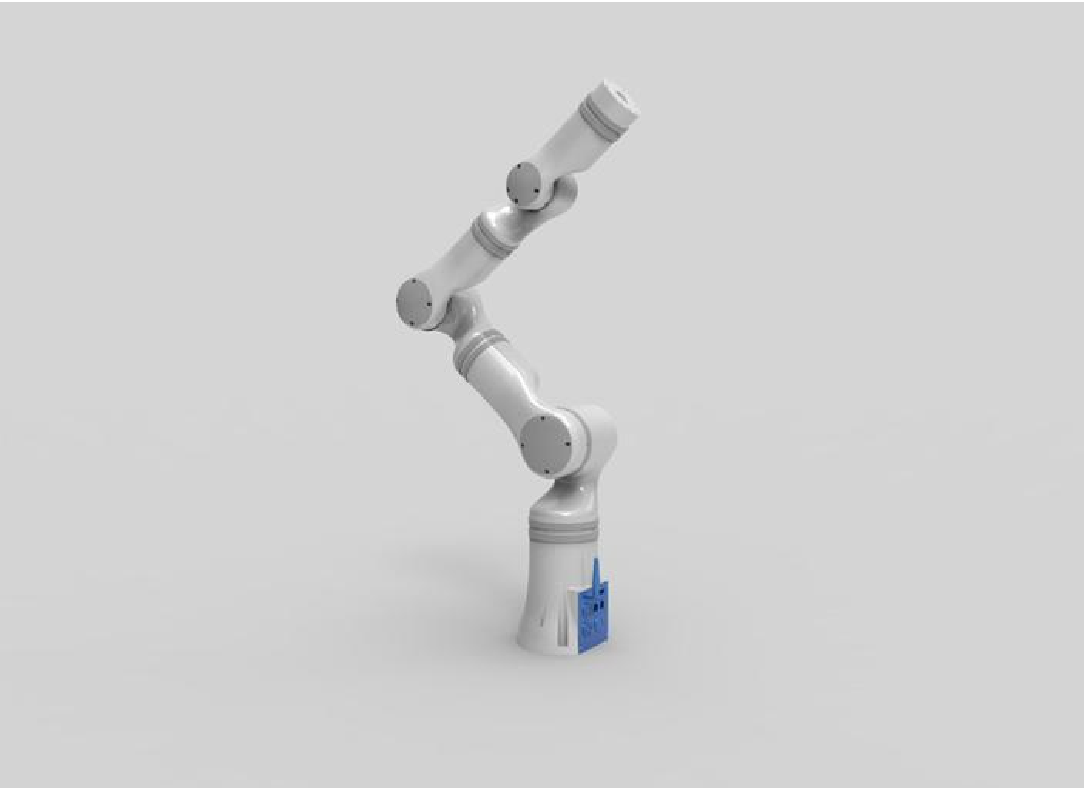
RM robot arm Interface Function Description

V3.1



RealMan Intelligent Technology (Beijing) Co., Ltd.

Revision Record

Version	Date	Comment
V1.0	2020-05-01	Init
V1.1	2020-05-10	Amend
V1.2	2020-05-15	Amend (Generalized revision)
V1.3	2020-05-17	Modify some format and organize some description
V1.4	2020-05-25	Organize some format
V1.5	2020-06-05	Modify the procedure of WIFI configuration
V1.6	2020-06-30	Add the IO protocol
V1.7	2020-07-03	Modify some protocol names
V1.8	2020-08-04	Add some protocol of arm-end interface
V1.9	2021-03-12	Add functions, e.g., path point control etc.
V2.0	2021-05-20	Add Movej_P; Add arm-end PWM setting; Add the one-axis force sensor setting
V2.1	2021-09-17	Add Modbus protocol
V2.2	2021-09-27	Add the collision protection level setting. Add mobile platform and lift.
V2.3	2021-10-19	Fix known errors
V2.4	2022-01-28	Update six-axis and one-axis force function
V2.5	2022-04-12	Fix known errors
V2.6	2022-04-25	Add pose passthrough
V3.0	2022-05-13	Merge 6 axis and 7 axis robot arm interfaces;Add algorithm tool interfaces;Optimize some interface
V3.1	2022-06-09	Correct errors

Contents

1. Introduction

To facilitate the developers to control the robot by the host computer, RealMan (RM) provides the TCP/IP Socket interface function. The user can establish communication with the robot through WIFI (AP mode or STA mode) and Ethernet port to control the robot.

2. Function Introduction

Interface functions are divided into Windows version and Linux version, which can be directly loaded into C++ or C projects for use. Interface functions encapsulate user instructions into standard JSON format and send them to the robot arm, and parse the data returned by the robot arm for the user.

The interface function is written based on TCP/IP protocol, where Robot arm IP address: 192.168.1.18, port number: 8080.

No matter in WiFi mode or Ethernet mode, the robot carries out socket communication with the specified IP and port number. The robot is in Server mode and the user is in Client mode. In addition, to facilitate user debugging, the robot can also switch to USB mode where the robot communicates with the outside through the UART-USB interface. The user can connect the computer through the USB cable, and control the robot by sending commands through the serial port of the host computer according to the JSON Control Protocol.

3. Usage Instruction

The interface function package contains two folders:

(1) json_linux_api: Linux OS interface function;

(2) json_windows_api： Windows OS interface function.

The two parts are identical except for a slight difference in calling the system Socket library.

The composition of APIs is shown in the following figure:

(1) cJSON.h： Header files for the cJSON library. They define the structure, data types, and functions of the cJSON library.

(2) rm_define.h： A custom header file for the robot that contains the defined data types, structures, and error codes.

