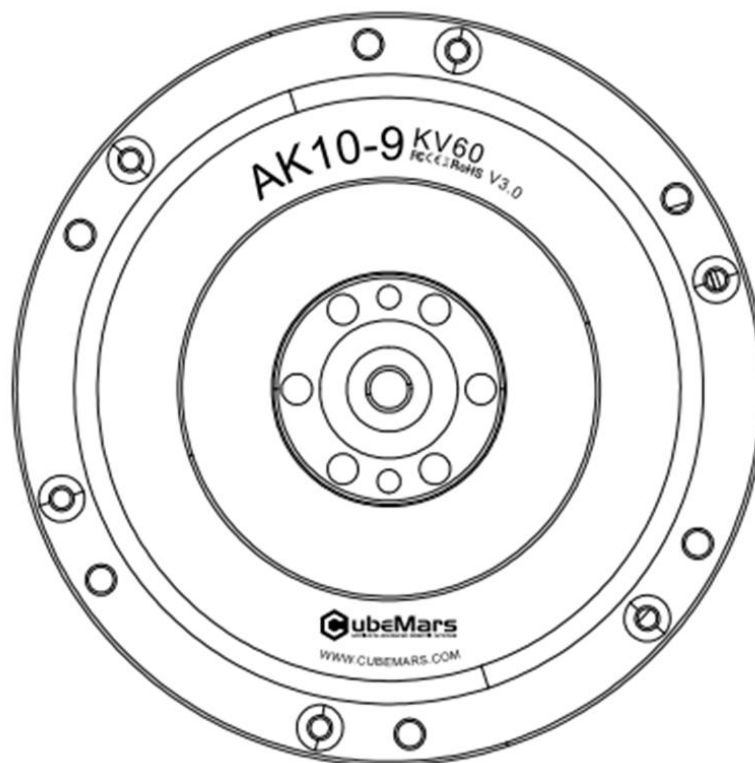


V3 Drive Software User Manual

V1.0.1



Content

V3 Drive Software User Manual	1
Precautions	3
Product Features	3
Disclaimer	4
1. User Manual for the Upper Computer	4
1.1 Upper Computer Interface and Documentation	4
1.1.1 Configuration	6
1.1.2 Real-time Status	10
1.1.3 Real-time Data	11
1.1.4 Language Switching Between Chinese and English	12
1.1.5 Control	12
1.1.6 Connection	14
1.1.7 Stop	15
1.2 Driver Board Calibration	15
1.2.1 Calibration Procedure	15
1.3 Control Demonstration	16
1.3.1 Servo Mode	16
1.3.2 Force Control Mode (MIT)	20
1.4 Firmware Update	22
2. Driver Board Communication Protocol and Description	23
2.1 Servo Mode Control Modes and Description	23
2.1.1 Duty Cycle Mode	24
2.1.2 Current Loop Mode	25
2.1.3 Current Brake Mode	25
2.1.5 Position Ring Mode	26
2.1.6 Set Origin Mode	27
2.1.7 Position-Velocity Loop Mode	28
2.1.8 Motor Failure Mode	29
2.1.9 Feedback Message Configuration	29
2.2 Force Control Mode(MIT) Communication Protocol	30

2.3 Motor Message Format	34
2.3.1 CAN upload message protocol	34
2.3.2 Serial Port Protocol	36
2.4 Control Command Examples	44
2.4.1 CAN Control Command Examples	44
2.4.2 Serial Port Control Command Examples	46

Precautions

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

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

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

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

Product Features

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

Disclaimer

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

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

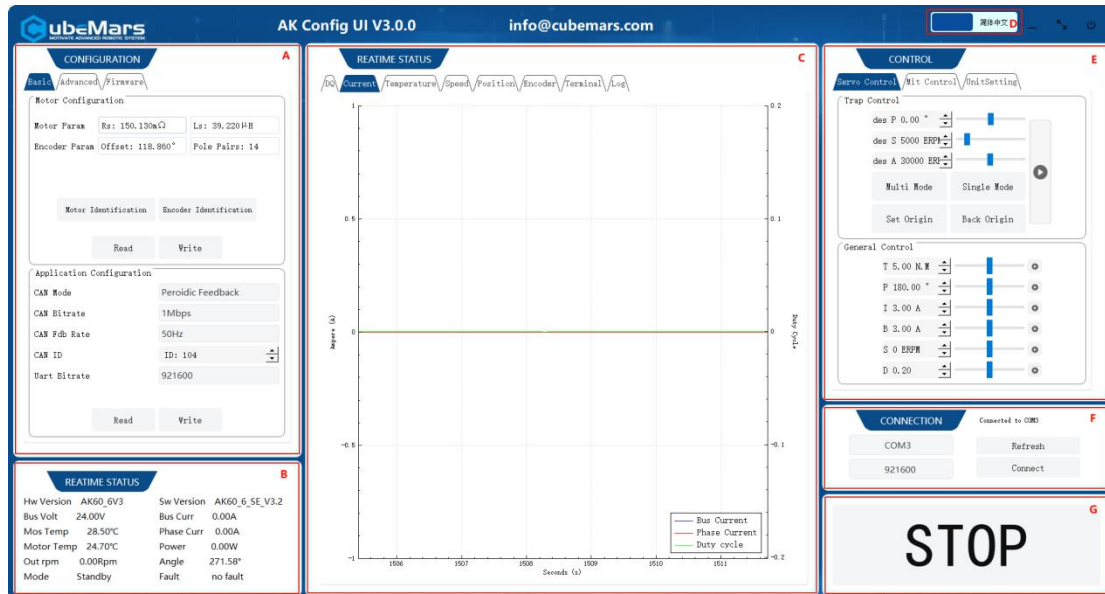
CubeMars is a trademark of Nanchang Kude Intelligent Technology Co., Ltd. and its affiliated companies. The product names and brands mentioned in this document are trademarks of the companies. This product and manual are copyrighted by Nanchang Kude Intelligent Technology Co., Ltd. No copying or reprinting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

Version Changed Record

Date	Version	Content
2026.4.29	V1.0.0	1.First Edit
2026.6.16	V1.0.1	1. New AK45-36 、 AK45-10and AK40-10 parameters

1. User Manual for the Upper Computer

1.1 Upper Computer Interface and Documentation



- A. configure
- B. real-time status
- C. real-time data
- D. Language Switching (Chinese/English)
- E. control
- F. connection
- G. Stop

1.1.1 Configuration

1.1.1.1 Basic Configuration

CONFIGURATION

Basic
Advanced
Firmware

Motor Configuration

Motor Param	Rs: 23.78mΩ	Ls: 23.78μH
Encoder Param	Offset:0.00°	Pole Pairs:21

Motor Identification

Encoder Identification

Read

Write

Application Configuration

CAN Mode	Periodic Feedback
CAN Bitrates	1Mbps
CAN Fdb Rate	100Hz
CAN ID	ID: 10 ▲ ▼
Uart Bitrates	921600
Timeout	1000ms ▲ ▼

Read

Write

The Basic Settings interface allows users to configure motor parameters and communication parameters; the Motor Settings section supports motor parameter identification, encoder parameter calibration, and parameter reading and writing; for detailed motor calibration procedures, refer to Section 1.2; the Application Settings section supports configuration of CAN port mode, CAN bus rate, CAN feedback rate, CAN ID, and serial port baud rate, enabling users to select appropriate settings based on actual operating conditions.

The CAN bus mode consists of two types: periodic feedback mode and request-response mode. In periodic feedback mode, message packets are transmitted at regular intervals based on the CAN feedback rate; in request-response mode, message packets are transmitted only when the motor receives the correct control command.

Motor Identification

Motor parameter identification can be performed to verify whether the motor is in motion via the position curve; once completed, the system will automatically stop and update the motor parameters to the upper computer.

⚠: The entire motor parameter identification process must be conducted under no-load conditions; otherwise, it may result in inaccurate parameter identification or motor damage. At the start of the identification process, a sharp, brief sound will be heard, after which the motor will begin to rotate accompanied by increased noise.

Encoder Identification

By performing encoder parameter identification, you can verify whether the motor is in motion via the position curve; once completed, the system will automatically stop and update the encoder parameters to the upper computer.

⚠: During the entire motor encoder parameter identification process, the motor must remain in no-load condition; otherwise, inaccurate parameter identification or motor damage may occur. The identification process will produce a relatively high-pitched noise and cause significant motor heating; repeated consecutive executions of this process may lead to a sudden rise in motor temperature.

Read

Read the drive board motor parameters and encoder parameters to the upper computer.

Write

Write the upper-level motor parameters and encoder parameters to the drive board.

1.1.1.2 Advanced Settings

⚠: Please note that customers using this product for the first time should not arbitrarily modify the relevant parameters in the advanced settings; otherwise, the motor may exhibit abnormal operation.

