AK Series Module Driver User Manual

V1.0.14

Table of Contents

Precautions

1.Ensure that the circuit is free of short circuits, and connect the interfaces as required.

2. When the driver board is outputting, heating may occur. Please use caution to avoid burns.

3. Before use, check whether all components are intact. In case of missing or aging components, please stop using and contact technical support promptly.

4. Multiple optional control modes cannot be switched while the driver board is running, and the communication protocols between different control modes are different. If switching is needed, restart the power supply and then make changes. Using the wrong protocol control may result in the burning of the driver board!

5. Strictly adhere to the working voltage, current, temperature, and other parameters specified in this document; otherwise, it may cause permanent damage to the product!

Product Features

The AK series motor driver boards use high-performance driver chips within the same level, employing the Field Oriented Control (FOC) algorithm. They are paired with advanced self-disturbance control technology to control speed and angle. The boards can be configured and firmware upgraded using the CubeMars Tool tuning software.

Disclaimer

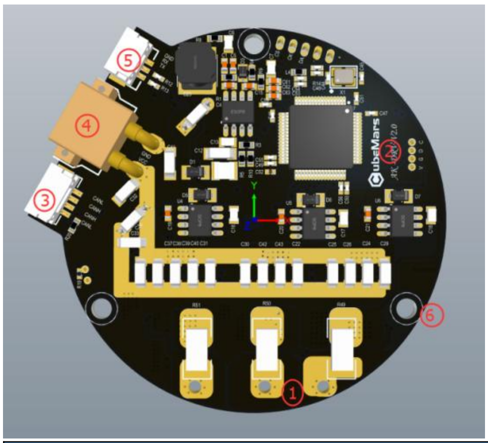
Thank you for purchasing the AK series Actuator. Before use, please carefully read this disclaimer. Once used, it is deemed an acceptance of the entire content of this disclaimer. Strictly adhere to the product manual and relevant laws, regulations, policies, and guidelines for installing and using this product. CubeMars will not assume legal responsibility for any losses caused by improper use, installation, or modification by the user.

Version Change Log

DATE	VERSION	CHANGE CONTENT
2021.09.01	Ver. 1.0.0	Created version
2021.10.08	Ver.1.0.1	Changes in 5.1 and 5.2 codes
2021.10.29	Ver.1.0.2	Updates in data definitions for 5.1, 5.2, and 5.3
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	Additional information in 5.3
2022.01.20	Ver.1.0.6	Speed change for AK60-6 motor in 5.3
2023.07.19	VER.1.0.10	1. Explanation of red light indications 2. Addition of parameters for 80-8 60KV MIT
2023.08.29	VER.1.0.12	1.Correction of position velocity loop routine code 2.Modification of byte order notation in servo mode
2023.12.11	VER.1.0.13	Improvement in error reporting, modifications in origin mode and transmission code
2023.12.28	VER.1.0.13	1.Re-translated by Randy according to Version 1.0.13. 2. 5.1.6 Origin Mode Code Modification 3. 5.1.7 Position-speed Mode Code Modification Revision of Section 5.1 Position Loop Velocity Loop Description 4. Addition of Video Links and Descriptions to 4.2.1, 4.2.2, and 4.4

1. Driver Information

1.1 Driver Appearance Introduction & Specifications



① Three-Phase Power Line Connection Terminal

② Hardware Version Number

③ CAN Communication Connection Port

④ DC Power Interface

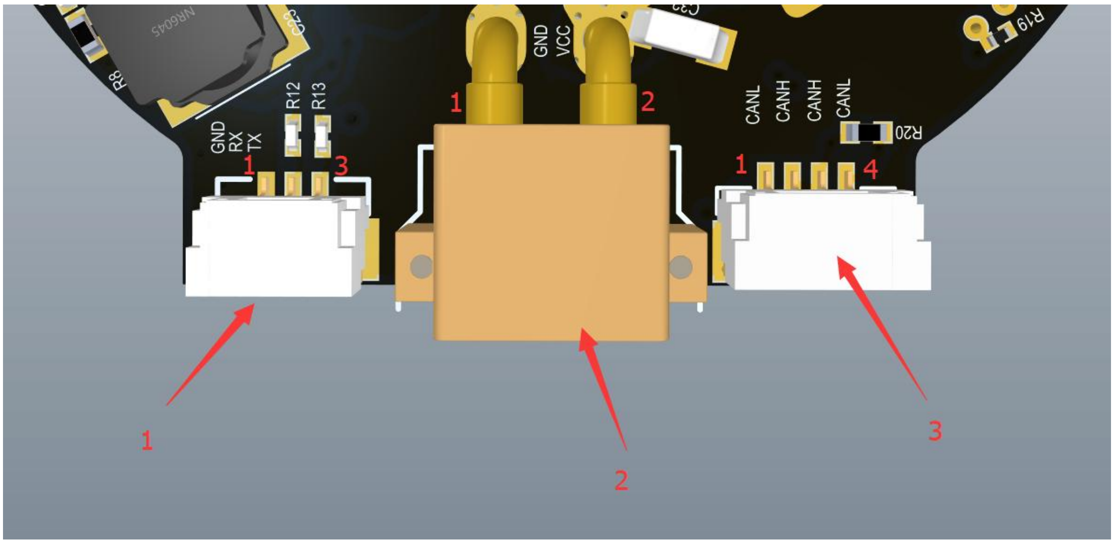
⑤ Serial Communication Connection Port

⑥ Mounting Holes

Specifications	
Rated Operating Voltage	48V
Maximum Allowed Voltage	52V
Rated Operating Current	20A
Maximum Allowed Current	60A
Standby Power Consumption	≤50mA
CAN Bus Bit Rate	1Mbps (Not recommended to change)
Dimensions	62mm×58mm
Operating Environment Temperature	-20℃ to 65℃
Maximum Allowable Temperature of the Driver	100℃ (Over-temperature protection)
Encoder Accuracy	14bit（Single-turn absolute value）

1.2 Driver Interface and Definitions

1.2.1 Driver Interface Diagram



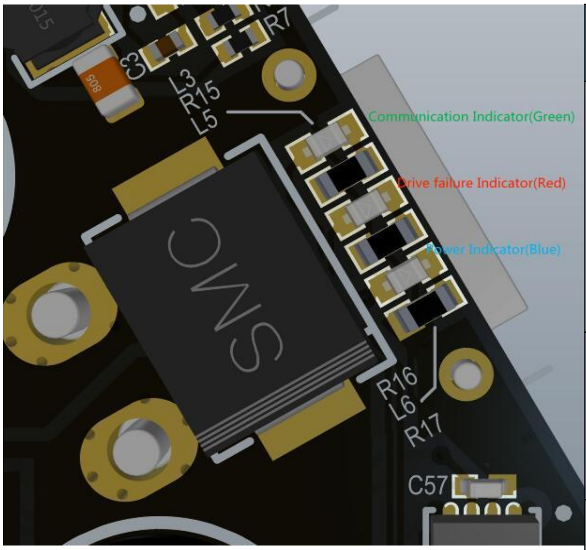
1.2.2 Driver Interface Recommended Brands and Models

No.	Onboard Interface Model	Brand	Line-end Interface Model	Brand
1	A1257WR-S-3P	CJT(Changjiang Connector)	A1257H-3P	CJT(Changjiang Connector)
2	XT30PW-M	AMASS(AMASS)	XT30UPB-F	AMASS(AMASS)
3	A1257WR-S-4P	CJT(Changjiang Connector)	A1257H-4P	CJT(Changjiang Connector)

1.2.3 Driver Interface Pin Definitions

NO.	Interface Function	Pin	Description
1	**Serial Communication**	1	Serial signal ground (GND)
		2	Serial signal output (TX)
		3	Serial signal input (RX)
2	**Power Input**	1	Power negative pole (-)
		2	Power 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 Driver Indicator Light Definitions