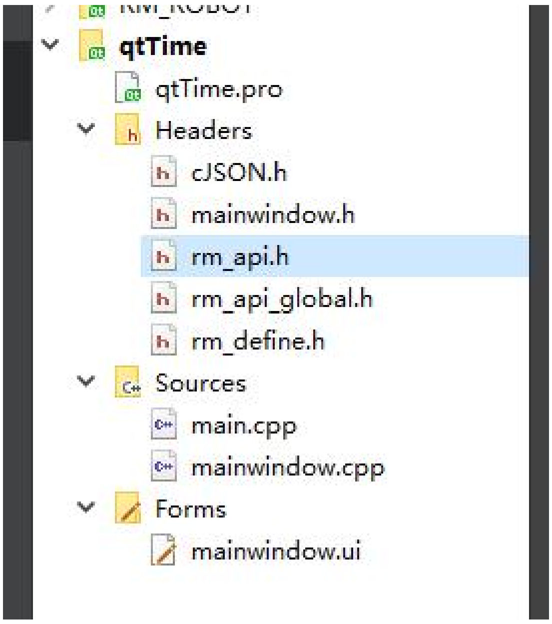
(3) rm65_api.h, rm-65_api_global.h: Interface function definition.

(4) rm-65_API.dll：Interface function implementation.

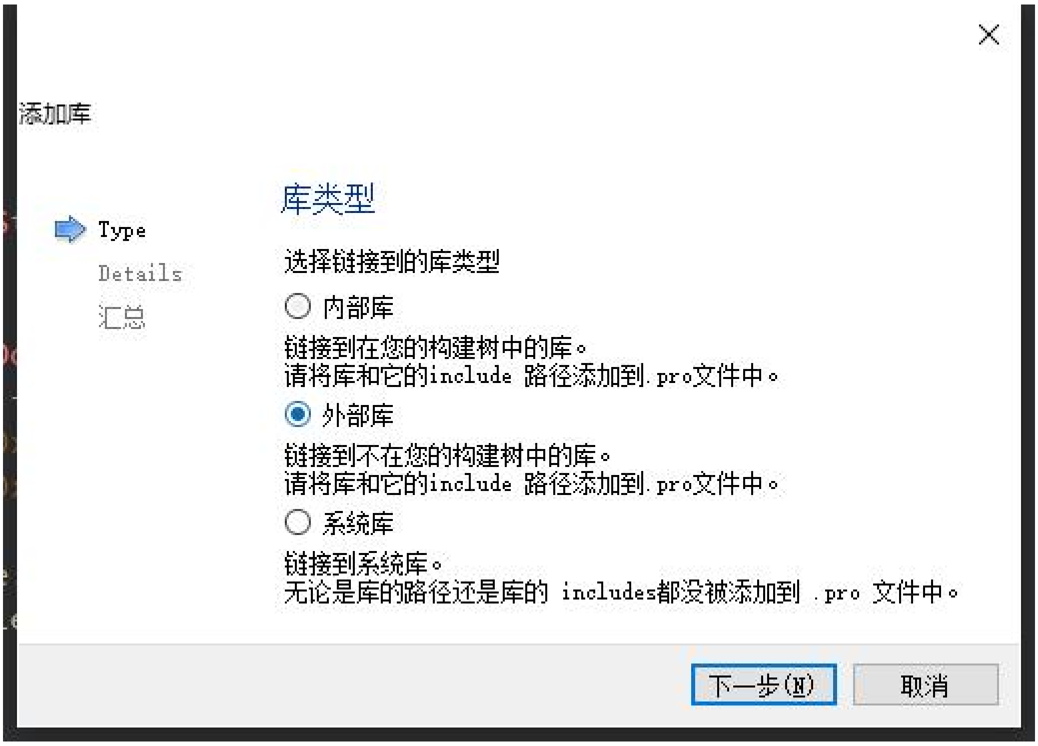
The header files of the APIs are handled and can be loaded directly into a C++ or C project for use, but calling the function are slightly different for Windows and Linux operating systems.

Use steps :

Step 1： First, copy all. h files to the project directory, right-click the project to add existing files.



Step 2：（linux）right-click the project, add library, external library.



Select the dll file and its path.

（win）Copy the dll file to compile directory, write dll file path in pro file.

```
LIBS+=E:\QT\build-qtTime-Desktop_Qt_5_9_1_MinGW_32bit-Debug\debug\RM_API.dll
```

Step 3： After the addition is completed, the Header files can be loaded into the project to use.

```
#include "rm_api.h"
#include "rm_api_global.h"
#include "rm_define.h"
```

Code use examples :

```
    ui->setupUi(this);

    RM_API* api = new RM_API(6); //声明使用六关节函数

    std::string version = api->API_Version(); //获取版本信息
    std::cout << version <<std::endl;

    int a = api->Arm_Socket_Start("192.168.1.18",8080,200);//连接机械臂

    float joint[6] = {10,20,30,40,50,60};
    a = api->Movej_Cmd(joint,50,0,1); //机械臂movej运动
    std::cout << a <<std::endl;

    api->Arm_Socket_Close();

    delete api;
}
```

```
Debugging starts
3.0
0
```

The use process is as follows:

(1) Connect with the robot arm through WIFI or Ethernet port to ensure that the host computer and the robot arm controller are in the same network.

(2) Call the Arm_Socket_Start() function to connect the socket with the robot arm. Note that for the Windows operating system, it may take 2~3 times to establish the connection. Therefore, it is necessary to call the function again to ensure the successful socket connection according to the return of this function. In addition, the timeout argument needs to be input when the function is called. This argument is used to report an error and exit the function if the client does not receive the data during the timeout time of the robot in the blocking mode. The recommended timeout is not less than 20ms.

(3) After successful connection, the user calls the interface function as needed.

(4) After use, call Arm_Socket_Close() function to close the socket connection(s) and release system resources.