CONFIGURATION

Basic
Advanced
Firmware

Advanced Motor Configuration

Phase Current Limit	Min: -30.00A	▲▼	Max: 30.00A	▲▼
Bus Current Limit	Min: -30.00A	▲▼	Max: 30.00A	▲▼
Bus Voltage Limit	Min: 10.00V	▲▼	Max: 60.00V	▲▼
Motor temp Limit	Start: 50.00°C	▲▼	End: 60.00°C	▲▼
Mos temp Limit	Start: 50.00°C	▲▼	End: 60.00°C	▲▼
Motor Ident Param	Rs: 10.00mΩ	▲▼	Ls: 60.00μH	▲▼
Encoder Ident Param	Pole Pairs: 10	▲▼	Ofs: 0.00°	▲▼
Gear Param Edit	Ratio: 10	▲▼	Ofs: 0.00°	▲▼
Current Control	Kp: 0.0000	▲▼	Ki: 0.0000	▲▼
Speed Control	Kp: 0.0000	▲▼	Ki: 0.0000	▲▼
Position Control	Kp: 0.0000	▲▼	Ki: 0.0000	▲▼
			Kd: 0.0000	▲▼
Motor ERPM limit	Min: -60000 rpm	▲▼	Max: 60000	▲▼

Read
Write
Restore

Load from file
Save to file

The advanced settings interface provides users with a wider range of customizable parameters, allowing them to configure settings according to their needs; for enhanced usability, in addition to the parameter read/write capabilities, the interface also offers parameter restoration, loading, and exporting functions.

Read

Read Parameters: Click to read the driver board parameters to the upper computer.

Write

Write Parameters: Click this button to write the current upper computer parameters to the driver board.

⚠: Before modifying motor parameters, be sure to first read the existing parameters; otherwise, other motor parameters may become incorrect. If this occurs, please download the default APP parameters for the corresponding motor from the official website, and then use the "Import Settings" feature to write these default parameters into the motor!

Restore

Parameter Restoration: Clicking this will restore the upper computer's parameters

to the default values for the corresponding model.



Load from file

Loading Parameters: Load the parameter file onto the upper computer; after clicking, simply select the parameter file corresponding to the desired motor model. The two available parameter file formats are AppParams and McParams.



Save to file

Exporting parameters: Click "Export Parameters," select a file save path, and you will obtain a custom parameter file.

⚠: Please strictly adhere to the specified voltage, current, power, and temperature limits when using this product. Should any injury to the human body, or irreversible damage to the driver board and motor, result from non-compliant operation of this product, our company shall not assume any legal liability.

1.1.1.3 Firmware Update

CONFIGURATION

Basic
Advanced
Firmware

IAP Option

Choose Fw AK10_9_V3_260515_V3

Jump to IAP
Jump to APP

Upload
Cancel

Upload Progress

☐

0%

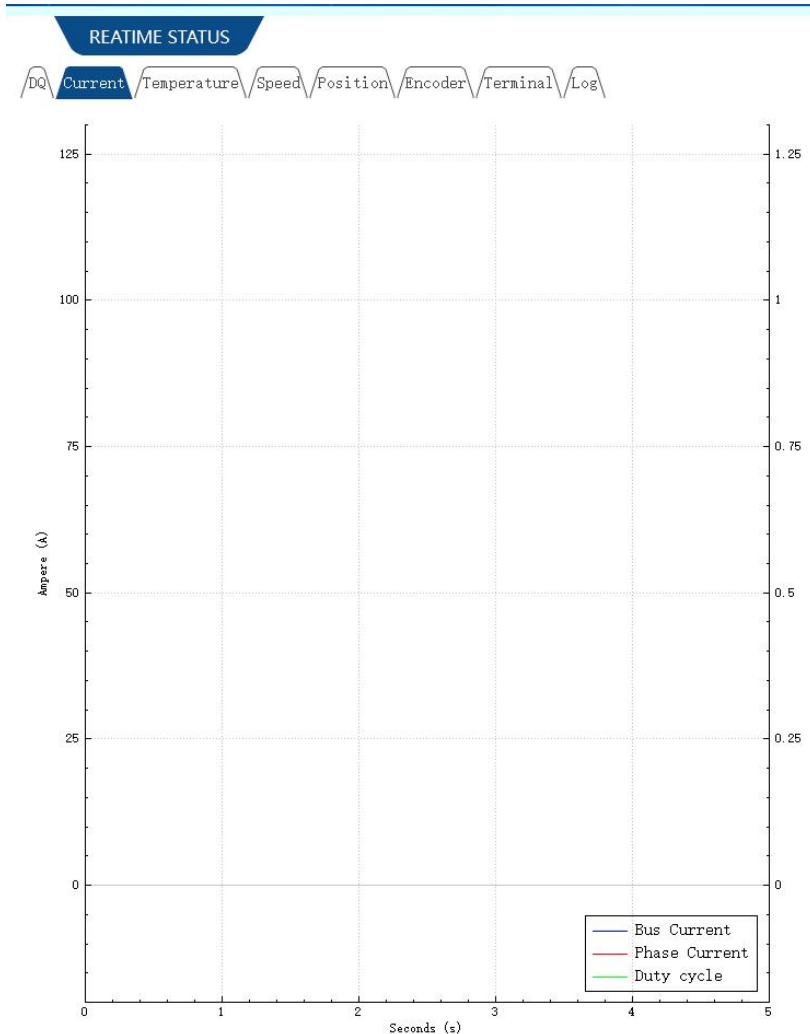
The firmware upgrade interface consists of three sections: firmware information, upgrade options, and upgrade progress. Customers can perform firmware upgrades according to their specific requirements; for detailed firmware upgrade steps, refer to Section 1.4.

1.1.2 Real-time Status

REALTIME STATUS			
Hw Version	Unknown	Sw Version	Unknown
Bus Volt	0.00V	Bus Curr	0.00A
Mos Temp	0.00°C	Phase Curr	0.00A
Mtr temp	0.00°C	Power	0.00W
Mtr rpm	0.00Rpm	Angle	0.00°
Mode	Standby	Fault	no fault

The real-time status interface displays the current software and hardware versions, the motor firmware version, motor parameter status, operating mode, and fault status.

1.1.3 Real-time Data



This page supports viewing real-time data feedback, a debugging interface, and image rendering. The available data include:

DQ: DQ voltage represents the motor voltage converted to the D-axis and Q-axis components, while DQ current represents the motor current converted to the D-axis and Q-axis components.

Current: "Current_in" denotes the input current of the motor, "Current_motor" denotes the phase current of the motor, and "Duty_cycle" denotes the duty cycle of the motor.

Temperature: "Temperature MOSFET" refers to the temperature of the motor MOSFET, while "Temperature Motor" refers to the temperature of the motor.

Speed: ERPM denotes the electrical rotational speed of the motor rotor.

Position: Position indicates the position of the motor rotor.

Encoder angle: The encoder angle refers to the angular position of the encoder.

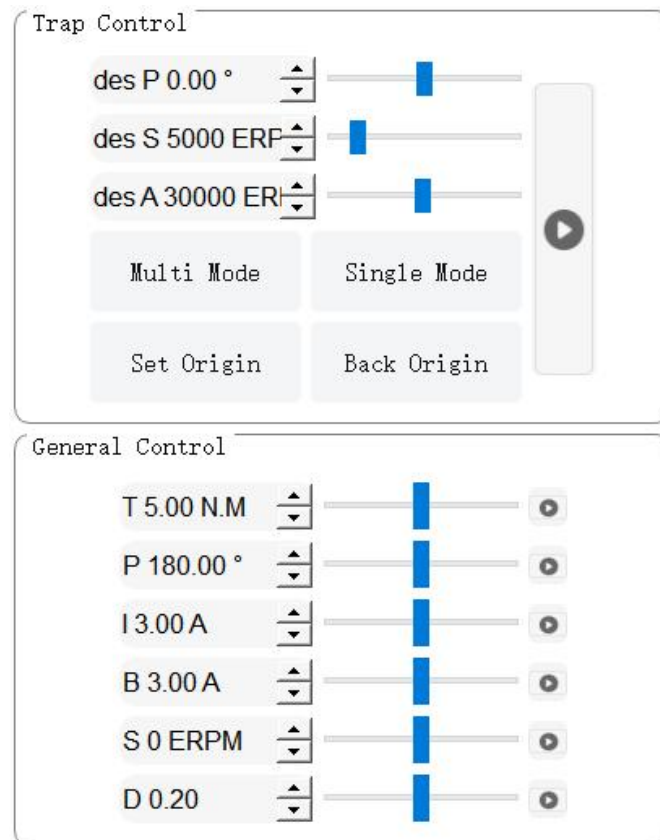
1.1.4 Language Switching Between Chinese and English



The upper computer currently supports language switching between Chinese and English; simply click "CN" or "EN" within the red box in the upper-right corner of the interface to switch languages.

1.1.5 Control

1.1.5.1 Servo Control



Multi Mode

In the position-speed loop mode, the motor can be set to a position range of -36,000° to 36,000°.

Single Mode

In the position-speed loop mode, the motor can be set to a position range of 0–359°.

Set Origin

The motor will set the current rotor position to the zero position;

Back Origin

: Rotate the motor to the zero position; (The motor will operate at its maximum speed; exercise caution)



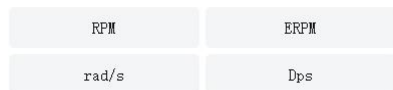
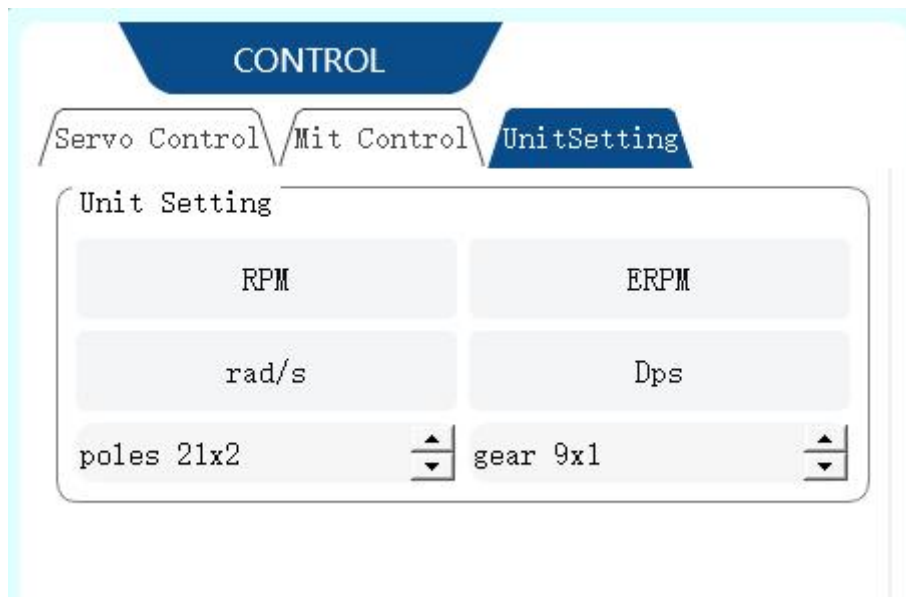
: Corresponding mode activation button.