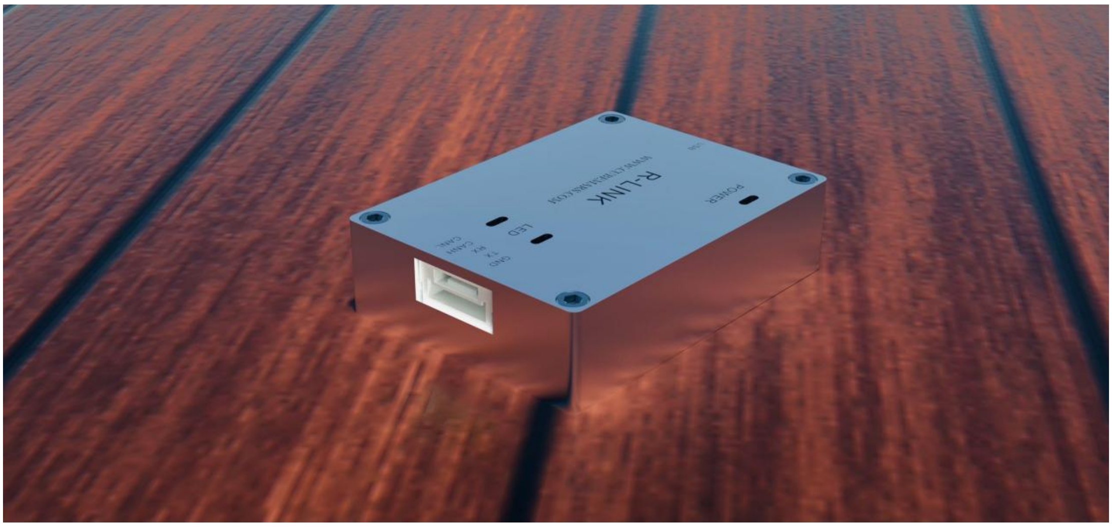
Indicator Light Definitions	
1.Power Indicator Light (Blue when lit)	This indicates the power status of the driver board. Under normal circumstances, the blue light will be lit when the power is inserted. If the blue light does not come on when the power is inserted, please immediately remove the power and do not power it up again.
2.Communication Indicator Light (Green when lit)	This light indicates the communication status of the driver board. The green light will only be lit when the driver board is communicating normally. If the green light is not lit, please check the CAN communication wiring first.
3.Driver Fault Indicator Light (Red when lit)	This light is used to indicate the fault status of the driver board. Under normal circumstances, the red light will only be lit when there is a fault with the driver board. Typically, it remains off during normal operation. When the driver fault indicator light is lit, it indicates that the driver board has suffered some damage. In such a case, power should be turned off immediately, and no further operation should be attempted. (A slight glow of the red light when the driver board is powered on is normal.)

1.4 Main Accessories and Specifications

No.	Item	Specification		Quantity	Remarks
1	Serial Cable	Wire	24AWG-300MM-PTFE-Silver Plated Wire-Black Yellow Green	1 Each	±2MM
		Connector	A1257H-3P	1PC	
			A2541H-3P	1PC	
2	Power Cable	Wire	16AWG-200MM-Silicone Wire-Red Black	1 Each	±2MM
		Connector	XT30UPB-M	1PCS	
			XT30UPB-F	1PCS	
3	CAN Communication Cable	Wire	24AWG-300MM-PTFE-Silver Plated Wire-White Blue	1 Each	±2MM
		Connector	A1257H-4P	2PCS	
			A2541H-2P	1PC	
4	Thermistor	MF51B103F3950-10K-3950		2PCS	
5	Electrolytic Capacitor	120Uf-63V-10x12MM		2PCS	Included with AK10-9 V2.0 and above
6	Power MOS	BSC026N08NS5-80V-2.6mΩ TPH2R608NH-75V-2.6mΩ		12PCS	Random

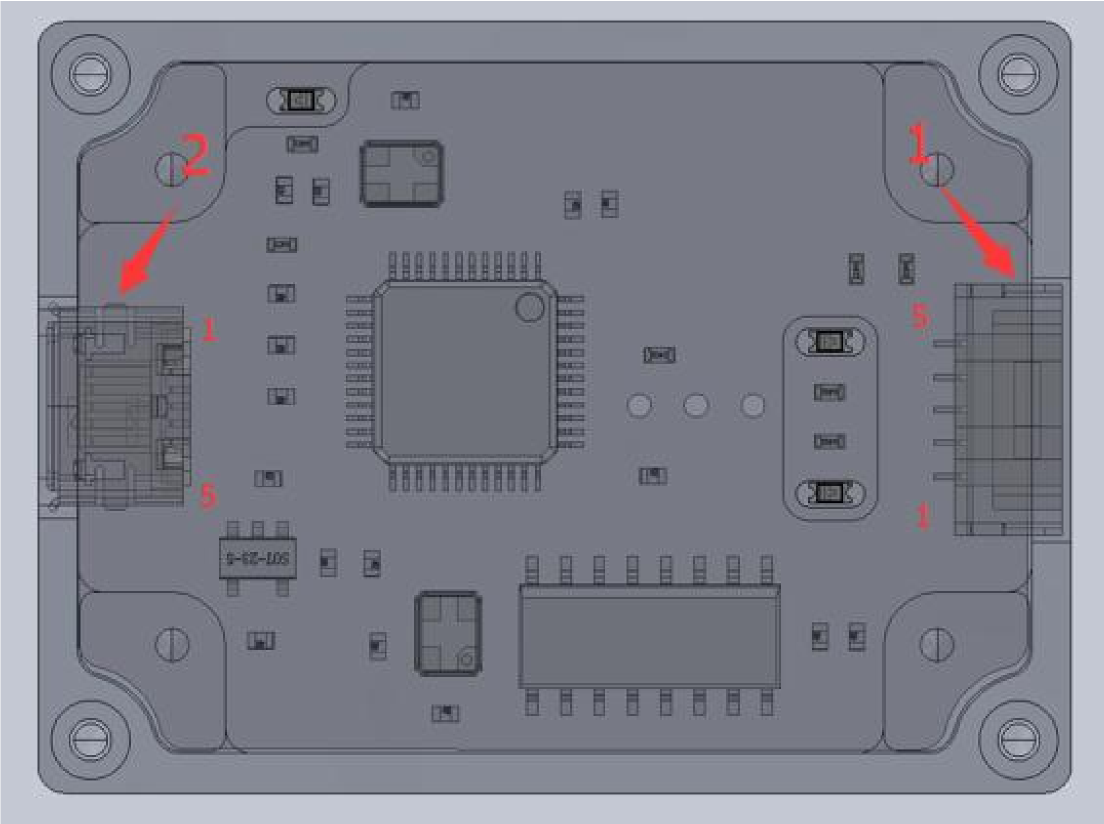
2. R-link Information

2.1 R-link Appearance Introduction & Specifications



Product Specifications	
Rated Operating Voltage	5V
Standby Power Consumption	≤30mA
Dimensions	39.2x29.2x10MM
Operating Environment Temperature	-20℃ to 65℃
Maximum Allowable Temperature of the Driver	85℃

2.2 R-link Interface and Definitions

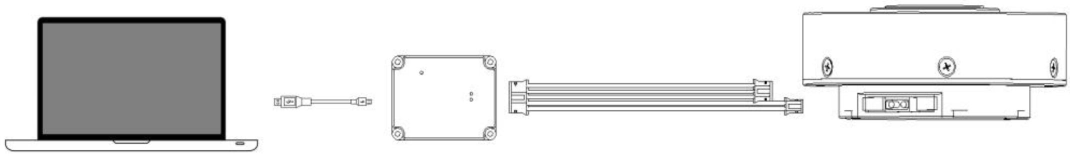


No.	Interface Function	Pin	Description
		1	CAN communication low side (CAN_L)
		2	CAN communication high side (CAN_H)
1	Communication Interface	3	Serial signal input (RX)
		4	Serial signal output (TX)
		5	Serial signal ground (GND)
		1	VBUS
		2	D-
2	USB Interface	3	D+
		4	ID
		5	GND

2.3 R-link Indicator Light Definitions

No.	Color Definition	Description
1	Green	Power Indicator Light, indicates R-link power status. The green light will be lit when the power is inserted under normal circumstances. If the green light is not lit when the power is inserted, please immediately remove the power and do not power it up again.
2	Blue	Serial Communication Output (TX), normally off. It blinks when there is data output from the R-link serial port.
3	Red	Serial Communication Input (RX), normally off. It blinks when there is data input into the R-link serial port.