4. Macro Definition

4.1 Controller Error Types

#	Error Code	Comment
1	0x0000	system is normal
2	0x0001	target angle exceeds the joint limit
3	0x0002	inverse kinematics error, inaccessible
4	0x0004	real-time kernel communication error
5	0x0008	cannot communicate with the robot arm end interface
6	0x0010	inverse kinematics of joint overspeed
7	0x0020	the robotic collides
8	0x0040	the robot has an emergency stop
9	0x0080	system memory overflow
10	0x0100	error in SD card initialization
11	0x0200	error in WIFI module initialization
12	0x0400	the system status is not updated in time
13	0x0800	error in temperature sensor initialization
14	0x1000	controller over temperature
15	0x2000	controller over current
16	0x4000	controller over voltage
17	0x8000	controller under voltage

4.2 Joint(s) Error Types

#	Error Code	Comment
1	0x0000	system is normal
2	0x0001	FOC frequency is too large
3	0x0002	the system voltage exceeds the safe range

4	0x0004	the system voltage is below the safe range
5	0x0008	temperature is too high
6	0x0010	start failed
7	0x0020	encoder error
8	0x0040	motor current exceeds safe range
9	0x0080	software error
10	0x0100	temperature sensor error
11	0x0200	joint position out of limit
12	0x0400	driver chip error
13	0x0800	position error tracking over-limit protection
14	0x1000	current sensor detects error during power on
15	0x2000	joint locking brake error
16	0xF000	joint frame(s) lost

4.3 Data Structure Definition

4.3.1 POSE

Struct: Position and Pose.

```
typedef struct
{
	//Position
	float px;
	float py;
	float pz;
	//Euler angle
	float rx;
	float ry;
```

```
    float rz;
}POSE;
```

Struct Member

px,py,pz

The coordinates of the position， type: float, unit： m

rx,ry,rz

The angles of the pose， type: float, unit： rad

4.3.2 FRAME

Coordinate system.

```
typedef struct
{
    char *frame_name;
    POSE pose;
    float   payload;
    float   x;
    float   y;
    float   z
}FRAME;
```

Struct Member

frame_name

The name of the coordinate system， string.

Pose

The position and pose of the coordinate system. See 1.5.1.

payload

End payload (g)

x,y,z

End payload position

4.3.3 JOINT_STATE

```
typedef struct
{
	float joint[6];
	float temperature[6];
	float voltage[6];
	float current[6];
	byte en_state[6];
	uint16_t err_flag[6];
	uint16_t sys_err;
}JOINT_STATE;
```

Struct Member

joint

The angles of each joint，type: float，unit：degree(°).

temperature

Joint temperature，type: float，unit：Celsius degree.

voltage

Joint voltage，type: float，unit：V

current

Joint current，type: float，unit：mA

en_state

Enabling status.

err_flag

Joint error code，type: unsigned int

sys_err

Robot arm system error code，type: unsigned int

4.3.4 ARM_CTRL_MODES

Robot arm control modes

```
typedef enum
{
    None_Mode = 0,
    Joint_Mode = 1,
    Line_Mode = 2,
    Circle_Mode = 3,
}ARM_CTRL_MODES;
```

Enum Member

None_Mode

Non-planning mode

Joint_Mode

Joint spatial planning mode

Line_Mode

Cartesian space linear planning model

Circle_Mode

Cartesian space circular planning model

4.3.5 POS_TEACH_MODES

Robot arm position teaching mode

```
typedef enum
{
    X_Dir = 0,
    Y_Dir = 1,
    Z_Dir = 2,
}POS_TEACH_MODES;
```

Enum Member

X_Dir

X axis， 0 by default

Y_Dir

Y axis， 1 by default

Z_Dir

Z axis， 2 by default

4.3.6 ORT_TEACH_MODES

Robot arm pose teaching mode

```
typedef enum
{
    RX_Rotate = 0,
    RY_Rotate = 1,
    RZ_Rotate = 2,
}ORT_TEACH_MODES;
```

Enum Member

RX_Rotate

X axis， 0 by default

RY_Rotate

Y axis， 1 by default

RZ_Rotate

Z axis， 2 by default

4.3.7 ARM_COMM_TYPE

Controller communication method selection

```
typedef enum
{
```

```
	WIFI_AP = 0,
	WIFI_STA = 1,
	BlueTeeth = 2,
	USB          = 3,
	Ethernet   =4
}ARM_COMM_TYPE;
```

Enum Member

WIFI_AP

WIFI in AP mode

WIFI_STA

WIFI in STA mode

BlueTeeth

Bluetooth mode

USB

Communicate via controller’s UART-USB port

Ethernet

Ethernet port

5. Interface Library Function Description

This function is used to control the opening and closing of the socket connection.

5.1 socket related functions

5.1.1 RM_API

This function is used to set the type of the robot arm.

RM_API(int Joint_Number);

Argument

①Joint_Number

Set the number of the joints of the robot arm:RM-65 or RML63 set the parameter to 6 and RM-75 set the parameter to 7 .

5.1.2 Arm_Socket_Start

This function is used to connect the robot arm.

Arm_Socket_Start(char* arm_ip,int arm_prot,int recv_timeout);

Argument

①arm_ip

Target robot's ip. The default ip address of the robot arm is 192.168.1.18 .

②arm_prot

Target robot's prot. The default port number of the robot arm is 8080.

③recv_timeout

Sets the timeout time for receiving robot data to prevent the program from blocking all the time. Unit: ms。No less than 50ms as recommended.

Return

0-Succeeded，1-Failed

5.1.3 Arm_Sockrt_State

This function is used to get the connection status of the robot arm.

int Arm_Sockrt_State();

Return

0-Succeeded， 1-Failed

5.1.4 Arm_Socket_Close

This function is used to disconnect the robot arm.

void Arm_Socket_Close();

5.1.5 Arm_Socket_Buffer_Clear

This function is used to clean up the data in the socket receive cache.

void Arm_Socket_Buffer_Clear();

5.2 Joint configuration functions

Note: All parameters of the RM robot arm have been configured to the best state before leaving the factory, and it is not recommended that the user modify the underlying parameters of the joint. If the user really needs to modify, first of all, the robot arm should be in a disenabled state, and then send a modified parameter instruction, after the parameter setting is successful, send the joint recovery enable instruction. It should be noted that when the joint is restored, the user needs to ensure that the joint is in a static state to avoid positioning errors in the joint during the activation process. After the joint is enable, the user can control the joint movement.