1.1.5.2 Force Control(MIT)



The Force Control Mode (Motion Control Mode, MIT MODE) provides three distinct control modes—position, speed, and torque loops—determined by the transmitted data; for detailed operational procedures, refer to section 1.3.2.

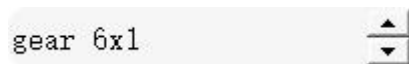
1.1.5.3 Unit Configuration



Each button corresponds to the unit it needs to convert to, supporting conversion and display for rotational speed (RPM, ERPM, rad/s, DPS);

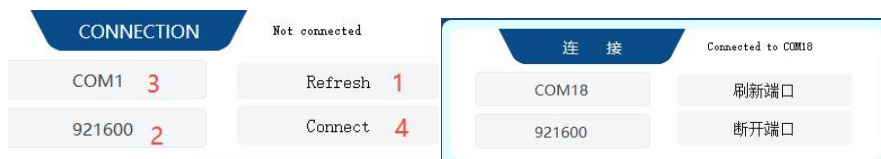


Enter the number of motor poles; as shown in the figure on the left, 21 represents the number of pole pairs.



: Enter the motor reduction ratio.

1.1.6 Connection



The connection module allows configuration of the serial port communication. As shown in the left figure, first click "Refresh Port," then select the communication baud rate, choose the correct port, and finally click "Connect Port" for the motor. When the displayed text changes from "Not connected" to "Connected to COMX," it indicates that the serial port connection is successful. The successful connection state is illustrated in the right figure; clicking "Disconnect Port" will terminate the communication.

1.1.7 Stop

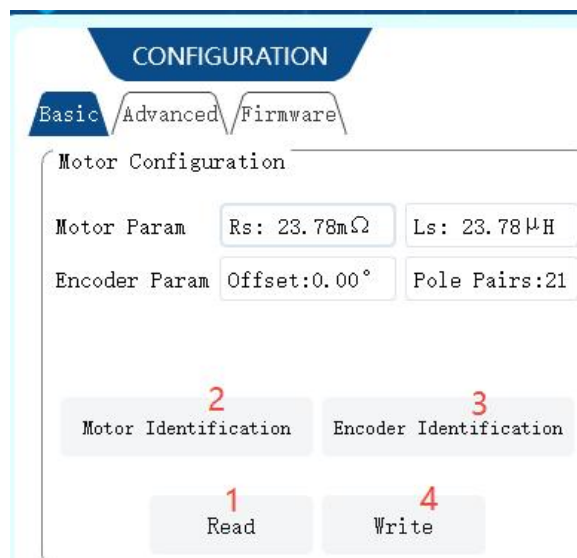


The experiment stop button: when clicked, the motor will cease operation.

1.2 Driver Board Calibration

If you have reinstalled the driver board on the motor, swapped the phase sequence of the motor's three-phase wires, or updated the firmware, you will need to perform a new calibration (calibration is already performed during initial factory setup, so no additional calibration is required). Once calibrated, the motor can be used normally.

1.2.1 Calibration Procedure



STEP0:

Verify that the motor's input power supply is stable and that the connectors are properly connected; once a successful connection with the upper computer is established, access the system configuration page.

STEP1:

Click "Read" until the connection  interface appears;

STEP2:

Click "Motor Identifications"; following a brief  buzz, the interface will appear. The motor will then begin to rotate ; after approximately 10

seconds, the motor will stop rotating. Once the interface reappears, the motor parameter recognition process has been completed.

STEP3:

Click "Encoder Identifications"; the motor will rotate at low speed , and after approximately 45 seconds, the connection interface will indicate that the encoder parameter recognition has been completed.

STEP4:

Click "Write"; once the  connection interface appears, the calibration is complete.

⚠: The entire motor parameter identification process must be conducted under no-load conditions; otherwise, it may result in inaccurate parameter identification or motor damage. During the motor parameter identification process, the motor generates significant heat, and repeated consecutive executions of this process can cause the motor temperature to rise sharply.

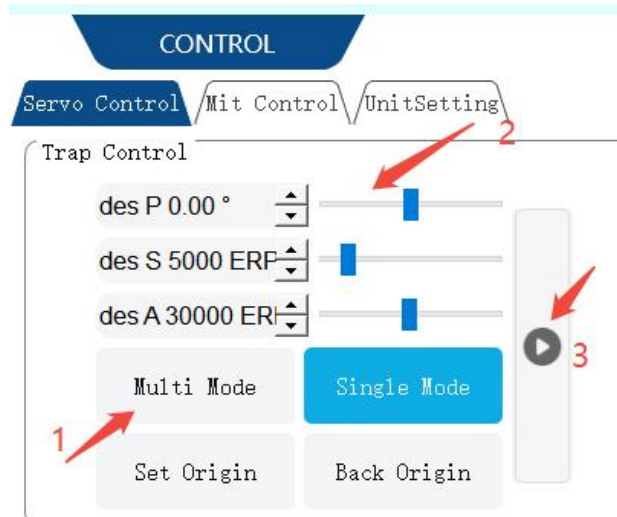
1.3 Control Demonstration

Once you have confirmed that the motor power supply and load are correctly installed, that the wiring is correct, and that the drive board has been calibrated, you may begin using the motor. The following sections provide detailed instructions on how to operate the servo control and force control modes, respectively.

1.3.1 Servo Mode

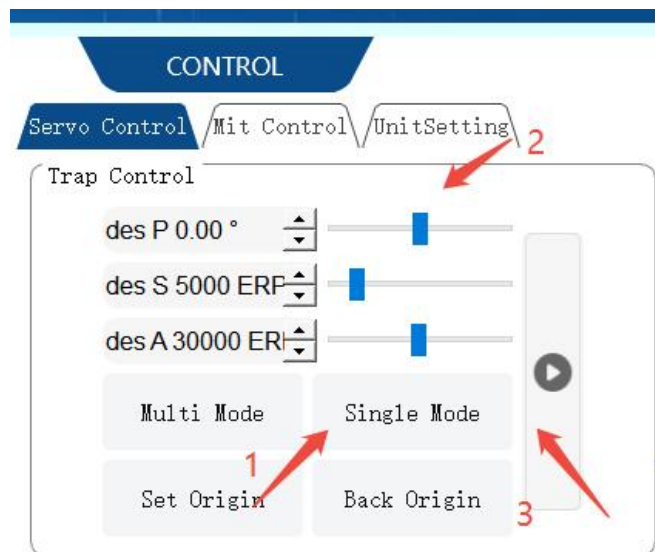
1.3.1.1 Multi-loop Position-Speed Loop Mode

Once it has been confirmed that the motor's input power supply is stable and the **connector** is properly connected, and after successfully establishing a connection with the upper computer, click "Multi-Round Mode" in the "Servo Control" interface, then enter the target position (set to ± 100 revolutions, i.e., $-36,000^\circ$ to $36,000^\circ$), the target speed, and the target acceleration; the motor will then move according to the specified target speed until it reaches the target position.



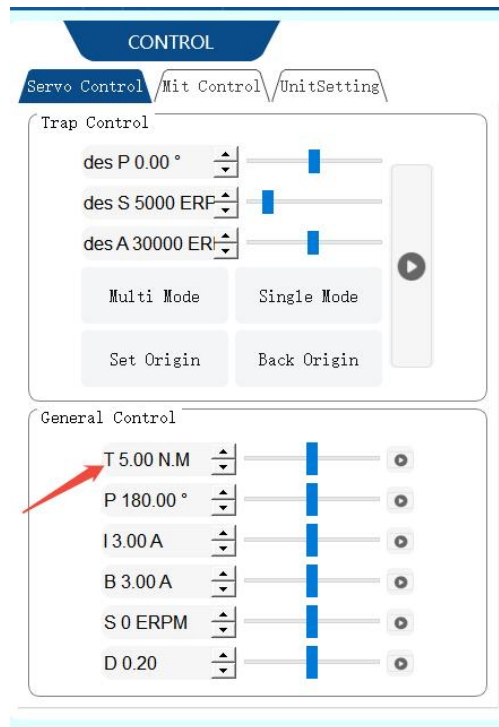
1.3.1.2 Single-loop position velocity loop mode

Once you have confirmed that the motor's input power supply is stable, the connector is properly connected, and you have successfully established a connection with the upper computer, click "Single-Cycle Mode" in the "Servo Control" interface, then enter the target position (at this stage, the position range is limited to one full cycle, i.e., 0°–359°), the target speed, and the target acceleration; the motor will then move at the specified speed until it reaches the target position.