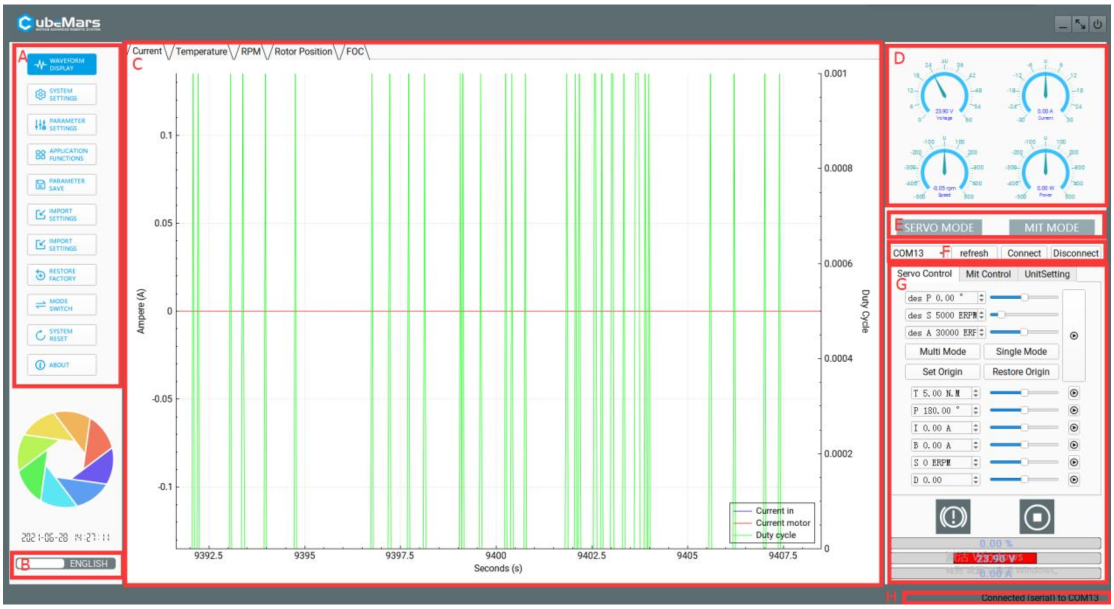
3. Connection of the Driver and R-link and Precautions



USB Cable on R-Link	---->	PC End
5-Pin Connector	---->	R-Link 5-Pin Port
4-Pin Terminal (CAN Port)	---->	4-Pin Port on the Motor (CAN)
3-Pin Terminal (UART Port)	---->	3-Pin Port on the Motor (UART)

4. Upper Computer Usage Instructions

4.1 Upper Computer Interface and Explanation



A. Main Menu Bar

B. Chinese/English Switch

C. Main Page

D. Real-Time Data Display

E. Current Mode

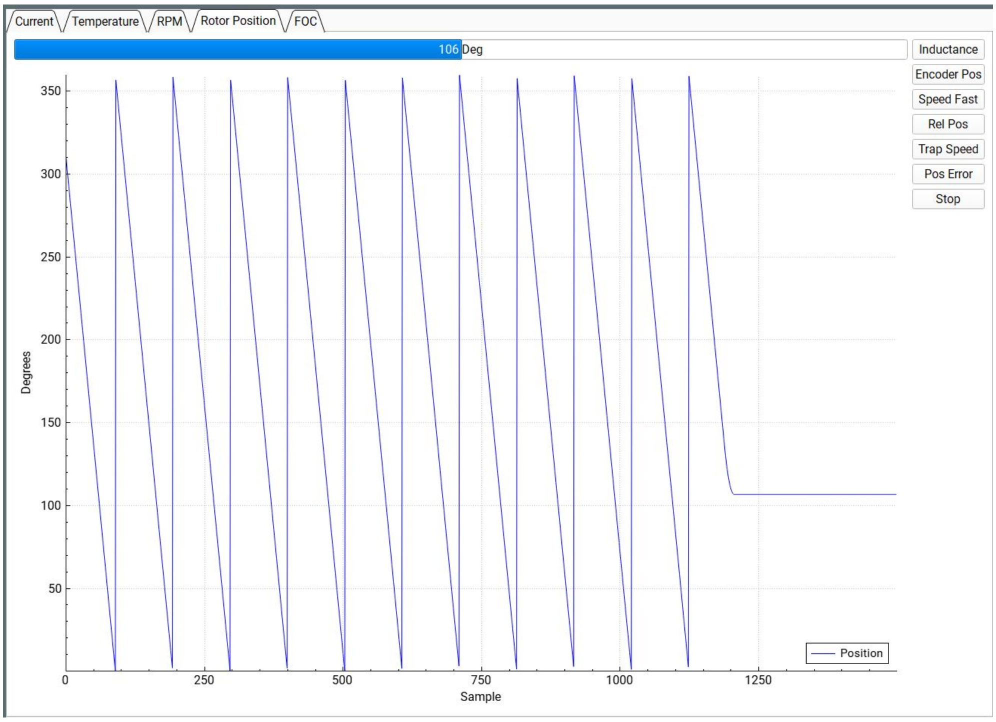
F. Serial Port Selection

G. Control Parameters

4.1.1 Main Menu Bar

4.1.1.1 Waveform Display

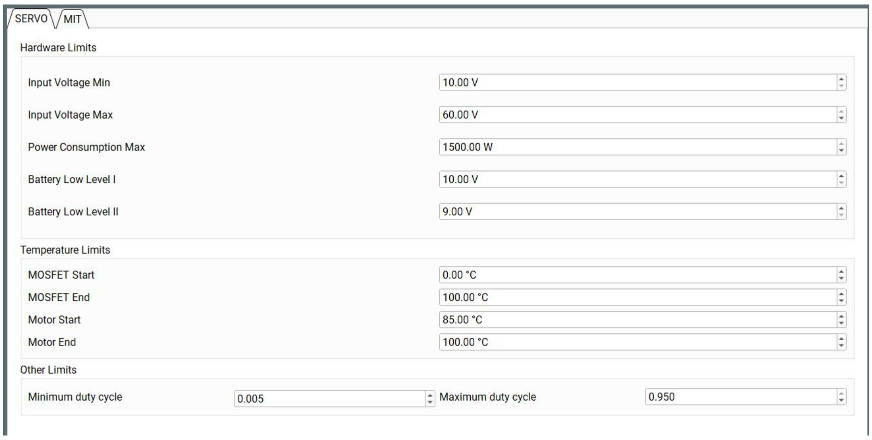
This page supports viewing real-time data feedback and drawing graphs. Data includes motor current, temperature, real-time speed, internal encoder position, external encoder position, high-frequency speed, rotor position, path planning, position deviation, DQ current, etc.



4.1.1.2 System Settings

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

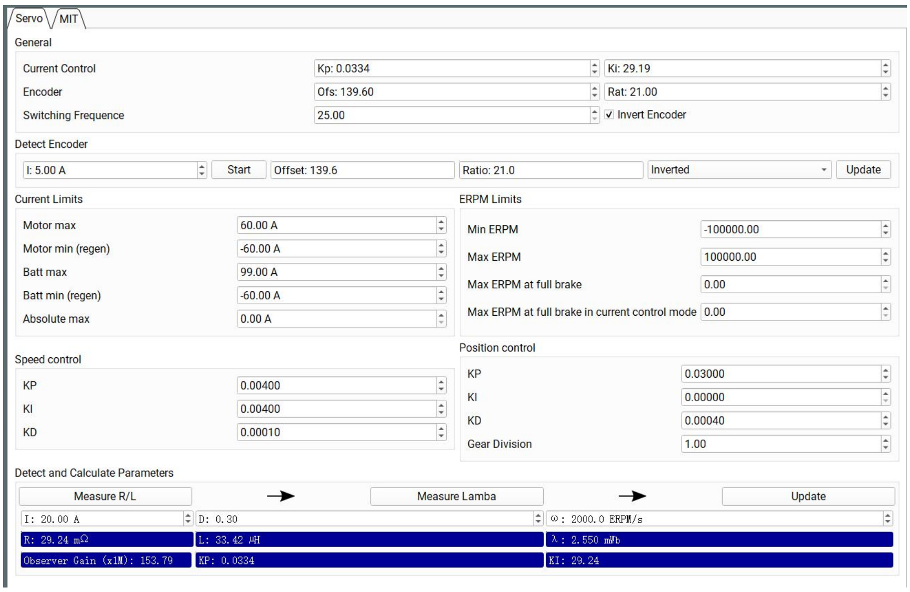
⚠： Be sure to strictly follow the specified voltage, current, power, and temperature usage. Any injuries caused to humans or irreversible damage to the driver board and motor due to improper operation of this product will not be the responsibility of the company.



