A HB current Electrical speed
HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //4 bit data accepted/1000.0

Example of Sending Mode of Position Velocity Loop

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

/*
180° 5000ERPM Acceleration 30000/S
Parameters is position+velocity+acceleration */
Pos=(float)buffer_get_int32(data, &ind) / 1000.0) // 4 bit data accepted/1000.0
Speed=(float)buffer_get_int32(data, &ind) //4 bit data accepted
Acc_Speed=(float)buffer_get_int32(data, &ind)//4 bit data accepted

Example of setting multi-turn mode sending

Serial command：02 05 5C 00 00 00 00 9E 19 03 //Set the motor position loop as multi-turn operation mode ±100 turns

Setting the lap mode to send an example

Serial command：02 05 5D 00 00 00 00 34 48 03 //Set the motor position loop to single-turn operation mode 0-360 degrees

Set the current position to 0 to send the instance

Serial command：02 02 5F 01 0E A0 03 //Set the current position loop of the motor as the zero reference point of the position loop

Serial check：

```
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;
}
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
  };


//int16 data bits
void buffer_append_int16(uint8_t* buffer, int16_t number, int32_t *index) {
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//uint16 data bits
void buffer_append_uint16(uint8_t* buffer, uint16_t number, int32_t *index) {
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//int32data bits
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 bits
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;
}

//int64data bits
void buffer_append_int64(uint8_t* buffer, int64_t number, int32_t *index) {
    buffer[(*index)++] = number >> 56;
    buffer[(*index)++] = number >> 48;
    buffer[(*index)++] = number >> 40;
    buffer[(*index)++] = number >> 32;
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//uint64data bits
void buffer_append_uint64(uint8_t* buffer, uint64_t number, int32_t *index) {
    buffer[(*index)++] = number >> 56;
    buffer[(*index)++] = number >> 48;
    buffer[(*index)++] = number >> 40;
    buffer[(*index)++] = number >> 32;
    buffer[(*index)++] = number >> 24;
    buffer[(*index)++] = number >> 16;
    buffer[(*index)++] = number >> 8;
    buffer[(*index)++] = number;
}

//CRC calibration
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;
}

  //data sort and send
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 power mode communication protocol

Special Can code

Enter motor control mode {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }
Exit motor control mode {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFD}
Set the current position of the motor to 0 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFE }
Note: entering motor control mode before using CAN communication control motor!

If you want to read the current state when there is no state, the command sent is:
{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }）

MIT mode drive board receives data definition

Identifier: set motor ID (default: 1)　　Frame type: standard frame
Frame format: DATA　　DLC: 8 bytes

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

Data fields	DATA[4]	DATA[5]	DATA[6]		DATA[7]
Data bits	7-0	7-0	7-4	3-0	0-7
The data content	KP value 8 bit low	KD value 8 bit high	KD value 4 bit low	Current value 4 bit high	Current value 8 bit low

MIT power mode driver board sending data definition

Identifier: 0X00+ Drive ID Frame type: standard frame

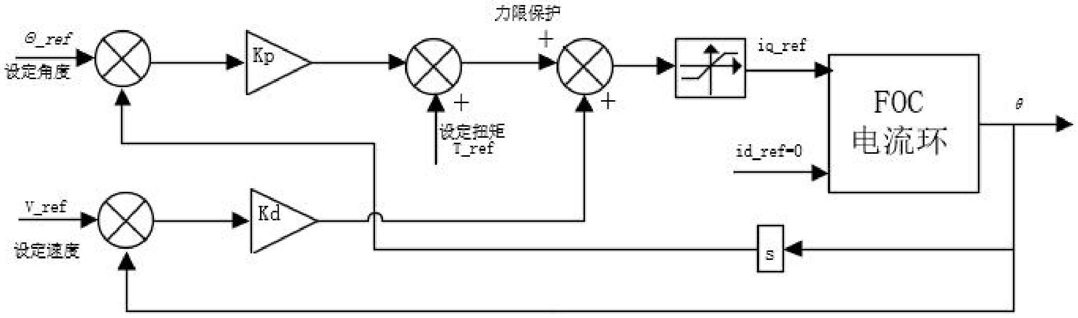
Frame format: DATA DLC: 6 bytes

Data fields	DATA[0]	DATA[1]	DATA[2]	DATA[3]	DATA[4]
Data bits	7-0	7-0	7-0	7-0	7-4
The data content	Driver board ID code	Motor position 8 bit higher	Motor position 8 bit lower	Motor speed 8 bit higher	Motor speed 4 bit lower

Data fields	DATA[4]	DATA[5]	DATA[6]	DATA[7]
Data bits	3-0	7-0	7-0	7-0
The data content	Current 4 bit higher	Current 8 bit lower	Motor temperature	An error code

CAN Speed: 1 MHZ

Simple block diagram of MIT power mode



Operation control mode send and receive code routines

Module	AK10-9	AK60-6	AK70-10	AK80-6	AK80-9	AK80-80/64
Position (rad)	-12.5f-12.5f					
Speed (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

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
Kp range	0-500					
Kd range	0-5					

Sends routine code

```
void pack_cmd(CANMessage * msg, float p_des, float v_des, float kp, float kd, float t_ff){
/// limit data to be within bounds ///
float P_MIN =-95.5;
float P_MAX =95.5;
float V_MIN =-30;
float V_MAX =30;
float T_MIN =-18;
float T_MAX =18;
float Kp_MIN =0;
float Kp_MAX =500;
float Kd_MIN =0;
float Kd_MAX =5;
float Test_Pos=0.0;
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);
/// convert floats to unsigned ints ///
int p_int = float_to_uint(p_des, P_MIN, P_MAX, 16);
int v_int = float_to_uint(v_des, V_MIN, V_MAX, 12);
int kp_int = float_to_uint(kp, KP_MIN, KP_MAX, 12);
int kd_int = float_to_uint(kd, KD_MIN, KD_MAX, 12);
int t_int = float_to_uint(t_ff, T_MIN, T_MAX, 12);
/// pack ints into the can buffer ///
msg->data[0] = p_int>>8;                //  Position 8 higher
msg->data[1] = p_int&0xFF;              //  Position 8 lower
msg->data[2] = v_int>>4;                // Speed 8 higher
msg->data[3] = ((v_int&0xF)<<4)|(kp_int>>8);  //
Speed 4 bit lower KP 4bit higher
msg->data[4] = kp_int&0xFF;             // KP 8 bit lower
msg->data[5] = kd_int>>4;               // Kd 8 bit higher
msg->data[6] = ((kd_int&0xF)<<4)|(kp_int>>8);      //
                                                KP 4 bit lower torque 4 bit higher
```

```
        msg->data[7] = t_int&0xff;              // torque 4 bit lower

        }
```

When sending packets, all the numbers should be converted into integer numbers by the following functions and then 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

```
        void unpack_reply(CANMessage msg){
        /// unpack ints from can buffer ///
        int id = msg.data[0]; //驱动 ID 号
        int p_int = (msg.data[1]<<8)|msg.data[2];           //Motor position data
        int v_int = (msg.data[3]<<4)|(msg.data[4]>>4);      // Motor speed data
        int i_int = ((msg.data[4]&0xF)<<8)|msg.data[5];     // Motor torque data

        /// 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, -I_MAX, I_MAX, 12);
                if(id == 1){
postion = p;                            //
                                        Read the corresponding data according to the ID code
speed = v;
torque = i;
                }
        }
```

All numbers 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;
}
```