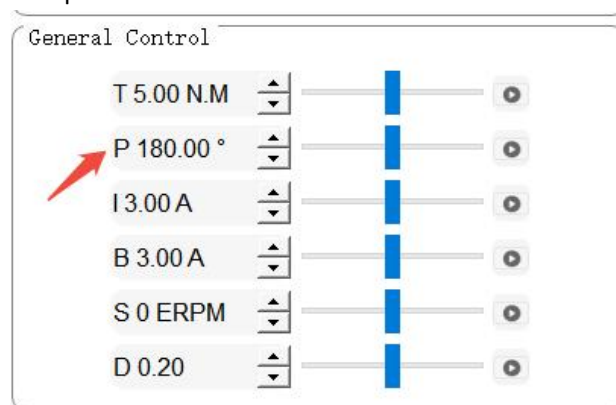
1.3.1.3 Torque Mode (Holding brake)

Once it has been confirmed that the motor's input power supply is stable, the connectors are properly connected, and a successful connection with the upper computer has been established, enter the desired torque T in the "Servo Control" interface; the motor will then apply braking under the specified torque.



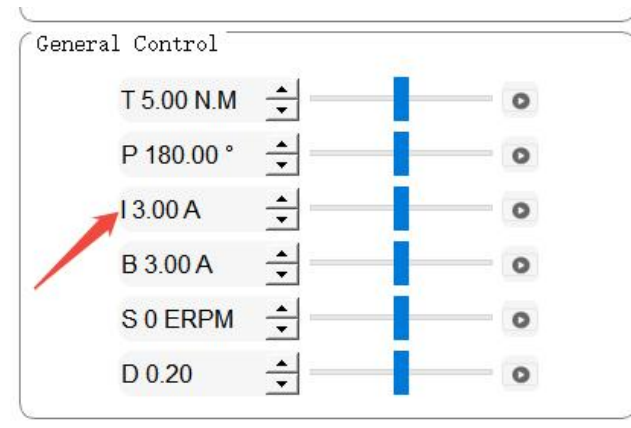
1.3.1.4 Position Loop Mode

Once it has been confirmed that the motor's input power supply is stable and the connector is properly connected, and after successfully establishing a connection with the upper computer, enter the desired position P in the "Servo Control" interface; the motor will then reach the desired position at its maximum speed and acceleration.



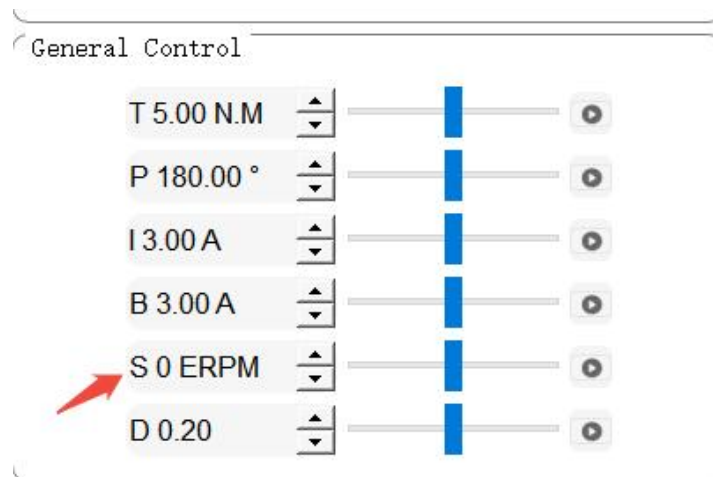
1.3.1.5 Current loop mode

Once it has been confirmed that the motor's input power supply is stable, the connectors are properly connected, and a successful connection with the upper computer has been established, enter the desired current I in the "Servo Control" interface; the motor will then operate at this desired current.



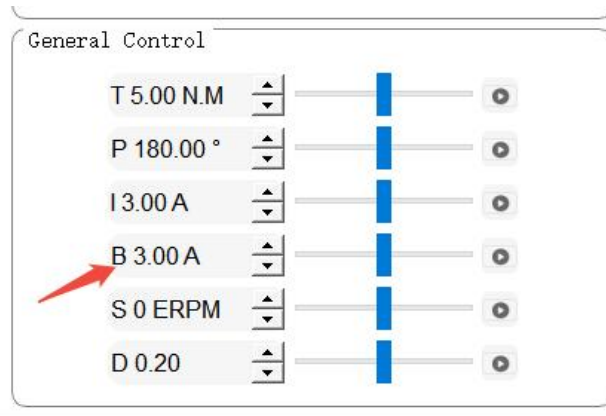
1.3.1.6 Speed Loop Mode

Once it has been confirmed that the motor's input power supply is stable, the connectors are properly connected, and the motor is operating in servo mode, and after a successful connection with the upper computer has been established, enter the desired rotational speed S ($\pm 50,000$ ERPM) in the "Servo Control" interface; the motor will then operate at the specified rotational speed (with the acceleration set to its default maximum value).



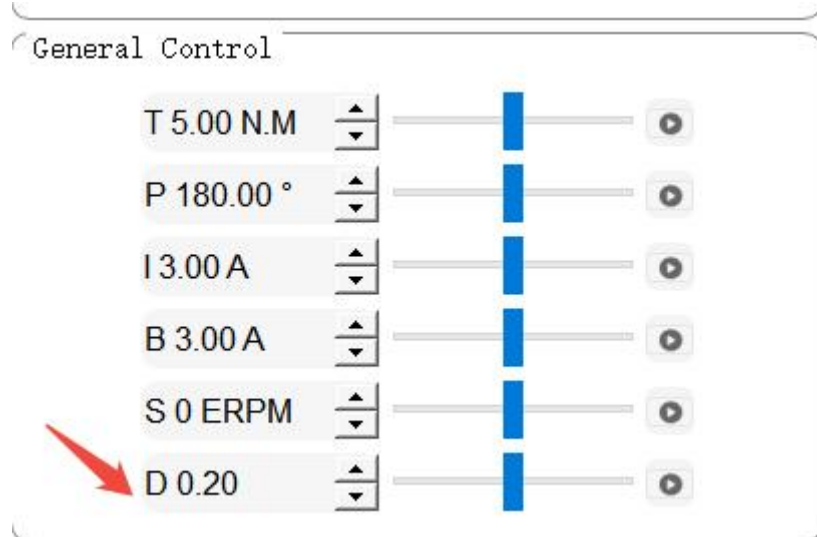
1.3.1.7 Brake Loop Mode

Once it has been confirmed that the motor's input power supply is stable, the connectors are properly connected, and a successful connection with the upper computer has been established, enter the desired braking current B in the "Servo Control" interface; the motor will then brake using the specified current.



1.3.1.8 Duty Cycle Mode

Once it has been confirmed that the motor's input power supply is stable, the connectors are properly connected, and the motor is operating in servo mode, and after a successful connection with the upper computer has been established, enter the desired duty cycle (default: 0.005–0.95) in the "Servo Control" interface; the motor will then operate according to the specified duty cycle.

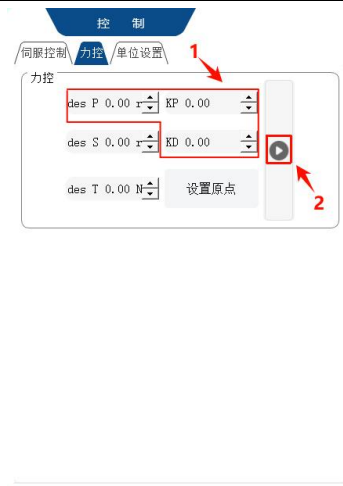


1.3.2 Force Control Mode (MIT)

1.3.2.1 Force-Control Position Mode

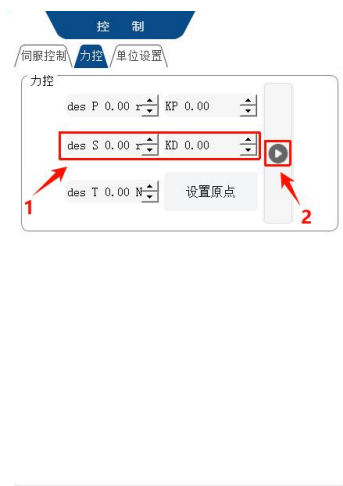
Ensure that the motor's input power supply is stable and that the connectors are properly connected. After successfully establishing a connection with the upper computer, input the desired position along with KP and KD, then click "Start"; the motor will then execute the position movement.

(When operated with only the specified target position and the Kp and Kd values set for the motor, the motor will rotate toward the specified position.)



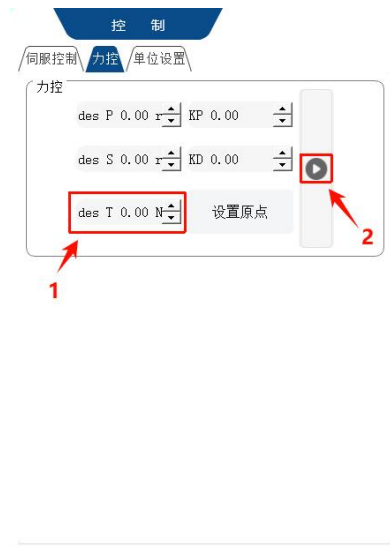
1.3.2.2 Force-Controlled Speed Mode

Once you have confirmed that the motor's input power supply is stable, the connectors are properly connected, and you have entered the desired input speed S and the KD value, click "Start"; the motor will then begin rotating at the specified speed. (When only the motor operating speed and the KD value are specified, the motor will rotate at the designated speed.)