5.2.1 Set_Joint_Speed

This function is used to set the maximum joint velocity， unit： RPM。

int Set_Joint_Speed(byte joint_num, float speed, bool block);

Argument

①joint_num

Joint number,1~6

②speed

Joint rotation speed， unit： RPM

③block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.2.2 Set_Joint_Acc

This function is used to set the maximum joint acceleration，unit：RPM/s。

int Set_Joint_Acc(byte joint_num, float acc, bool block);

Argument

①joint_num

Joint number，1~6

②acc

Joint acceleration，unit：RPM/s

③block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.2.3 Set_Joint_Min_Pos

This function is used to set the minimum position that the joint can reach，unit：degree.

int Set_Joint_Min_Pos(byte joint_num, float joint, bool block);

Argument

①joint_num

Joint number，1~6

②joint

The minimum position of the joint，unit：°

③block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.2.4 Set_Joint_Max_Pos

This function is used to set the maximum position that the joint can reach.

int Set_Joint_Max_Pos(byte joint_num, float joint, bool block);

Argument

①joint_num

Joint number，1~6

②joint

The maximum position of the joint，unit：°

③block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.2.5 Set_Joint_EN_State

This function is used to set the enabling state of the joint.

int Set_Joint_EN_State(byte joint_num, bool state, bool block);

Argument

①joint_num

Joint number，1~6

②state

true-Enabling，false-Disabling.

③block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.2.6 Set_Joint_Zero_Pos

This function is used to set the current position as the joint origin.

int Set_Joint_Zero_Pos(byte joint_num, bool block);

Argument

①joint_num

Joint number， 1~6

②block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.2.7 Set_Joint_Err_Clear

This function is used to clear the joint error code.

int Set_Joint_Err_Clear(byte joint_num, bool block);

Argument:

①joint_num

Joint number， 1~6

②block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return:

0-Succeeded， 1-Failed.

5.2.8 Start_Calibrate

This function is used to start the joints calibration.

int Start_Calibrate(bool block);

Argument:

①block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return:

0-Succeeded，1-Failed.

5.2.9 Stop_Calibrate

This function is used to stop the joints calibration.

int Stop_Calibrate(bool block);

Argument:

① block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return:

0-Succeeded，1-Failed.

5.2.10 Get_Calibrate_State

This function is used to query the status of the joints calibration.

int Get_Calibrate_State(float *joint_state, uint16_t *state,int* time,bool block);

Argument:

① joint_state

The status of the joints. 0-not calibrated, 1-in calibration, 2-calibrated, 3-joint stucked,4-calibration timeout, 5-error.

② err

Error information of each joint.

0x0001FOC frequency too high

0x0010 start-up failure

0x0100 temperature sensor error

0x0002 overvoltage

0x0020 code plate error

0x0200 position overrun error

0x0004 undervoltage

0x0040 overcurrent

0x0400 DRV8320 error

0x0008 overtemperature

0x0080 software error

0x0800 position tracking error overrun

0x1000 Current detection error

③time

Calibration duration (s).

④block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return:

0-Succeeded，1-Failed.

5.3 Joint parameter query functions

5.3.1 Get_Joint_Speed

This function is used to query the maximum joint velocity.

int Get_Joint_Speed(float *speed);

Argument

①*speed

The array address that stores the speed of Joint 1~6，unit：RPM

Return

0-Succeeded，1-Failed

5.3.2 Get_Joint_Acc

This function is used to query the maximum joint acceleration.

int Get_Joint_Acc(float *acc);

Argument

①*acc

The array address that stores the accelerations of Joint 1~6，unit：RPM/s

Return

0-Succeeded，1-Failed

5.3.3 Get_Joint_Min_Pos

This function is used to obtain the minimum joint position.

int Get_Joint_Min_Pos(float *min_joint);

Argument

① *min_joint

The array address that stores the min positions of Joint 1~6，unit：°

Return

0-Succeeded，1-Failed

5.3.4 Get_Joint_Max_Pos

This function is used to obtain the maximum joint position.

int Get_Joint_Max_Pos(float *max_joint);

Argument

① *max_joint

The array address that stores the max positions of Joint 1~6，unit：°

Return

0-Succeeded，1-Failed

5.3.5 Get_Joint_EN_State

This function is used to obtain the enabling state.

int Get_Joint_EN_State(byte *state);

Argument

① *state

The array address that stores the enabling state of Joint 1~6， 1-enabling，0-disabling.

Return

0-Succeeded，1-Failed

5.3.6 Get_Joint_Err_Flag

This function is used to obtain the joint error code.

int Get_Joint_Err_Flag(uint16_t *state);

Argument

①*state

It contains the joint error code.

Return

0-Succeeded， 1-Failed

5.3.7 Get_Joint_Software_Version

This function is used to query the joint software version number.

int Get_Joint_Software_Version(float* version);

Argument

①version

The software version numbers of joint 1~6.

Return

0-Succeeded， 1-Failed

5.4 Configuration of the arm-end movement

5.4.1 Set_Arm_Line_Speed

This function is used to set the maximum linear velocity of the arm end.

int Set_Arm_Line_Speed(float speed, bool block);

Argument

①speed

The maximum liner speed of the arm end， unit: m/s

②block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.2 Set_Arm_Line_Acc

This function is used to set the maximum linear acceleration of the arm end.

int Set_Arm_Line_Acc(float acc, bool block);

Argument

① acc

The maximum linear acceleration of the arm end， unit: m/s^2.

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.3 Set_Arm_Angular_Speed

This function is used to set the maximum angular velocity of the arm end.

int Set_Arm_Angular_Speed(float speed, bool block);

Argument

① speed

The maximum angular speed of the arm end， unit: rad/s

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.4 Set_Arm_Angular_Acc

This function is used to set the maximum angular acceleration of the arm end.

int Set_Arm_Angular_Acc(float acc, bool block);

Argument

① acc

The maximum angular acceleration of the arm end， unit: rad/s^2

②block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.5 Get_Arm_Line_Speed

This function is used to obtain the line speed of the robot arm end.

int Get_Arm_Line_Speed(float *speed);

Argument

①speed

The line speed of each joint end.

Return

0-Succeeded， 1-Failed

5.4.6 Get_Arm_Line_Acc

This function is used to obtain the line acceleration of the robot arm end.

int Get_Arm_Line_Acc(float *acc);

Argument

①acc

The line acceleration of each joint end.