4.1.1.3 Parameter Settings

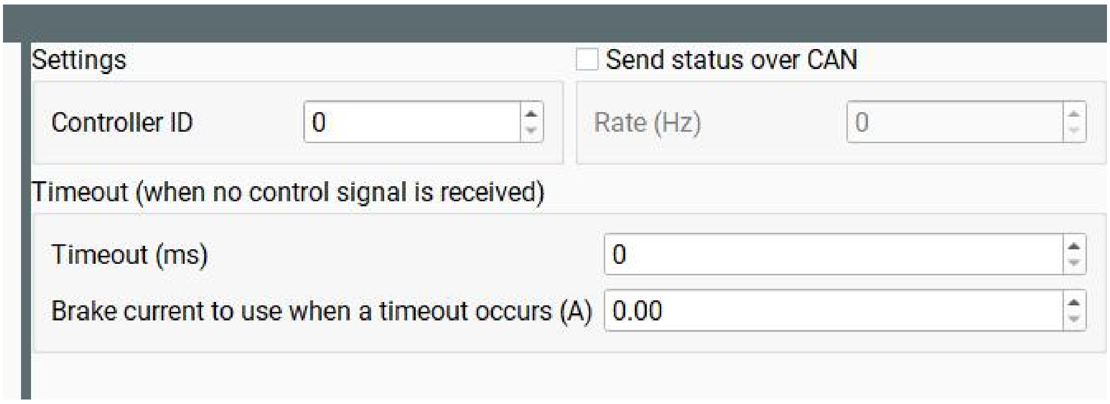
This page is mainly for adjusting driver board parameters, including but not limited to current loop Kp -Ki, encoder bias, current maximum and minimum values, speed maximum and minimum values, speed loop Kp-Ki-KD, reduction ratio, and calibration of the encoder, and motor parameter tuning

⚠： Be sure to strictly follow the specified voltage, current, power, and temperature usage. Any injuries caused to humans or irreversible damage to the driver board and motor due to improper operation of this product will not be the responsibility of the company.



4.1.1.4 Application Functions

This page is mainly for CAN ID settings, CAN communication rate, and settings for sudden CAN communication interruption.



4.1.1.5 Read Parameter (Important)

Save the current motor parameters to the upper computer.

⚠： Whenever rewriting motor parameters, be sure to click this button first. Otherwise, other motor parameters may be incorrect. If such a situation occurs, please download the default APP parameters for the corresponding motor from the official website and write them into the motor through "Import Settings."

4.1.1.6 Write Parameters

Save the parameters from the upper computer to the motor.

4.1.1.7 Export Settings

Save the upper computer parameters as two files with the suffix ".McParams" and ".AppParams," and save them to the computer.

Where ".McParams" file is: 

Where ".AppParams" file is: 

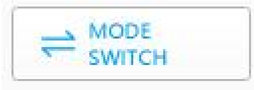
4.1.1.8 Import Settings

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

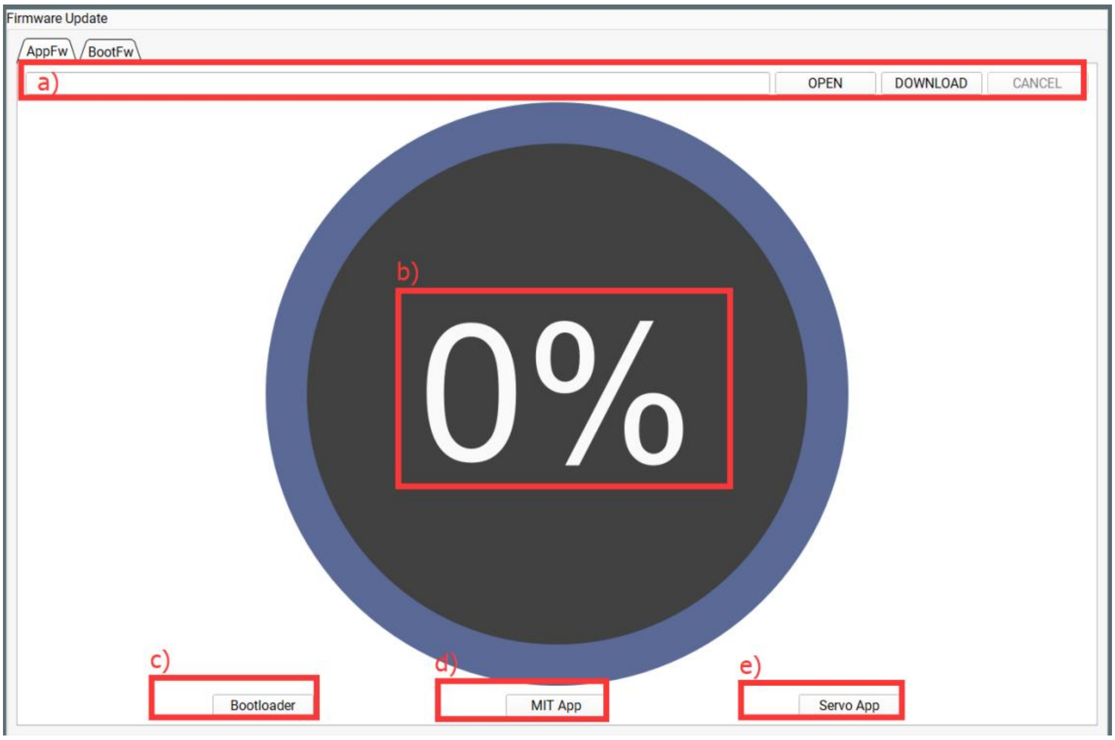
4.1.1.9 Restore to Factory Settings

This function is currently not available.

4.1.1.10 Mode Switch



This page is mainly for switching the control modes of the driver board, including "Bootloader Mode," "Servo Mode," and "MIT Mode."
It also includes firmware updates.



- a) Import Firmware Area: Can import a ".bin" file from the computer.
- b) Firmware Update Progress Bar
- c) Enter Bootloader Mode
- d) Enter MIT Mode
- e) Enter Servo Mode

4.1.1.11 System Reset

Stop the motor and restart it.

4.1.1.12 About

Displays the current version of the upper computer and the official homepage.

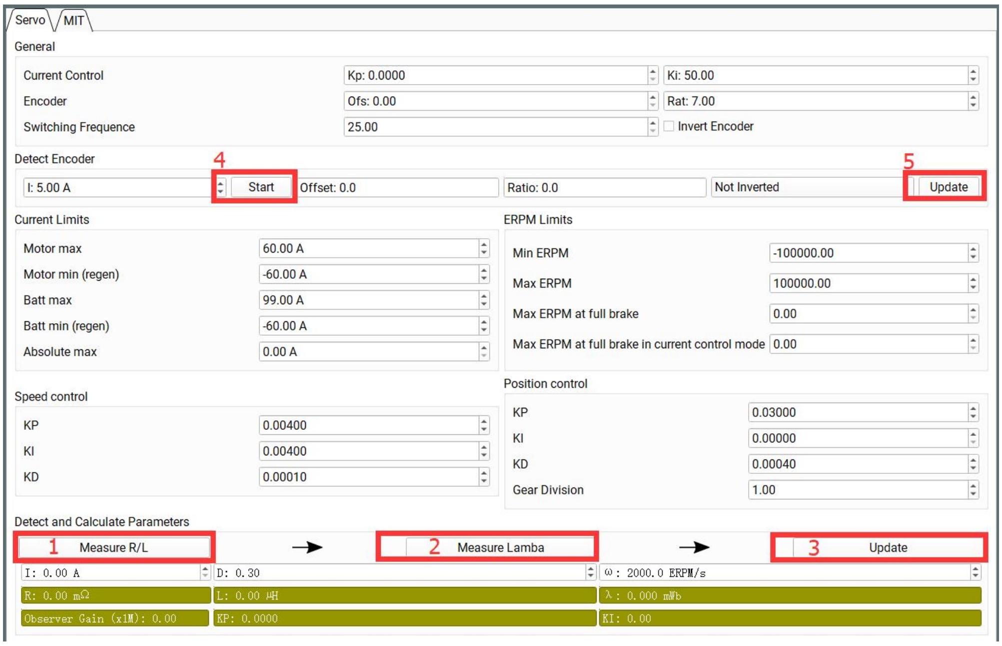
4.2 Driver Board Calibration

When you reinstall the driver board on the motor, change the phase sequence of the motor's three-phase lines, or update the firmware, calibration must be performed. After calibration, the motor can be used normally.