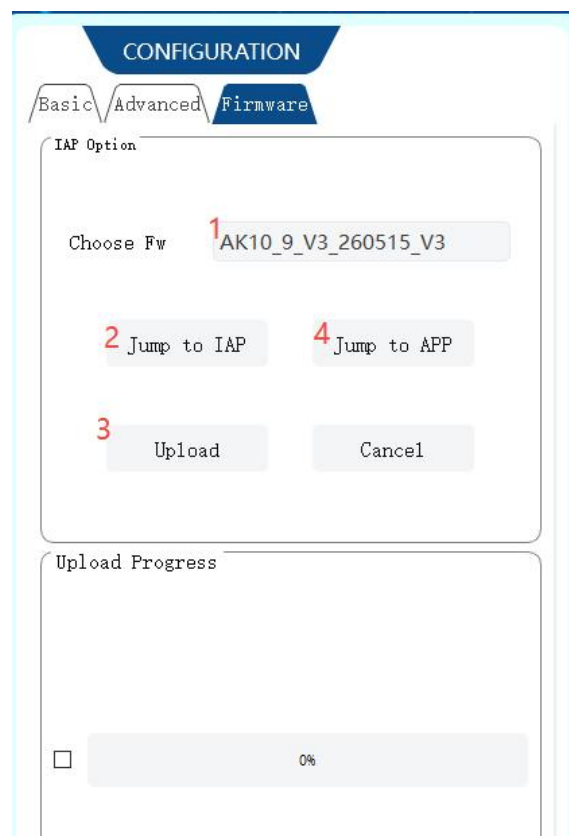


1.3.2.3 Torque Control Mode

Once you have confirmed that the motor's input power supply is stable, the connectors are properly connected, and the desired input torque T has been set, click "Start"; the motor will then execute a torque motion. (When operating with only a specified torque value, the motor will rotate at that designated torque value.)



1.4 Firmware Update



1. Select the corresponding firmware from the drop-down menu;
2. Click "Jump to IAP";
3. Click "Ipload" and wait until the upgrade progress bar reaches 100%;
4. Click "Jump to App," wait for 5 seconds, and the motor will enter operating mode, indicating that the firmware update has completed.

2. Driver Board Communication Protocol and Description

2.1 Servo Mode Control Modes and Description

The servo mode offers six control modes, each associated with a specific ID number.

Duty cycle mode: A specified voltage is applied to the motor based on a predefined duty cycle, similar to square-wave drive operation;

Current loop mode: Given the specified I_q current of the motor, since the motor's output torque is given by $i_q \times K_T$, this mode can be used as a torque loop (the K_T value can be referenced from the parameter range specified in the force control mode communication protocol);

Current braking mode: Set the specified braking current for the motor to lock it in place at its current position (take note of the motor temperature during operation);

Speed mode: specifies the operating speed of the motor (with acceleration set to its maximum value by default);

Position mode: Given a specified target position for the motor, the motor will move to that specified position (with speed and acceleration set to their maximum default values);

Position-velocity loop mode: specifies the desired position, velocity, and acceleration of the motor.

Data definition for the servo mode drive board

Identifier: Control Mode ID + Drive ID; Frame Type: Extended Frame

Frame format: DATA; DLC: 8 bytes

The servo motor employs the CAN protocol, utilizing an extended frame format as shown below.

Can ID bits	[28]-[8]	[7]-[0]
Field name (Function definition)	Control Mode ID	Drive ID

The control mode has 9 characteristic values {0,1,2,3,4,5,6,15,16}, each corresponding to one of 9 distinct functions.

Duty cycle mode: 0

Current loop mode: 1

Current braking mode: 2

Speed mode: 3

Position Mode: 4

Set origin mode; 5

Position-velocity loop mode: 6

Motor disabled mode: 15

Feedback message setting: 16

The following provides various examples of motor control using different modes.

Below are various example calls to 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 mode
    CAN_DISABLE_=15,                 //Motor disabled mode
    CAN_Frame_=16                     //Feedback message setting
} 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 send TxMessage data
}

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

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

2.1.1 Duty Cycle Mode

Data Definition for Duty Cycle Mode

data bit	Data[0]	Data[1]	Data[2]	Data[3]
scope	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Cycle ratio: 25–32	Cycle ratio: 17–24	Cycle ratio: 9–16 bits	Duty cycle: bits 1–8

```

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);
}

```

2.1.2 Current Loop Mode

Data definition for current loop mode

data bit	Data[0]	Data[1]	Data[2]	Data[3]
scope	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Current: 25–32 bits	Current: 17–24 bits	Current: 9–16 bits	Current: 1–8 bits

The current value is of the int32 type, where the range -60000 to 60000 represents -60 A to 60 A.

Current loop mode transmission 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);
}

```

2.1.3 Current Brake Mode

Data Definition for Current Brake Mode

data bit	Data[0]	Data[1]	Data[2]	Data[3]
scope	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Brake current: 25–32 bits	Brake current: bits 17–24	Brake current: 9–16 bits	Brake current (bits 1–8)

The brake current value is of the int32 type, where values ranging from 0 to 60,000 correspond to

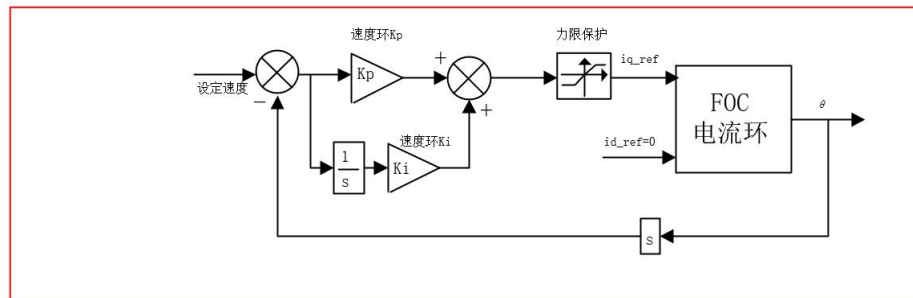
currents from 0 to 60 A.

Current Brake Mode Transmission 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);
}
```

2.1.4 Speed Loop Mode

Simplified speed loop block diagram



Data definition for speed loop mode

data bit	Data[0]	Data[1]	Data[2]	Data[3]
scope	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Speed: 25–32 bits	Speed: 17–24 bits	Speed: 9–16 bits	Speed: 1–8 digits

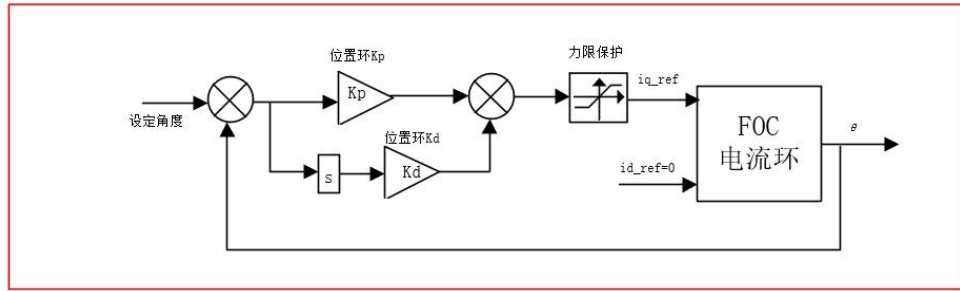
The speed value is of the int32 type; the range of -100,000 to 100,000 represents -100,000–100,000 ERPM (electrical rotational speed).

Speed loop transmission 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);
}
```

2.1.5 Position Ring Mode

Simplified Position Feedback Control Block Diagram



Data Definition for Position Ring Mode

Data bit	Data[0]	Data[1]	Data[2]	Data[3]
scope	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Positions 25–32	Positions 17–24	Positions 9–16	Positions 1–8

Here, the position is of the int32 type, and the range of -360,000,000 to 360,000,000 represents a position from -36,000° to 36,000°.

Position Ring Transmission Routine

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

2.1.6 Set Origin Mode

Data bit	Data[0]
Scope	0~0x01
Corresponding variable	Configuration command

The setting instruction is of type `uint8\_t`: a value of 0 sets the temporary origin (reset upon power loss), while a value of 1 sets the permanent zero point (automatic parameter saving).

Set the origin send routine

```
void comm_can_set_origin(uint8_t controller_id, uint8_t set_origin_mode) {
    int32_t send_index = 1;
```

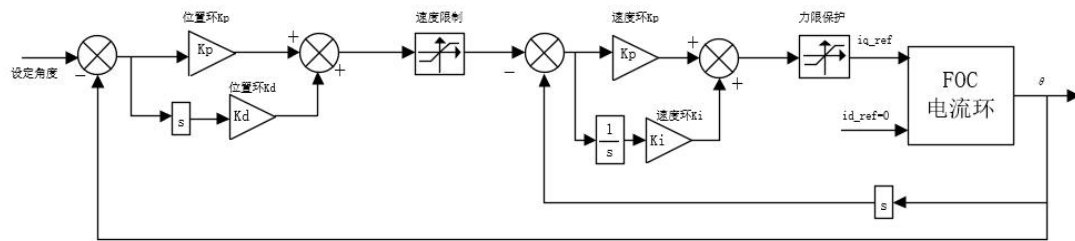
```

uint8_t buffer;
buffer=set_origin_mode;
comm_can_transmit_eid(controller_id |
    ((uint32_t) CAN_PACKET_SET_ORIGIN_HERE << 8), buffer, send_index);
}

```

2.1.7 Position-Velocity Loop Mode

Simplified block diagram of the position-velocity loop



Data Definition for Position-Velocity Loop Mode

data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
scope	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Positions 25–32	Position s 17–24	Position s 9–16	Position s 1–8	Speed increase by eight orders of magnitude	Speed is eight bits lower	Acceleration increase by eight bits	Low 8-bit acceleration

The position field is of type int32, with a range of -360,000,000 to 360,000,000, corresponding to -360,000° to 360,000°.

The speed is of the int16 type, with a range of -32768 to 32767 corresponding to -327680 to 327680 ERPM (electrical rotational speed);

The acceleration is of the int16 type, with a range of 0–32767; each unit corresponds to 10 ERPM (electrical rotational 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 = 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);
}