Return

0-Succeeded， 1-Failed

5.4.7 Get_Arm_Angular_Speed

This function is used to obtain the angular speed of the robot arm end.

int Get_Arm_Angular_Speed(float *speed);

Argument

①speed

The angular speed of each joint end.

Return

0-Succeeded， 1-Failed

5.4.8 Get_Arm_Angular_Acc

This function is used to obtain the line speed of the robot arm end.

int Get_Arm_Angular_Acc(float *acc);

Argument

①acc

The angular acceleration of each joint end.

Return

0-Succeeded， 1-Failed

5.4.9 Set_Arm_Tip_Init

This function is used to set the arm-end parameters as their initial values.

int Set_Arm_Tip_Init(bool block);

Argument

①block:0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.10 Set_Collision_Stage

This function is used to set the dynamic collision level of the robot arm. Level 0~8. The higher the value is, the more sensitive the collision detection is, and at the same time, it is more likely to occur false collision detection. The default collision level is 0 after the robot arm is energized, that is, the collision is not detected.

int Set_Collision_Stage (int stage, bool block);

Argument

①stage

Collision detection level： 0~8

②block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.4.11 Get_Collision_Stage

This function is used to query the robot arm dynamic collision level, grade 0 to 4, the higher the value, the more sensitive the collision detection, but also more prone to accidental collision detection. The default collision level after the robot arm is powered on is 0, i.e., no collision is detected.

int Get_Collision_Stage (int stage, bool block);

Argument

① stage

Collision detection level：0~8

② block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.4.12 Set_DH_Data

This function is used to set DH parameters. It is generally used after the calibration of the robot parameters.

int Set_DH_Data (float lsb, float lse, float lew, float lwt, float d3, bool block);

Argument

① lsb, lse, lew, lwt, d3

DH，unit：mm

② block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.13 Get_DH_Data

This function is used to obtain robot's DH parameters.

int Get_DH_Data (float* lsb, float* lse, float* lew, float* lwt, float* d3);

Argument

①lsb, lse, lew, lwt, d3

DH parameters, unit: mm

Return

0-Succeeded, 1-Failed

5.4.14 Set_Joint_Zero_Offset

This function is used to set the origin compensation angle of each joint. It is generally called after the calibration of robot origin position.

int Set_Joint_Zero_Offset (float *offset, bool block);

Argument

①offset

Robot origin compensation angle array of Joint1~6， unit： degree.

②block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.4.15 Set_Arm_Dynamic_Parm

This function is used to set robot arm's dynamic parameters.

int Set_Arm_Dynamic_Parm (float *offset, bool block);

Argument

①offset

Joint 1 to 6 dynamic parameters.

②block

0-Non-blocking, returning immediately after sent; 1-Blocking, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded, 1-Failed

5.5 Robot Arm End Interface Board

5.5.1 Get_Tool_Software_Version

This function is used to query the end interface board software version number.

int Get_Tool_Software_Version(int *version);

Argument

①version

The end interface board software version number

Return

0-Succeeded, 1-Failed

5.6 Robot arm state servo configuration

5.6.1 Set_Arm_Servo_State

This function is used to turn on or off the controller to query the servo state of the robot arm and control the data load rate of CANFD bus.

int Set_Arm_Servo_State(bool cmd, bool block);

Argument

①cmd:

true-On, false-Off

②block:

0-Non-blocking, returning immediately after sent; 1-Blocking, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.7 Tool coordinate system configuration

5.7.1 Auto_Set_Tool_Frame

This function is used to automatically set the tool coordinate system using the six-point method, particularly for recording calibration points. The robot controller can only store 10 tool information at most. Before creating a new tool, please make sure that the number of tools does not exceed the limit, otherwise the new tool cannot be created successfully.

int Auto_Set_Tool_Frame(byte point_num, bool block);

Argument

① point_num

1~6 denote the six calibration points.

② block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: the controller can only store ten tools. When tool number exceeds ten, controllers will not respond. Please check the existing tools before calibration.

5.7.2 Generate_Auto_Tool_Frame

This function is used to automatically set the tool coordinate system using the six-point method, particularly for generating coordinate system. The robot controller can only store 10 tool information at most. Before creating a new tool, please make sure that the number of tools does not exceed the limit, otherwise the new tool cannot be created successfully.

int Generate_Auto_Tool_Frame(char *name, float payload,float x,float y,float z, bool block);

Argument

①name

Name of the tool coordinate system. Must be less 10 characters.

②payload

Tool end payload (g).

③x,y,z

Tool end position.

④block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: the controller can only store ten tools. When tool number exceeds ten, controllers will not respond. Please check the existing tools before calibration.

5.7.3 Manual_Set_Tool_Frame

This function is used to manually set the tool coordinate system. The robot controller can only store 10 tool information at most. Before creating a new tool, please make sure that the number of tools does not exceed the limit, otherwise the new tool cannot be created successfully.

int Manual_Set_Tool_Frame(char *name, POSE pose, float payload,float x,float y,float z, bool block);

Argument

①name

Tool coordinate system name, no more than ten bytes.

②pose

Position of the executive end of the new tool relative to the center of the flange of the robot arm.

③ **payload**

Tool end payload (g).

④ **x,y,z**

Tool end position.

⑤ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: the controller can only store ten tools. When tool number exceeds ten, controllers will not respond. Please check the existing tools before calibration.

5.7.4 Change_Tool_Frame

This function is used to change the current tool coordinate system.

int Change_Tool_Frame(char *name, bool block);

Argument

① **name**

Target tool coordinate system name.

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.7.5 Delete_Tool_Frame

This function deletes the specified tool coordinate system.

int Delete_Tool_Frame(char *name, bool block);

Argument

① **name**

Tool coordinate system name that is to be deleted.

②**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: After deleting the coordinate system, the robot will switch to the tool coordinate system at the flange end of the robot.