4.2.1 Servo Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, enter the system settings page, click "Measure R/L," "Measure Lamba," "Update," "Start," and finally "Update."

⚠:For your convenience, kindly follow the step-by-step instructions provided in the video to avoid any inadvertent errors:
Servo Mode Calibration:
https://www.youtube.com/watch?v=_Cj5eYb2aw8&t=249s

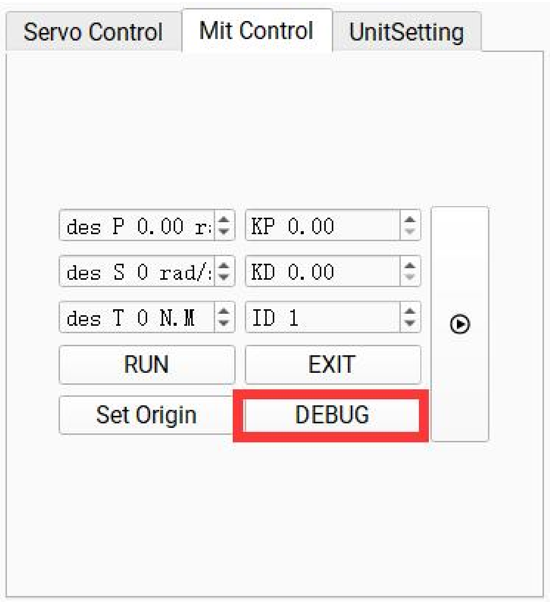


4.2.2 MIT Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in MIT mode. After successfully connecting to the upper computer, click "Debug" on the "MIT" page. Then, enter "calibrate" in the input field, wait for about 30 seconds, and the output field will scroll the position value of the encoder in real-time. When the output field prints "Encoder Electrical Offset (rad)," the motor will automatically restart, and the serial port will print driver information. During calibration, the voltage is approximately 1A at 48V. After calibration, the current returns to around 0.02A.

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

MIT Mode Calibration (9:53): https://www.youtube.com/watch?v=Y6BMy1lSnvA

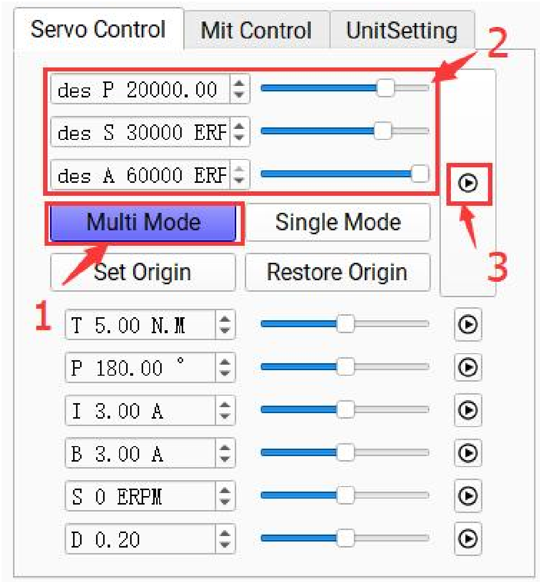


4.3 Control Demonstration

4.3.1 Servo Mode

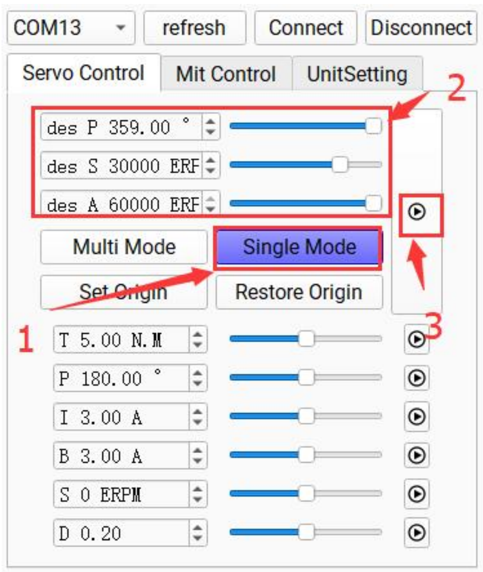
4.3.1.1 Multi Mode Position-Speed Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, click "Multi Mode" on the "Servo Control" page. Enter the desired position (at this time, the position is ±100 turns, i.e., -36000° to 36000°), the desired speed, and acceleration. The motor will move at the desired speed with the desired acceleration until the desired position is reached.



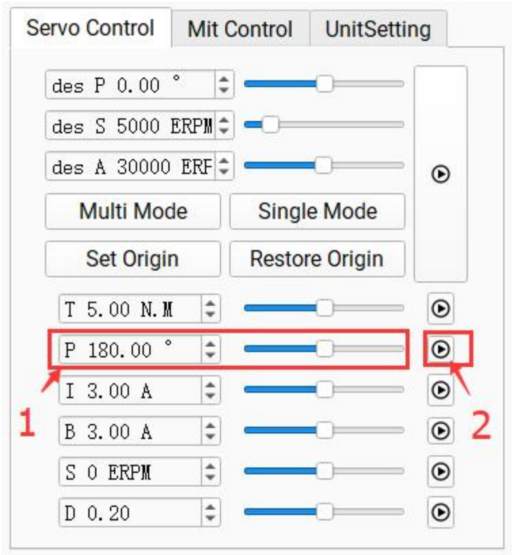
4.3.1.2 Single Mode Position-Speed Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, click "Single Mode" on the "Servo Control" page. Enter the desired position (at this time, the range of position is only one turn, i.e., 0° to 359°), the desired speed, and acceleration. The motor will move at the desired speed until the desired position is reached.



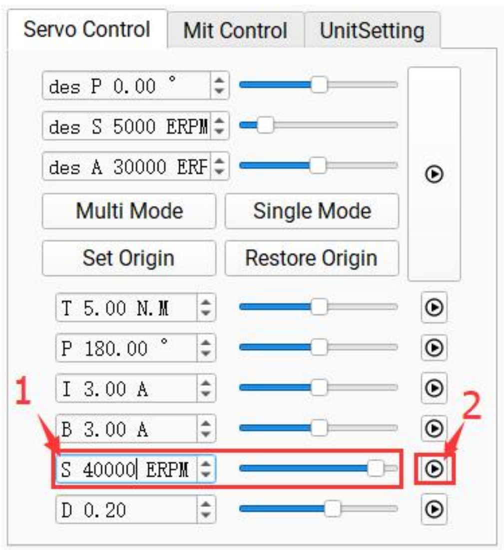
4.3.1.3 Position Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, enter the desired position on the "Servo Control" page, and the motor will reach the desired position at the maximum speed.



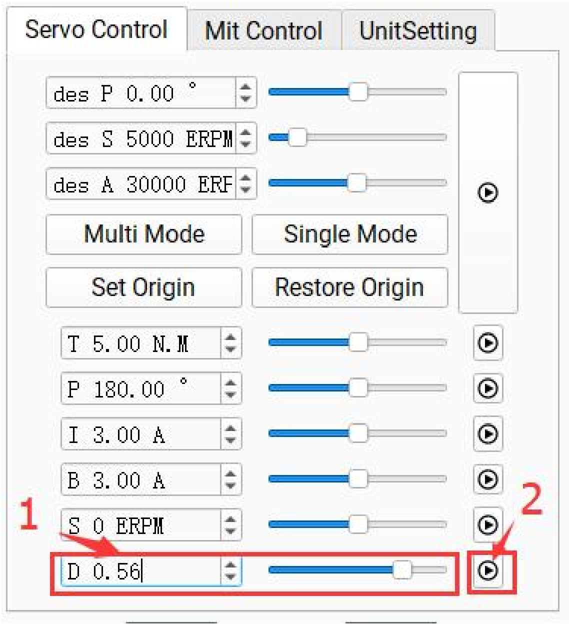
4.3.1.4 Speed Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, enter the desired speed (±50000ERPM) on the "Servo Control" page, and the motor will move at the desired speed.