```

2.1.8 Motor Failure Mode

Data bit	(None)
Scope	(None)
Corresponding variable	(None)

No data content needs to be transmitted; simply sending the corresponding function frame will disable the motor. The motor will then set the DATA[7] bit in frame ID 0x29 to 0x77 and transmit this frame back once, indicating that the disabling operation was successful.

```

void comm_can_disable(uint8_t controller_id) {
    Int16_t send_index1 = 0;
    uint8_t buffer[8];
    comm_can_transmit_eid(controller_id |
        ((uint32_t)CAN_DISABLE_ << 8), buffer, send_index1);
}

```

2.1.9 Feedback Message Configuration

Data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Scope	0	0	0	0	0	0	0~0xff	0~0xff
Corresponding variable	Retain	Retain	Retain	Retain	Retain	Retain	High 8-bit parameter	Low 8-bit parameter

⚠: Note: This command is written to Flash; please avoid sending this command frequently.

The corresponding parameter bit is set to 1 to enable, and to 0 to disable (parameters are preserved during power loss); the mapping is as follows:

Specifically, the [15]th bit of the parameter indicates the addition of a 0x2A feedback frame (at byte position 4);

Among these, the [14]th parameter indicates the single-loop mode, with a positional range of 0–360°;

The remaining positions are retained.

For example, a parameter value of 0x8000 indicates the addition of a 4-byte position feedback.

```
void comm_can_set_frame(uint8_t controller_id, uint16_t frame) {
    int32_t send_index = 0;
    Int16_t send_index1 = 6;
    uint8_t buffer[8];
    buffer_append_uint16(buffer, frame, & send_index1);
    comm_can_transmit_eid(controller_id |
        ((uint32_t)CAN_Frame_ << 8), buffer, send_index1);
}
```

2.2 Force Control Mode(MIT) Communication Protocol

The force control mode comprises three control modes, all of which share a single control ID; the specific control mode is determined based on the transmitted data. For detailed operations of each mode via the upper computer, refer to section 1.3.2.

Position loop mode: The motor operates solely based on the specified target position and the Kp and Kd values; the motor will then rotate toward the specified position.

Speed loop mode: When only the motor operating speed and the Kd value are specified, the motor will rotate at the specified speed;

Current loop mode: The motor operates solely based on a specified torque value, rotating to achieve that torque.

The force control mode utilizes the CAN protocol and employs an extended frame format as shown below.

Can ID bits	[28]-[8]	[7]-[0]
Field name (Function definition)	Control Mode ID	Drive ID

The characteristic value of the control mode ID for the force-control mode is 8.

Data Definition for the Force Control Mode Driver Board

Identifier: Control Mode ID + Drive ID; Frame Type: Extended Frame

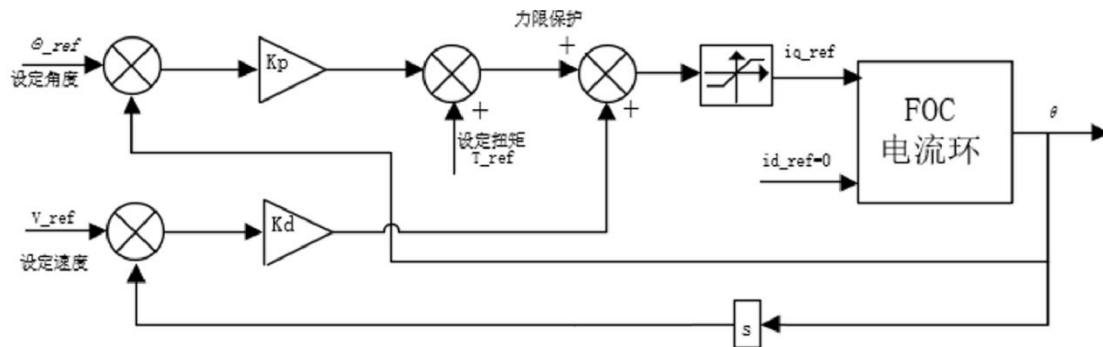
Frame format: DATA

DLC: 8 bytes

Data field	DATA[0]	DATA[1]		DATA[2]	DATA[3]
data bit	7-0	7-4	3-0	7-0	7-0
Data Content	The KP value is 8 bits high.	The KP value is 4 digits lower	The KD value is 4 digits higher	The KD value is low by 8 bits.	The position value has an 8-bit precision.

Data field	DATA[4]	DATA[5]	DATA[6]		DATA[7]
data bit	7-0	7-0	7-4	3-0	0-7
Data Content	The position value is 8 bits low.	The speed value has an 8-bit precision.	The speed value is 4 digits lower	The torque value is 4 digits higher.	The torque value is 8 bits lower.

Simplified Block Diagram of Force Control Mode



Parameter Range

Modular motor	KV	Motor position (rad)	torque coefficient	Motor speed (rad/s)	Motor torque (N·m)	Kpscope	Kdscope
AK10-9	60	-12.56f-12.56f	1.3137	-28.0f-28.0f	-54.0f-54.0f	0-500	0-5
AK40-10	170		0.4894	-45.0f-45.0f	-5.0f-5.0f		
AK45-10	75		1.1286	-20.0f-20.0f	-8.0f-8.0f		
AK45-36	80		3.9790	-6.0f-6.0f	-34.0f-34.0f		
AK60-6	80		0.5994	-60.0f-60.0f	-12.0f-12.0f		
AK60-39	80		3.4616	-10.0f-10.0f	-80.0f-80.0f		
AK70-9	60		1.0621	-30.0f-30.0f	-32.0f-32.0f		
AK80-8	60		1.0569	-38.0f-38.0f	-32.0f-32.0f		
AK80-9	100		0.7044	-65.0f-65.0f	-25.0f-25.0f		
AKE60-8	80		0.7382	-40.0f-40.0f	-15.0f-15.0f		
AKE80-8	30		1.7909	-20.0f-20.0f	-35.0f-35.0f		
AKE90-8	35		1.8265	-20.0f-20.0f	-200.0f-200.0f		
AKH70-16	41		2.8334	-13.0f-13.0f	-110.0f-110.0f		
AKH70-48	41		8.8123	-5.0f-5.0f	-280.0f-280.0f		

The relationship between torque and current is as follows:

$$T = K_t \times I_q$$

Where:

T denotes the output torque at the output end, expressed in N·m;

Kt is the torque coefficient, with units of N·m/A;

Iq is the torque current, with units in A.

Send code routine in force-control mode

Sending routine code (using AK10-9 as an example)

```
u8 SERVO_Can_Send_Msg(uint32_t ExtId,u8* msg,u8 len)
{
    u8 mbox;
    u16 i=0;
    //      TxMessage.StdId=stdId;          // Standard Identifier
    TxMessage.ExtId=ExtId;          // Set Extended Identifier
    TxMessage.IDE=CAN_Id_Extended;  // Standard Frame
    TxMessage.RTR=CAN_RTR_Data;     // Data Frame
    TxMessage.DLC=len;              // Data Length
    for(i=0;i<len;i++)
        TxMessage.Data[i]=msg[i];
    mbox= CAN_Transmit(CANx, &TxMessage);
    i=0;
    while((CAN_TransmitStatus(CAN1,      mbox)==CAN_TxStatus_Failed)&&(i<0XFFF))i++;
    //wating for end
    if(i>=0XFFF)return 1;
    return 0;
}
```

```
float fmaxf(float x, float y){
    /// Returns maximum of x, y ///
    return (((x)>(y))?(x):(y));
}
```

```
float fminf(float x, float y){
    /// Returns minimum of x, y ///
    return (((x)<(y))?(x):(y));
}
```

```
float fmaxf3(float x, float y, float z){
    /// Returns maximum of x, y, z ///
    return (x > y ? (x > z ? x : z) : (y > z ? y : z));
}
```

```
float fminf3(float x, float y, float z){
    /// Returns minimum of x, y, z ///
    return (x < y ? (x < z ? x : z) : (y < z ? y : z));
}
```

```
float P_MIN = -12.56f;  
float P_MAX = 12.56f;  
float V_MIN = -28.0f;  
float V_MAX = 28.0f;  
float T_MIN = -54.0f;  
float T_MAX = 54.0f;  
float Kp_MIN = 0;  
float Kp_MAX = 500.0f;  
float Kd_MIN = 0;  
float Kd_MAX = 5.0f;  
float Test_Pos = 0.0f;
```

```
int p_int ;  
int v_int ;  
int kp_int ;  
int kd_int ;  
int t_int ;
```

```
void pack_cmd(uint8_t controller_id, float p_des, float v_des, float kp, float kd, float t_ff){  
    uint8_t buffer[8];
```

```
    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 ///
```

```
    p_int = float_to_uint(p_des, P_MIN, P_MAX, 16);  
    v_int = float_to_uint(v_des, V_MIN, V_MAX, 12);  
    kp_int = float_to_uint(kp, Kp_MIN, Kp_MAX, 12);  
    kd_int = float_to_uint(kd, Kd_MIN, Kd_MAX, 12);  
    t_int = float_to_uint(t_ff, T_MIN, T_MAX, 12);
```

```
    /// pack ints into the can buffer ///
```

```
    buffer[0] = kp_int >> 4;          //KP High 8 bit  
    buffer[1] = ((kp_int & 0xF) << 4) | (kd_int >> 8); //KP Low 4 bit, Kd High 4 bit  
    buffer[2] = kd_int & 0xFF;       //Kd Low 8 bit  
    buffer[3] = p_int >> 8;          //Position High 8 bit  
    buffer[4] = p_int & 0xFF;        //Position Low 8 bit  
    buffer[5] = v_int >> 4;          //Speed High 8 bit  
    buffer[6] = ((v_int & 0xF) << 4) | (t_int >> 8);    //Lower 4 bits of speed, upper 4 bits of torque
```

```
buffer[7] = t_int&0xff; //Torque Low 8 bits
SERVO_Can_Send_Msg(controller_id |(((uint32_t) CAN_PACKET_SET_mit << 8), buffer, 8);
}
```