5.7.6 Set_Payload

This function is used to set the load mass and center of mass at the arm end.

int Set_Payload(int payload, float cx, float cy, float cz, bool block);

Argument

①**payload**

The load mass at the arm end，unit：g

②**cx, cy, cz**

The center of the load mass at the arm end，unit：mm

③**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.7.7 Set_None_Payload

This function is used to cancel the load at the arm end.

int Set_None_Payload(bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.8 Tool coordinate system query

5.8.1 Get_Current_Tool_Frame

This function is used to get the current tool coordinate system.

int Get_Current_Tool_Frame(FRAME *tool);

Argument

① **tool**

The coordinate system that is to be returned.

Return

0-Succeeded， 1-Failed

5.8.2 Get_Given_Tool_Frame

This function is used to get the specific tool coordinate system.

int Get_Given_Tool_Frame(char *name, FRAME *tool);

Argument

① **name**

The specific tool name

② **tool**

The returned tool parameters

Return

0-Succeeded， 1-Failed

5.8.3 Get_All_Tool_Frame

This function is used to get all the tool coordinate system names.

int Get_All_Tool_Frame(FRAME_NAME *names);

Argument

① **names**

Array of tool names that is to be returned

Return

0-Succeeded，1-Failed

5.9 Operating coordinate system configuration

5.9.1 Auto_Set_Work_Frame

This function is used to automatically set the operating/working coordinate system using the three-point method. The robot controller can only store information of 10 operating coordinate systems at most. Before establishing a new operating coordinate system, please make sure that the number of operating coordinate systems does not exceed the limit, otherwise the establishment of a new operating coordinate system cannot be successful.

int Auto_Set_Work_Frame(char *name, byte point_num, bool block);

Argument

①name

Operating coordinate system name, no more than ten bytes.

②point_num

1~3 denote 3 calibration points，which is the origin, any point on x axis and any point on y axis，respectively. 4 denotes the generation of the coordinate system.

③block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: the controller can only store ten tools. When tool number exceeds ten, controllers will not respond. Please check the existing tools before calibration.

5.9.2 Manual_Set_Work_Frame

This function is used to manually set the operating coordinate system.

int Manual_Set_Work_Frame(char *name, POSE pose, bool block);

Argument

①**name**

Operating coordinate system name, no more than ten bytes.

②**pose**

The position and pose of the new operating coordinate system relative to the base coordinate system.

③**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: the controller can only store ten tools. When tool number exceeds ten, controllers will not respond. Please check the existing tools before calibration.

5.9.3 Change_Work_Frame

This function is used to change the operating coordinate system.

int Change_Work_Frame(char *name, bool block);

Argument

①**name**

Target operating coordinate system

②**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.9.4 Delete_Work_Frame

This function is used to delete the specific operating coordinate system.

int Delete_Work_Frame(char *name, bool block);

Argument

① **name**

Tool coordinate system name that is to be deleted.

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

Note: After deleting the coordinate system, the robot will switch to the Base coordinate system.

5.10 Operating coordinate system query

5.10.1 Get_Current_Work_Frame

This function is used to get the current operating coordinate system.

```
int Get_Current_Work_Frame(FRAME *frame);
```

Argument

① **frame**

Coordinate system

Return

0-Succeeded，1-Failed

5.10.2 Get_Given_Work_Frame

This function is used to get the specific operating coordinate system.

```
int Get_Given_Work_Frame(char *name, POSE *pose);
```

Argument

① **name**

The specific operating coordinate system name

② **pose**

The position and pose parameters of the operating coordinate system that is to be returned

Return

0-Succeeded， 1-Failed

5.10.3 Get_All_Work_Frame

This function is used to get the names of all operating coordinate systems.

int Get_All_Work_Frame(FRAME_NAME *names);

Argument

① names

Array of names of all the operating coordinate systems

Return

0-Succeeded， 1-Failed

5.11 Robot arm status query

5.11.1 Get_Current_Arm_State

This function is used to obtain the current status/state of the robot.

int Get_Current_Arm_State(float*joint, POSE *pose, uint16_t *Arm_Err, uint16_t *Sys_Err);

Argument

① joint

Array of angles of Joint 1~6

② pose

Current position and posture

③ Arm_Err

Robot error code

④ Sys_Err

Controller error code

Return

0-Succeeded， 1-Failed

5.11.2 Get_Joint_Temperature

This function is used to obtain the current temperature of the joint.

int Get_Joint_Temperature(float *temperature);

Argument

①**temperature**

Array of temperature of Joint 1~6

Return

0-Succeeded， 1-Failed

5.11.3 Get_Joint_Current

This function is used to obtain the current current of the joint.

int Get_Joint_Current(float *current);

Argument

①**current**

Array of current of Joint 1~6

Return

0-Succeeded， 1-Failed

5.11.4 Get_Joint_Voltage

This function is used to obtain the current voltage of the joint.

int Get_Joint_Voltage(float *voltage);

Argument

①**voltage**

Array of voltage of Joint 1~6

Return

0-Succeeded， 1-Failed

5.11.5 Get_Joint_Degree

This function is used to get robot joints' angles.

int Get_Joint_Degree (float *joint);

Argument

① joint

Array address of Joint 1 to 6.

Return

0-Succeeded， 1-Failed

5.11.6 Get_Arm_All_State

This function is used to obtain all the status/state of the robot.

int Get_Arm_All_State (JOINT_STATE *joint_state);

Argument

① joint_state

All the status/state of the robot

Return

0-Succeeded， 1-Failed

5.11.7 Get_Arm_Plan_Num

This function is used to get the number of robot arm trace tracking.

int Get_Arm_Plan_Num (int plan);

Argument

① plan

The total number of robot arm trace tracking.