4.3.1.5 Duty Cycle Mode

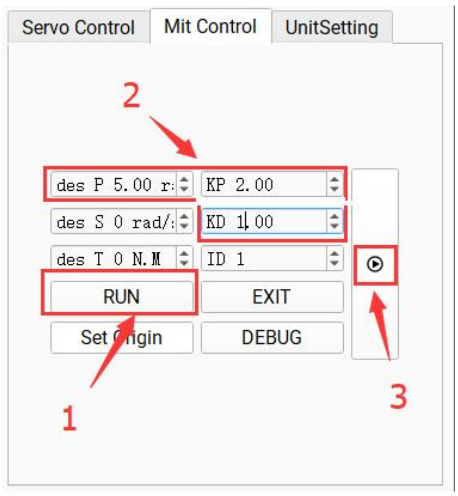
Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in servo mode. After successfully connecting to the upper computer, enter the desired duty cycle (default 0.005-0.95) on the "Servo Control" page. The motor will move at the specified duty cycle.



4.3.2 MIT Mode

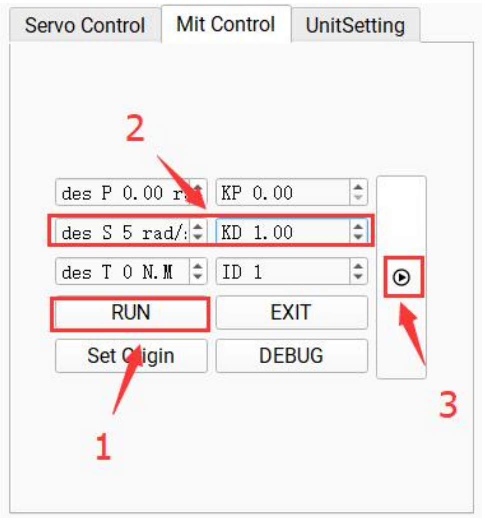
4.3.2.1 Position Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in MIT mode. After successfully connecting to the upper computer, enter the corresponding "CAN ID" on the "MIT Control" page. Then click "Run" to enter the motor mode. ***Enter the desired position, KP, and KD.*** The motor will perform position movement (default speed 12000 ERPM, acceleration 40000 ERPM).



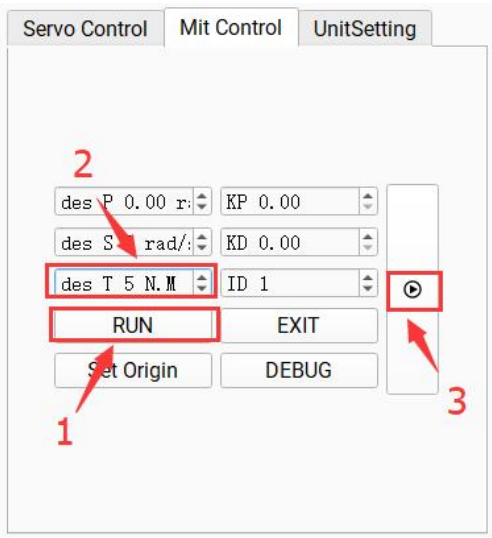
4.3.2.2 Speed Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in MIT mode. After successfully connecting to the upper computer, enter the corresponding "CAN ID" on the "MIT Control" page. Then click "Run" to enter the motor mode. ***Enter the desired speed and KD.*** The motor will perform speed movement.



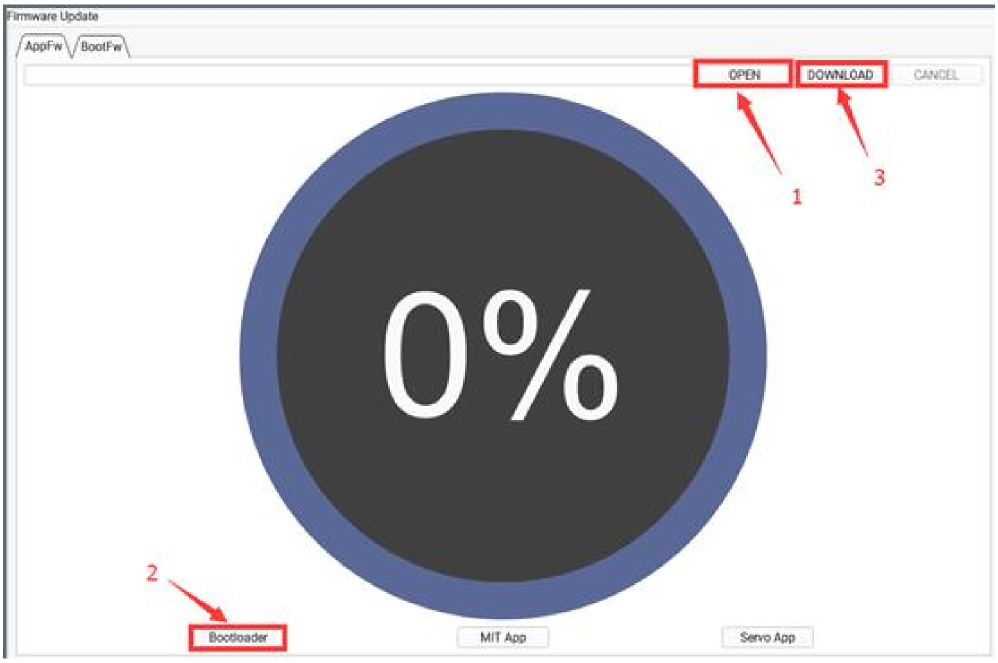
4.3.2.3 Torque Mode

Ensure that the motor input power is stable, R-LINK connection is normal, and the motor is in MIT mode. After successfully connecting to the upper computer, enter the corresponding "CAN ID" on the "MIT Control" page. Then click "Run" to enter the motor mode. Enter the desired torque, and the motor will perform torque movement.



4.4 Firmware Update

Step 1. Click "Open," select the firmware file with the ".BIN" suffix.
Step 2. Click "Bootloader."
Step 3. Click "Download," wait for the progress bar to complete to 100%, restart the power, and the firmware update is complete.



⚠:For your convenience, kindly follow the step-by-step instructions provided in the video to avoid any inadvertent errors:
Firmware installation and calibration:
https://www.youtube.com/watch?v=_Cj5eYb2aw8&t=251s
Please note that in this video demonstration, only the firmware for the servo mode has been uploaded. If your firmware package includes both servo and MIT firmware, please upload these two firmwares separately to ensure the proper functioning of both modes.
Additionally, if the firmware upload progress bar appears stuck and unresponsive, please follow these steps:

Step 1: Ensure normal power supply and connection.
Step 2: Go to the Mode Switch interface, click the "open" button, and select the firmware for your motor.
Step 3: Continuously click the bootloader button. Meanwhile, with your other hand, turn off the power and then turn it back on.

After performing these steps, you should see the progress bar start moving. Once the firmware reinstallation is complete, along with default parameter import and calibration actions, your motor should function normally.

5. Driver Board Communication Protocol and Explanation

5.1 Servo Mode Control Modes and Explanation

Servo mode has 6 control modes:

Duty Cycle Mode: Specifies the motor's duty cycle voltage in a square wave driving form.

Current Loop Mode: Specifies the Iq current of the motor. As the motor output torque = Iq * KT, it can be used as a torque loop.

Current Brake Mode: Specifies the braking current of the motor to fix the motor at the current position (pay attention to motor temperature during use).

Speed Mode: Specifies the running speed of the motor.

Position Mode: Specifies the position of the motor, and the motor will run to the specified position at the maximum speed.

Position-Speed Loop Mode: Specifies the position, speed, and acceleration of the motor. The motor will run to the specified position with the given acceleration and maximum speed.