All values must be converted to integers using the following function before being transmitted 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)-1)/span))));
}
```

2.3 Motor Message Format

2.3.1 CAN upload message protocol

Definition of feedback data for the servo mode drive board

Identifier: Function ID + Driver ID;

Frame Type: Extended Frame

Frame format: DATA

DLC: 8 bytes

The servo motor employs the CAN protocol, utilizing an extended frame format as shown below.

Can ID bits	[28]-[8]	[7]-[0]
Field name (Function definition)	Function ID	Drive ID

functionID:

0x2C: The frame is the initial frame sent when the motor enters servo mode, and the response information is fixed to 0xFA, 0xFB, 0xFC, or 0xFD;

0x29: The frame represents the real-time feedback of the motor's current state in servo mode, along with its data bits;

The real-time motor status CAN messages utilize a timed upload mode, with an upload frequency configurable from 1 to 2000 Hz and an upload payload of 8 bytes.

Data bit	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
Scope	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff	0~0xff
Corresponding variable	Ranked eighth	Ranked eight places lower	Speed increased by eight orders of magnitude	Speed is eight bits lower	The IQ current is eight bits high.	The IQ current is low by eight bits.	Driver board temperature	Error code

The position is of the int16 type, and the range of -32000 to 32000 corresponds to positions from -3200° to 3200°.

Here, the speed is of the int16 type, with a range of -32000 to 32000 representing -320,000 to 320,000 ERPM (electrical rotational speed).

Output shaft speed = ERPM (electrical speed) ÷ number of pole pairs ÷ reduction ratio;

The Iq current is of the int16 type, where the value range of -6000 to 6000 corresponds to -60 A to 60 A.

Here, the temperature field is of the int8 type, with a range of -20 to 127, representing the driver board temperature from -20 °C to 127 °C;

The error codes are of type 'uint8', where 0 indicates no fault, 1 indicates motor overtemperature fault, 2 indicates overcurrent fault, 3 indicates overvoltage fault, 4 indicates undervoltage fault, 5 indicates encoder fault, 6 indicates MOSFET overtemperature fault, and 7 indicates motor stall.

The following is an example of message reception.

```
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 fault code  
}
```

0x2A (Optional): Use the feedback message to set a command for additionally transmitting one frame of position information; the data field Data[0-3] carries 4 bytes of motor position information, where the 4-byte position field is of int32 type, with a range from -2147483648 to 2147483647, representing a position range from -21474836.48° to 21474836.47°.

2.3.2 Serial Port Protocol

The message format for the serial port is as follows:

Frame header	data Length	data frame	data bit	check bit		Frame end
0xAA	Excluding the frame header, frame trailer, and parity bit	See data frame definition		8-bit high	Low 8 bits	0xBB

For the parity bit calculation, refer to Section 2.3.2.3.

Data frame definition:

```
typedef enum {
    COMM_FW_VERSION = 65,
    COMM_JUMP_TO_BOOTLOADER=66,
    COMM_ERASE_NEW_APP=67,
    COMM_WRITE_NEW_APP_DATA=68,
    COMM_GET_VALUES=69, // Retrieve motor operating parameters
    COMM_SET_DUTY=70, // The motor operates in duty cycle mode
    COMM_SET_CURRENT=71, // The motor operates in current-loop mode
    COMM_SET_CURRENT_BRAKE=72, // Motor current brake mode operation
    COMM_SET_RPM=73, // The motor operates in speed loop mode
    COMM_SET_RPM=73, // The motor operates in speed loop mode
    COMM_SET_POS=74, // The motor operates in position loop mode
    COMM_SET_POS=74, // The motor operates in position loop mode
    COMM_SET_HANDBRAKE=75, // The motor operates in hand-brake current loop mode
    COMM_SET_DETECT=76, // Motor real-time feedback of current position command
    COMM_ROTOR_POSITION=87, // Motor feedback of current position
    COMM_GET_VALUES_SETUP=16, // Command for retrieving single or multiple motor
parameters
    COMM_SET_POS_SPD=60, // The motor operates in position-speed loop mode
    COMM_SET_POS_MULTI=61, // Set motor motion to multi-turn mode, range: ±100 turns
    COMM_SET_POS_SINGLE=62, // Set the motor motion to single-cycle mode
    COMM_SET_POS_UNLIMITED=63, // Reserved
    COMM_SET_POS_ORIGIN=64, // Set the motor origin
} COMM_PACKET_ID;
```

2.3.2.1 Parameter Acquisition

1. Obtain motor parameters

order:

AA 01 45 18 61 BB

respond:

AA 49 45 01 27 FC DF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

F2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FD 00 00 00 07 00 FF EF 54 F1 68 00
 00 00 00 00 00 00 00 00 00 FF FF FF FC 86 D5 BB

explain:

Acquire motor parameters; upon receipt, the motor provides feedback on its current status.

Parameter Analysis:

AA (Frame Header) + 55 (Data Length) + 45 (Data Frame) + MOS Temperature (2 bytes) + Motor Temperature (2 bytes) + Output Current (4 bytes) + Input Current (4 bytes) + Id Current (4 bytes) + Iq Current (4 bytes) + Motor Throttle Value (2 bytes) + Motor Speed (4 bytes) + Input Voltage (2 bytes) + Reserved Bytes (24 bytes) + Motor Status Code (1 byte) + Motor Outer Ring Position Value (4 bytes) + Motor Control ID (1 byte) + Temperature Reserved Bytes (6 bytes) + Vd Voltage Value (4 bytes) + Vq Voltage Value (4 bytes) + Current Control Mode (4 bytes) + Encoder Angle (4 bytes) + Outer Encoder Angle (4 bytes) + CRC + BB (Frame Footer)

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

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

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

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

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

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

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

Duty cycle = (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 ring position = (float) buffer_get_float(data, &ind)

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)

Encoder angle = (float) buffer_get_float(data, &ind)

External encoder angle = (float) buffer_get_float(data, &ind)

2. Determine the motor position

order:

AA 02 4C 04 08 25 BB

respond:

AA 05 57 00 1A B6 64 6E CD BB

explain:

Once the motor receives the command, it sends its current position every 10 ms.

Parameter Analysis:

Pos=(float)buffer_get_float (data, &ind))

3. Retrieve one or multiple parameters of the motor

order:

AA 05 13 00 00 00 01 FA A9 BB // Read MOS temperature AA 05 13 00 00 00 01 FA A9 BB // Read MOS temperature

respond:

AA 07 13 00 00 00 01 01 21 DF BB BB // Feedback MOS temperature

explain:

To retrieve parameters for one or multiple motors, the number of parameters to be retrieved is determined by the bit at position 4 in the command data field: if the bit at that position is set to 1, the motor will return the corresponding parameter; if it is set to 0, that field is skipped (parameter retrieval begins from bit position 0).

The corresponding motor parameters are listed in the following table:

data bit	31–18 bit	17 bit	16 bit	15 bit	14–9 bit	8 bit	7 bit	6 bit
explain	retention	any power-generating or power-driven machine ID 1byte	Motor position (4 bytes)	Motor error flag 1byte	retention	input voltage 2byte	motor speed 4byte	Duty cycle 2byte
data bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit		
explain	Iq current: 4 bytes	Id current 4byte	input current 4byte	output 4byte	Motor temperature 2byte	MOS temperature 2byte		

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

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

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

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

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

Motor throttle value = (float) buffer_get_int16(data, &ind) / 1000.0)

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

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

Motor position = (float) buffer_get_float(data, &ind)

Motor ID number = data

Motor error code:

```
typedef enum {
```

```

    FAULT_CODE_NONE = 0,
    FAULT_CODE_OVER_VOLTAGE, // Overvoltage
    FAULT_CODE_UNDER_VOLTAGE, // Under-voltage
    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, // Drive under-voltage fault
    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 over-limit fault
    FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE, // Encoder over-limit fault
    FAULT_CODE_ENCODER_SINCOS_BELOW_MIN_AMPLITUDE, // Encoder over-limit fault
    FAULT_CODE_ENCODER_SINCOS_ABOVE_MAX_AMPLITUDE, // Encoder over-limit fault
    FAULT_CODE_FLASH_CORRUPTION, // FLASH fault
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_1, // Fault in current sampling channel 1
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_2, // Fault in current sampling channel 2
    FAULT_CODE_HIGH_OFFSET_CURRENT_SENSOR_3, // Fault in current sampling channel 3
    FAULT_CODE_UNBALANCED_CURRENTS, // Current imbalance