Return

0-Succeeded， 1-Failed

5.12 Robot arm's initial position and pose

5.12.1 Set_Arm_Init_Pose

Function is used to set the initial position and angle of the robot.

int Set_Arm_Init_Pose(float *target, bool block);

Argument

①**target**

Array of joint initial position and angles of the robot

②**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.12.2 Get_Arm_Init_Pose

Function is used to get the initial position and angle of the robot.

int Get_Arm_Init_Pose(float *joint);

Argument

①**joint**

Array of joint initial position and angles of the robot

Return

0-Succeeded，1-Failed

5.12.3 Set_Install_Pose

This function is used to set the installation of the robot arm.

int Set_Install_Pose(float x,float y,bool block);

Argument

①**x**

Rotation angle

②**y**

Pitch angle

③**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction

Return

0-Succeeded， 1-Failed

5.13 Robot arm movement planning

5.13.1 Movej_Cmd

This function is used for joint spatial movement.

int Movej_Cmd(float *joint, byte v, float r, bool block);

Argument

①joint

Array of angles of Joint 1~6

②v

Speed percentage 1~100， i.e. the ratio of the planned speed and acceleration to the maximum linear speed and acceleration of the joint

③r

Trajectory blend radius, currently 0 by default

④block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to reach the target position or the planning failed

Return

0-Succeeded， 1-Failed

5.13.2 Movel_Cmd

This function is used for linear movement in Cartesian space.

int Movel_Cmd(POSE pose, byte v, float r, bool block);

Argument

①pose

Target position and pose， position unit： m， pose unit： rad

②v

Speed percentage 1~100， i.e. the ratio of the planned speed and acceleration to the maximum linear speed and acceleration of the joint

③r

Trajectory blend radius, currently 0 by default

④block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to reach the target position or the planning failed

Return

0-Succeeded， 1-Failed

5.13.3 Movec_Cmd

This function is used for circular movement in Cartesian space.

int Movec_Cmd(POSE pose_via, POSE pose_to, byte v, float r, byte loop, bool block);

Argument

①**pose_vai**

Position and pose of the middle point， position unit： m， pose unit： rad

②**pose_to**

Position and pose of the final point， position unit： m， pose unit： rad

③**v**

Speed percentage 1~100， i.e. the ratio of the planned angular speed and acceleration to the maximum angular speed and angular acceleration of the joint

④**r**

Trajectory blend radius, currently 0 by default

⑤**loop**

Scheduled loop number, currently 0 by default.

⑥**block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to reach the target position or the planning failed

Return

0-Succeeded， 1-Failed

5.13.4 Movej_CANFD

This function is used to transmit the angle directly to the robot through CANFD without planning.

int Movej_CANFD(float *joint, bool block);

Argument

① joint

Array of target angles of Joint 1~6

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， since this mode is directly sent to the robot without controller planning, as long as the controller runs normally and the target angle is within the reachable range, the robot immediately returns success, and the robot may still be running at this time. If there is an error, return failure immediately.

Return

0-Succeeded， 1-Failed

5.13.5 Movep_CANFD

This function is used to transmit the pose directly to the robot through CANFD without planning.

int Movep_CANFD(POSE pose, bool block);

Argument

① pose

Position and posture

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， since this mode is directly sent to the robot without controller planning, as long as the controller runs normally and the target angle is within the reachable range, the robot immediately returns success, and the robot may still be running at this time. If there is an error, return failure immediately.

Return

0-Succeeded，1-Failed

5.13.6 Move_Stop_Cmd

This function is used for emergency situations where the robot stops at the fastest speed and the trajectory is unrecoverable.

int Move_Stop_Cmd(bool block);

Argument

①**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.13.7 Move_Pause_Cmd

This function is used to pause the trajectory. The trajectory can be recovered.

int Move_Pause_Cmd(bool block);

Argument

①**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.13.8 Move_Continue_Cmd

This function is used to continue the current trajectory after the trajectory is paused.

int Move_Continue_Cmd(bool block);

Argument

①**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0- Succeeded，1-Failed

5.13.9 Clear_Current_Trajectory

This function is used to clear the current trajectory and must be used after a pause, otherwise an accident will occur to the robot arm.

int Clear_Current_Trajectory(bool block);

Argument

① block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.13.10 Clear_All_Trajectory

This function is used to clear all the trajectory and must be used after a pause, otherwise an accident will occur to the robot arm.

int Clear_All_Trajectory(bool block);

Argument

① block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.13.11 Get_Current_Trajectory

This function is used to obtain the trajectory information that is currently being planned.

int Get_Current_Trajectory(ARM_CTRL_MODES *type, float *data);

Argument

①type

The type of the planning

②data

If there is no trajectory planning task for the robot, the current angles of Joint 1~6 are returned. If the joint spatial planning task is being performed, the current angles of Joints 1~6 are returned. If Cartesian space planning is being performed, the current position and pose of the arm end are returned.

Return

0-Succeeded， 1-Failed

5.13.12 Movej_P_Cmd

This function is used to move joints to a target pose and position.

int Movej_P_Cmd(POSE pose, byte v, float r, bool block);

Argument

①pose

Target pose and position, position unit: meter, pose unit: radian.

Note: The target pose and position must be the pose and position of the arm-end flange center based on the base coordinate system.

②v

The velocity ratio: 1~100, that is, the percentage of the planned velocity and acceleration to the arm-end maximum linear velocity and linear.

③r

Trajectory blend radius, currently default 0.

④block

0- Non-blocking, returned immediately after command sent; 1- Blocking, waiting for robot arm to reach the target pose and position or planning failure.

Return

0-Succeeded， 1-Failed

5.14 Robot arm teach

5.14.1 Joint_Teach_Cmd

This function is used for joint teaching. The joint rotates in the specified direction from the current position and stops after receiving the stop command or reaching the limit of the joint.

int Joint_Teach_Cmd(byte num, byte direction, byte v, bool block);

Argument

① num

The index of the joint being taught， i.e. 1~6

② direction

Teaching direction， 0-negative direction， 1-positive direction

③ v

Speed percentage 1~100， i.e. the ratio of the planned speed and acceleration to the maximum linear speed and acceleration of the joint.

④ block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.14.2 Pos_Teach_Cmd

This function is used to teach the position of Cartesian space in the current operating coordinate system. Under the current operating coordinate system, the robot starts to move in a straight line according to the direction of the specified coordinate axis, and stops when the stop command is received or there is no inverse solution.

int Pos_Teach_Cmd(POS_TEACH_MODES type, byte direction, byte v, bool block);

Argument

① type

Teaching type

②**direction**

Teaching direction，0-negative direction，1-positive direction

③**v**

Speed percentage 1~100，i.e. the ratio of the planned speed and acceleration to the maximum linear speed and acceleration of the arm end.