The servo protocol uses the CAN protocol with an extended frame format. The format is as follows:

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

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

Duty Cycle Mode: 0

Current Loop Mode: 1

Current Brake Mode: 2

Speed Mode: 3

Position Mode: 4

Set Origin Mode: 5

Position-Speed Loop Mode: 6

The following provides examples of controlling the motor in various modes:

The following are examples of invoking 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,                   // Speed Mode
      CAN_PACKET_SET_POS,                   // Position Mode
      CAN_PACKET_SET_ORIGIN_HERE,           // Set Origin Mode
      CAN_PACKET_SET_POS_SPD,               // Position-Speed 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;
for(i=0;i<len;i++)
		TxMessage.Data[i]=data[i];
	CAN_Transmit(CHASSIS_CAN, &TxMessage); //CAN port sends 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 data transmission definition

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

```
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 data transmission definition

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

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

Example for Current Loop Mode Transmission:

```
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 data transmission definition

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

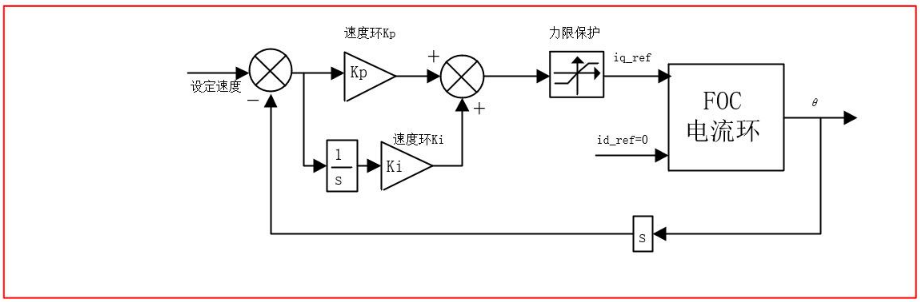
The brake current value is of type int32, and the value 0-60000 represents 0-60A.

Example for Current Brake Mode Transmission:

```
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 Speed Loop Mode

Speed loop simplified control block diagram



Speed Loop mode data transmission definition

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

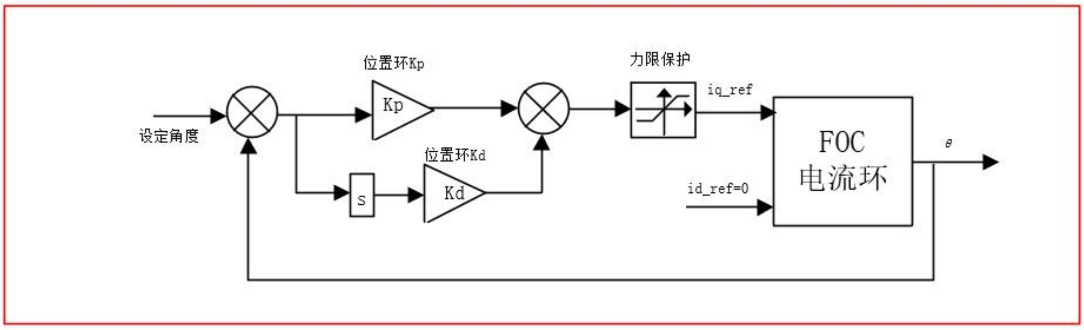
The speed value is of type int32, and the range is -100000-100000, representing -100000-100000 electrical RPM.

Example for Speed Loop Mode Transmission

```
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 simplified control block diagram



Position Loop mode data transmission definition

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

The position value is of type int32, and the range is -360000000-360000000, representing -36000°-36000°.

Example for Position Loop Mode Transmission:

```
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 bits	Data[0]
Range	0~0x02
Variables	Set Instruction

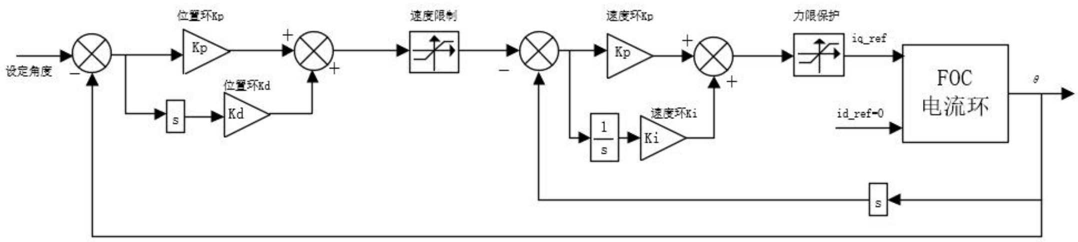
The setting command is of type uint8_t, where 0 represents setting a temporary origin (cleared after power-off), and 1 represents setting a permanent zero point (parameters automatically saved).

Example for Set Origin Mode Transmission

```
void comm_can_set_origin(uint8_t controller_id, uint8_t set_origin_mode) {
	int32_t send_index = 0;
	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-Speed Loop Mode

Position-Speed loop simplified control block diagram



Position-Speed Loop mode data transmission 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
Variables	Position 25-32 bits	Position 17-24 bits	Position 9-16 bits	Position 1-8 bits	Speed High 8 bits	Speed Low 8 bits	Acceleration High 8 bits	Acceleration Low 8 bits

- Position: int32, range -360000000~360000000 representing -36000°~36000°.
- Speed: int16, range -32768~32767 representing -327680~-327680 electrical RPM.
- Acceleration: int16, range 0~32767, representing 0~327670, 1 unit equals 10 electrical RPM/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 = 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 Message Formats

5.2.1 Servo Mode CAN Upload Message Protocol

In servo mode, the motor CAN message uses periodic upload mode, and the upload frequency can be set to 1-500Hz, with an upload size of 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
Variables	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

- ***Position:*** int16, range -32000~32000 represents -3200°~3200°.
- ***Speed:*** int16, range -32000~32000 represents -320000~320000 electrical RPM.
- ***Current:*** int16, range -6000~6000 represents -60~60A.
- ***Motor Temperature:*** int8, range -20~127 represents the driver board temperature -20°C~127°C.
- ***Error Code:*** uint8, 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.

Example of Receiving Message:

```
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 Code
  }
```

5.2.2 Servo Mode Serial Message Protocol

The protocol for servo mode serial communication is as follows:

Frame Head (0x02)	Data Length (excluding frame head, frame tail, and checksum)	Data Frame	Data	Checksum High 8 bits	Checksum Low 8 bits	Frame Tail (0x03)

Checksum Bit Calculation Code Reference Chapter Five

Data Frame Definitions:

```
typedef enum {
COMM_FW_VERSION = 0,
COMM_JUMP_TO_BOOTLOADER,
COMM_ERASE_NEW_APP,
COMM_WRITE_NEW_APP_DATA,
COMM_GET_VALUES,       // Get motor operating parameters
COMM_SET_DUTY,         // Motor operates in duty cycle mode
COMM_SET_CURRENT,    // Motor operates in current loop mode
COMM_SET_CURRENT_BRAKE, // Motor operates in current brake mode
COMM_SET_RPM,                // Motor operates in speed loop mode
COMM_SET_POS,                // Motor operates in position loop mode
COMM_SET_HANDBRAKE,      // Motor operates 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 command
COMM_SET_POS_SPD=91,        // Motor operates in position-speed loop mode
COMM_SET_POS_MULTI=92,      // Set motor motion to single-turn mode
COMM_SET_POS_SINGLE=93,    // Set motor motion to multi-turn mode, range ±100 turns
COMM_SET_POS_UNLIMITED=94, // Reserved
COMM_SET_POS_ORIGIN=95, // Set motor origin
} COMM_PACKET_ID;
```

I. Example for Getting Motor Parameters

Serial Command: 02 01 04 40 84 03 // Get motor parameters command, the motor responds once with the motor status

Motor Serial Response Example：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 Head) + 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) + Throttle Value (2 bytes) + Motor Speed (4 bytes) + Input voltage (2 bytes) + Reserved (24 bytes) + Motor Status Code (1 byte) + Motor Outer Loop Position Value (4 bytes) + Motor ID Number (1 byte) + Temperature Reserved Value (6 bytes) + Vd Voltage (4 bytes) + Vq Voltage (4 bytes) + CRC + 03 (Frame Tail)

Conversion formulas for motor parameters:

```
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;
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) / 1000000.0;
Motor ID Number = data;
Vd Voltage = (float)buffer_get_int32(data, &ind) / 1000.0;
Vq Voltage = (float)buffer_get_int32(data, &ind) / 1000.0;
```

Motor feedback position command

Serial command: 02 02 0B 04 9C 7E 03 // Motor sends current position every 10 ms after receiving this command

Example of motor feedback position value transmission (Prior to this, send a feedback position command to the motor. After the motor receives it, it will send the current position every 10 milliseconds.)