} mc_fault_code;

```

2.3.2.2 Control Commands

1. Duty cycle transmission mode

command:

AA 05 46 00 00 4E 20 D6 4C BB // 0.20 duty cycle

AA 05 46 FF FF B1 E0 88 3F BB // -0.20 Duty cycle

Parameter Analysis:

Duty cycle = (float)buffer_get_int32(data, &ind) / 100000.0 // Value is the number of 4-bit data samples received divided by 100000.0

2. Current loop transmission mode

command:

AA 05 47 00 00 13 88 30 1C BB //5 A IQ Current

AA 05 47 FF FF EC 78 58 3C BB //-5 A IQ Current

Parameter Analysis:

Current = (float) buffer_get_int32(data, &ind) / 1000.0 // Value: received 4-bit data divided by 1000.0

3. Brake Current Transmission Mode

Command:

AA 05 48 00 00 13 88 55 E5 BB //5A Brake current

AA 05 48 FF FF EC 78 3D C5 BB //-5A Brake current

Parameter Analysis:

I_Brake=(float)buffer_get_int32(data,&ind) / 1000.0 // Value: receive 4-bit data / 1000.0

4. Speed loop transmission mode

command:

AA 05 49 00 00 03 E8 90 61 BB // 1000 ERPM (Electrical RPM)

AA 05 49 FF FF FC 18 F8 41 BB // -1000 ERPM (electrical RPM)

Parameter Analysis:

Speed = (float) buffer_get_int32(data, &ind) // Returns the value of the 4-bit data received

5. Position Loop Transmission Mode

command:

AA 05 4A 0A BA 95 00 E1 4D BB // Motor rotates to 180 degrees

AA 05 4A 05 5D 4A 80 84 93 BB // Motor rotates to 90 degrees

Parameter Analysis:

Pos = (float) buffer_get_int32(data, &ind) / 1000000.0 // Value obtained by dividing the received 4-bit data by 1,000,000.0

6. Handbrake current transmission mode

command:

AA 05 4B 00 00 13 88 BB 37 BB //5A HB current

AA 05 4B FF FF EC 78 D3 17 BB //-5A HB current

Parameter Analysis:

HAND_Brake= (float) buffer_get_int32(data, &ind) / 1000.0 // Value: receive 4-bit data / 1000.0

7. Position-Velocity Loop Transmission Mode

command:

AA 0D 3C 00 02 BF 20 00 00 13 88 00 00 75 30 18 1C BB

Parameter Analysis:

Position + Velocity + Acceleration

180° Rotation Speed: 5000 ERPM

Acceleration: 30000 ERPM/s

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

// Value = received 4-byte data / 1000.0

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

// Value = received 4-byte data

Acc_Speed = (float)buffer_get_int32(data, &ind)

// Value = received 4-byte data

8. MIT Transmission Mode

command:

AA 15 60 00 00 00 00 00 00 17 70 00 00 00 00 00 00 00 00 07 D0 93 DA BB

Set Kd to 2 and the angular velocity to 6 rad/s.

AA 15 60 00 00 00 00 FF FF E8 90 00 00 00 00 00 00 00 00 07 D0 87 5C BB

Set Kd to 2 and the angular velocity to -6 rad/s.

Parameter Analysis:

Position + Velocity + Torque + KP + KD

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

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

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

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

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

data conversion:

```
int16_t buffer_get_int16(uint8_t* buffer, int32_t *index)
```

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

```
uint16_t buffer_get_int16(uint8_t* buffer, int32_t *index)
```

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

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

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

```
uint32_t buffer_get_uint32(uint8_t* buffer, int32_t *index)
```

```
{
    uint32_t res = ((uint32_t)buffer[*index])<<24|
```

```
((uint32_t)buffer[*index+1])<<16|
((uint32_t)buffer[*index+2])<<8|
((uint32_t)buffer[*index+3]));

return res;
}

float buffer_get_float(uint8_t* buffer, int32_t *index)
{
    float *res;
    uint32_t data;
    data = buffer_get_uint32(buffer,index);
    res = (float*)&data;
    return *res;
}

//Int16 data bit alignment
void buffer_append_int16(uint8_t* buffer, int16_t number, int32_t *index) {
    buffer[( *index )++] = number >> 8;
    buffer[( *index )++] = number;
}

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

//Int32 data bit alignment
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;
}

//uint32 data bit alignment
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;
}
}
```

2.3.2.3 Serial Port Check

```
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 };
```

```
//CRC verification
```

```
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;
}
```

```
//Packaging and transmission of data packets
```

```
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++] = 0xAA;
    handler_states[handler_num].tx_buffer[b_ind++] = len;
} else {
    handler_states[handler_num].tx_buffer[b_ind++] = 0xAB;
    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++] = 0xBB;

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

```

2.4 Control Command Examples

2.4.1 CAN Control Command Examples

Frame Format: Extended Frame – Data Frame (using AK10-9 motor ID 0x68 as an example)

Mode	ID	DATA	Explain
Duty cycle mode	00 00 00 68	00 00 4E 20	0.2 duty cycle
	00 00 00 68	FF FF B1 E0	-0.2 duty cycle
Current loop	00 00 01 68	FF FF F0 60	-4A IQ current
	00 00 01 68	00 00 0F A0	4A IQ current

Brake current mode	00 00 02 68	FF FF F0 60	-4A braking current
	00 00 02 68	00 00 0F A0	4A braking current
Speed loop	00 00 03 68	00 00 13 88	5000 ERPM (electrical revolutions per minute)
	00 00 03 68	FF FF EC 78	-5000 ERPM (electrical rotational speed)
Position loop	00 00 04 68	00 5B 8D 80	The motor has rotated to 600 degrees.
	00 00 04 68	FF A4 72 80	The motor has rotated to-600 degrees.
Position-velocity loop	00 00 06 68	00 98 96 80 03 E8 03 E8	The motor has rotated to 1000 degrees. 10,000 ERPM (electrical rotational speed) 10,000 electrical acceleration
	00 00 06 68	FF 67 69 80 FC 18 FC 18	The motor has rotated to-1000 degrees. -10,000 ERPM (electrical rotational speed) -10,000 Electrical Acceleration
MIT Speed Loop	00 00 08 68	00 06 66 7F FF 98 67 FF	Kd is set to 2 Set the speed to 6 rad/s
	00 00 08 68	00 06 66 7F FF 64 87 FF	Kd is set to 2 Set the speed to-6 rad/s
MIT Position Loop	00 00 08 68	01 06 66 BD 70 7F F7 FF	Set Kp to 2 Kd is set to 2 The motor has rotated to 6 rad.
	00 00 08 68	01 06 66 42 8F 7F F7 FF	Set Kp to 2 Kd is set to 2 The motor has rotated to-6 rad.
MIT Torque Ring	00 00 08 68	00 00 00 7F FF 7F F8 4B	Torque: 2 N

	00 00 08 68	00 00 00 7F FF 7F F8 97	Torque: 4 N
--	-------------	-------------------------	-------------

2.4.2 Serial Port Control Command Examples

Mode	Serial port command	Explain
Duty cycle mode	AA 05 46 00 00 4E 20 D6 4C BB	0.20 duty cycle
	AA 05 46 FF FF B1 E0 88 3F BB	-0.20 Duty cycle
Current loop	AA 05 47 00 00 13 88 30 1C BB	5 A IQ current
	AA 05 47 FF FF EC 78 58 3C BB	-5 A IQ current
Brake current mode	AA 05 48 00 00 13 88 55 E5 BB	5A braking current
	AA 05 48 FF FF EC 78 3D C5 BB	-5A braking current
Speed loop	AA 05 49 00 00 03 E8 90 61 BB	1000 ERPM (electrical revolutions per minute)
	AA 05 49 FF FF FC 18 F8 41 BB	-1000 ERPM (electrical revolutions per minute)
Position loop	AA 05 4A 0A BA 95 00 E1 4D BB	The motor has rotated 180 degrees.
	AA 05 4A 05 5D 4A 80 84 93 BB	The motor has rotated to 90 degrees.
Position-velocity loop	AA 0D 3C 00 02 BF 20 00 00 13 88 00 00 75 30 18 1C BB	180 degrees Speed: 5000 RPM Acceleration: 30,000 ERPM/s
MIT Speed loop	AA 15 60 00 00 00 00 00 00 17 70 00 00 00 00 00 00 07 D0 93 DA BB	Kd is set to 2 Set the speed to 6 rad/s
	AA 15 60 00 00 00 00 FF FF E8 90 00 00 00 00 00 00 07 D0 87 5C BB	Kd is set to 2 Set the speed to -6 rad/s
MIT Position loop	AA 15 60 00 00 17 70 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 07 D0 E3 A9 BB	Set Kp to 2 Kd is set to 2 The motor has rotated to 6 rad.
	AA 15 60 FF FF E8 90 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 07 D0 BB 05 BB	Set Kp to 2 Kd is set to 2 The motor has rotated to -6 rad.

MIT Torque Loop	AA 15 60 00 00 00 00 00 00 00 00 00 00 00 07 D0 00 00 00 00 00 00 00 00 00 B9 A2 BB	Torque: 2 N
	AA 15 60 00 00 00 00 00 00 00 00 00 00 00 0F A0 00 00 00 00 00 00 00 00 00 58 32 BB	Torque: 4 N