Serial command: 02 05 16 00 1A B6 64 D5 F4 03

Pos=(float)buffer_get_int32(data, &ind) / 1000.0

Motor single or multiple parameter acquisition command example

Serial command: 02 05 32 00 00 00 01 58 4C 03 // Get motor temperature command

Instruction Explanation: This command allows the retrieval of single or multiple motor parameters. The parameters to be retrieved are determined by the 4-byte data segment. Corresponding to a bit being set to 1, the motor will return the corresponding motor parameter, and for a bit set to 0, that field will be excluded.

The motor parameters corresponding to each bit are as follows:

Bit 32-19	Bit 18	Bit 17	Bit 16	Bit 10-15	Bit 9	Bit 8	Bit 7
Reserved	Motor ID (1 byte)	Motor position (4 bytes)	Motor error flag (1 byte)	Reserved	Input voltage (2 bytes)	Motor speed (4 bytes)	Duty cycle (2 bytes)
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1		
Iq current (4 bytes)	Id current (4 bytes)	Input current (4 bytes)	Output current (4 bytes)	Motor temperature (2 bytes)	MOS temperature (2 bytes)		

After receiving this command, the motor will respond with the corresponding parameters.

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

Conversion formulas for parameters sent by the motor:

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;

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,// Overvoltage
	FAULT_CODE_UNDER_VOLTAGE,// Undervoltage
	FAULT_CODE_DRV,// Driver fault
	FAULT_CODE_ABS_OVER_CURRENT,// Motor overcurrent
	FAULT_CODE_OVER_TEMP_FET,// MOS overtemperature
	FAULT_CODE_OVER_TEMP_MOTOR,// Motor overtemperature
	FAULT_CODE_GATE_DRIVER_OVER_VOLTAGE,// Driver overvoltage
	FAULT_CODE_GATE_DRIVER_UNDER_VOLTAGE,//Driver undervoltage
	FAULT_CODE_MCU_UNDER_VOLTAGE,// MCU undervoltage
	FAULT_CODE_BOOTING_FROM_WATCHDOG_RESET,// Undervoltage
	FAULT_CODE_ENCODER_SPI,// SPI encoder fault
	FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE,// Encoder below minimum
amplitude

FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE,//Encoder above maximum
amplitude
	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,// Unbalanced currents
} mc_fault_code;
```

II. Control command examples:

Example of duty cycle mode transmission

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) //Value as the received 4-byte data/10000.0

Example of current loop transmission

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

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

Current=(float)buffer_get_int32(data, &ind) / 1000.0 //Value as the received 4-byte data/1000.0

Example of brake current mode transmission

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

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

I_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 //Value as the received 4-byte data/1000.0

Example of speed loop transmission

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) //Value as the received 4-byte data

Example of position loop transmission

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

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

Pos=(float)buffer_get_int32(data, &ind) / 1000000.0 //Value as the received 4-byte data/1000000.0

Example of handbrake current mode transmission

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

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

HAND_Brake=(float)buffer_get_int32(data, &ind) / 1000.0 // Value as the received 4-byte data/1000.0

Example of position-speed loop mode transmission

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

/*

180 degrees, speed 5000 ERPM, acceleration 30000/S

Data segment: Position + Speed + Acceleration

*/

Pos=(float)buffer_get_int32(data, &ind) / 1000.0) // Position value as the received 4-byte data/1000.0

Speed=(float)buffer_get_int32(data, &ind) // Value as the received 4-byte data

Acc_Speed=(float)buffer_get_int32(data, &ind)// Value as the received 4-byte data

Example of multi mode transmission

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

Example of single mode transmission

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

Example of setting the current position as zero position transmission

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

Shortest distance return-to-zero command

Serial command: 02 05 65 00 00 00 00 3A 8B 03 // Make the motor return to the relative zero position in the shortest distance

Serial checksum:

```
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 };


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

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

//Arrange int32 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;
}

//Arrange uint32 data
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;
}

//Arrange int64 data
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;
}

//Arrange uint64 data
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 Checksum
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;
}

 //Organize and send data packet
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 Mode Communication Protocol

Special CAN Codes

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 Current Motor Position as zero position: {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0XFE }

Note: It is necessary to enter Motor Control Mode before controlling the motor using CAN communication!

PS: (If you want to read the current state in a stateless manner, the command to send is {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0XFC }）

MIT Mode Driver Board Receive Data Definition

Identifier: Set Motor ID (default is 1)

Frame Type: Standard Frame

Frame Format: DATA

Data Length Code (DLC): 8 Bytes

Data Field	DATA[0]	DATA[1]	DATA[2]	DATA[3]	
Data Bits	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 Field	DATA[4]	DATA[5]	DATA[6]		DATA[7]
Data Bits	7-0	7-0	7-4	3-0	0-7
Data Content	KP Value Low 8 bits	KD Value High 8 bits	KD Value Low 4 bits	Current Value High 4 bits	Current Value Low 8 bits

MIT Mode Driver Board Send Data Definition

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

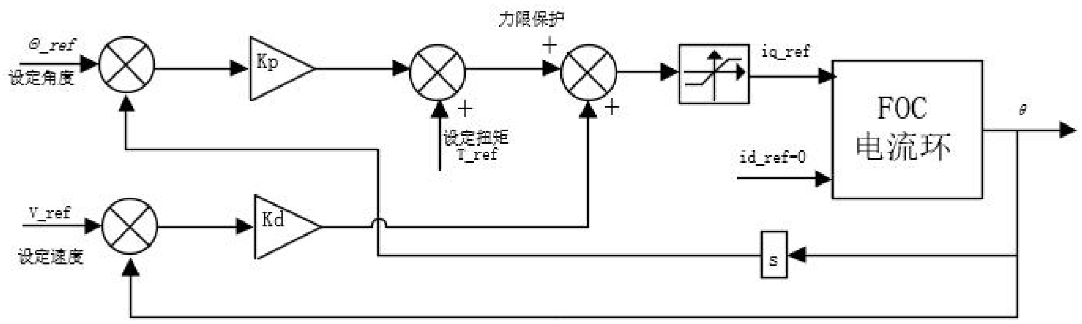
Frame Format: DATA Data Length Code (DLC): 8 Bytes

Data Field	DATA[0]	DATA[1]	DATA[2]	DATA[3]	DATA[4]
Data Bits	7-0	7-0	7-0	7-0	7-4
Data Content	Driver ID Number	Motor Position High 8 bits	Motor Position Low 8 bits	Motor Speed High 8 bits	Motor Speed Low 4 bits

Data Field	DATA[4]	DATA[5]	DATA[6]	DATA[7]
Data Bits	3-0	7-0	7-0	7-0
Data Content	Current Value High 4 bits	Current Value Low 4 bits	Motor Temperature	Motor Error Flag

CAN Speed: 1 MHz

MIT mode simplified control block diagram



Parameter Ranges:

Module	AK10-9	AK60-6	AK70-10	AK80-6	AK80-9	AK80-80	AK80-8
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	-37.5f-37.5f
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
Kp Range	0-500						
Kd Range	0-5						

MIT Mode Sending&Receiving Code Example

Sending Example 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 =-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;

    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 High 8
    msg->data[1] = p_int&0xFF;      // Position Low 8
    msg->data[2] = v_int>>4;            // Speed High 8 bits
    msg->data[3] = ((v_int&0xF)<<4)|(kp_int>>8);    // Speed Low 4 bits KP High 4 bits
    msg->data[4] = kp_int&0xFF;     // KP Low 8 bits
    msg->data[5] = kd_int>>4;       // Kd High 8 bits
    msg->data[6] = ((kd_int&0xF)<<4)|(t_int>>8);        // KP Low 4 bits Torque High 4 bits
    msg->data[7] = t_int&0xff;      // Torque Low 8 bits
}
```

When sending packets, all numbers need to go through the following function to be converted into integer values before being 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)));
}
```

Receiving Example Code

```
void unpack_reply(CANMessage msg){
    /// unpack ints from can buffer ///
    int id = msg.data[0]; //Driver ID
    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
    Int T_int = msg.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, -I_MAX, I_MAX, 12);
    float T =T_int;
    if(id == 1){
      postion = p;                  // Read corresponding data based on ID
      speed = v;
      torque = i;
      Temperature = T-40;     // Temperature range: -40~215
    }
  }
```

When receiving, convert all values to floating-point numbers using 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;
}
```