④**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.14.3 Ort_Teach_Cmd

This function is used to teach the position of Cartesian space in the current operating coordinate system. The robot rotates around the specified coordinate axis in the current operating coordinate system and stops when the stop command is received or there is no inverse solution.

int Ort_Teach_Cmd(ORT_TEACH_MODES type, byte direction, byte v, bool block);

Argument

①**type**

Teaching type

②**direction**

Teaching direction，0-negative direction，1-positive direction

③**v**

Speed percentage 1~100，i.e. the ratio of the planned angular speed and acceleration to the maximum angular speed and acceleration of the joint.

④**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the

controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.14.4 Teach_Stop_Cmd

This function is used to stop the teaching.

int Teach_Stop_Cmd(bool block);

Argument

①**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.15 Robot arm stepping

5.15.1 Joint_Step_Cmd

This function is used for joint stepping. The joint steps from the current position to a specified angle.

int Joint_Step_Cmd(byte num, float step, byte v, bool block);

Argument

①**num**

Joint number，1~6

②**step**

Stepping angle

③**v**

Speed percentage 1~100，i.e. the ratio of the planned rotation speed and acceleration to the maximum rotation speed and acceleration of the specified joint.

④**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful reach or a failure instruction.

Return

0-Succeeded，1-Failed

5.15.2 Pos_Step_Cmd

This function is used for position step in the current operating coordinate system. Under the current operating coordinate system, the end of the robot steps the specified distance in the direction of the specified coordinate axis and returns success when it reaches the position. It returns failure when there are planning errors.

int Pos_Step_Cmd(POS_TEACH_MODES type, float step, byte v, bool block);

Argument

①type

Teach type

②step

Stepping distance，unit: m

③v

Speed percentage 1~100，i.e. the ratio of the planned speed and acceleration to the maximum linear speed and acceleration of the arm end

④block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful reach or a failure instruction.

Return

0-Succeeded，1-Failed

5.15.3 Ort_Step_Cmd

This function is used for pose step in the current operating coordinate system. Under the current operating coordinate system, the end of the robot steps to the specified radian around the specified coordinate axis direction. When it reaches the position, it returns success when it reaches the position. It returns failure when there are planning errors.

int Ort_Step_Cmd(POS_TEACH_MODES type, float step, byte v, bool block);

Argument

①**type**

Teach type

②**step**

Stepping radian，unit: rad，resolution: 0.001rad

③**v**

Speed percentage 1~100， i.e. the ratio of the planned angular speed and acceleration to the maximum angular speed and acceleration of the arm end

④**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful reach or a failure instruction.

Return

0-Succeeded，1-Failed

5.16 Controller configuration

5.16.1 Get_Controller_State

This function is used to get the status of the controller.

int Get_Controller_State(float *voltage, float *current, float *temperature, uint16_t *sys_err);

Argument

①**voltage**

Voltage

②**current**

Current

③**temperature**

Temperature

④**sys_err**

Controller running error code

Return

0-Succeeded，1-Failed

5.16.2 Set_WiFi_AP_Data

This function is used for setting WIFI AP mode of the controller in non-blocking mode. After the robot receives it, the parameters will be changed. After the buzzer sounds, the change will be applied successfully.

int Set_WiFi_AP_Data(char *wifi_name, char* password);

Argument

①**wifi_name**

Controller wifi name

②**password**

wifi password

Return

0-Succeeded，1-Failed

5.16.3 Set_WiFI_STA_Data

This function is used for setting WIFI STA mode of the controller in non-blocking mode. After the robot receives it, the parameters will be changed. After the buzzer sounds, the change will be applied successfully. Communicate with the controller in WIFI STA mode after reboot the controller.

int Set_WiFI_STA_Data(char *router_name, char* password);

Argument

①**router_name**

Router name

②**password**

Router Wifi pasword

Return

0-Succeeded，1-Failed

5.16.4 Set_USB_Data

This function is used to set the baud rate of the UART_USB interface of the controller in non-blocking mode. After receiving it, robot changes the parameters, and then immediately communicates with the external through the UART-USB interface.

The controller will record the current baud rate after the command is issued, and will still use the same baud rate for external communication after power failure and restart.

int Set_USB_Data(int baudrate);

Argument

①**baudrate**

Baud rate, selectable range: 9600, 38400, 115200 and 460800, if the user sets other values, the controller will proceed using 460800.

Return

0-Succeeded，1-Failed

5.16.5 Set_RS485

This function is used to set the RS485.

After this command is issued, if the Modbus mode is on, it will be automatically shut down, and the controller will record the current baud rate, which will still be used for external communication after power failure and restart.

int Set_RS485(int baudrate);

Argument

①**baudrate**

Baud rate, selectable range: 9600, 38400, 115200 and 460800, if the user sets other values, the controller will proceed using 460800.

Return

0-Succeeded，1-Failed

5.16.6 Set_Ethernet

This function is used for the controller to switch to Ethernet mode in non-blocking mode, and the robot will immediately communicate with the external through the Ethernet interface after receiving it.

int Set_Ethernet();

Return

0-Succeeded， 1-Failed

5.16.7 Set_Arm_Power

This function is used to set the power supply of the robot.

int Set_Arm_Power(bool cmd, bool block);

Argument

① **cmd**

true-power on， false-power off

② **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.16.8 Get_Arm_Power_State

This function is used to obtain the power state information.

int Get_Arm_Power_State(int* power);

Argument

① **power**

0-power on， 1-power off

Return

0-Succeeded， 1-Failed

5.16.9 Get_Arm_Software_Version

This function is used to query robot arm’s software version.

int Get_Arm_Software_Version(string *plan_version,string* ctrl_version);

Argument

①version

Software version number.

②ctrl_version

Kernel version number.

Return

0-Succeeded，1-Failed

5.16.10 Get_System_Runtime

To get controller's cumulative run time.

Get_System_Runtime (string *state,int *day, int *hour, int *min, int *sec);

Argument

① **state**

Error prompts. The system is normal if nothing returned.

② **day**

Day.

③ **hour**

Hour.

④ **min**

Minute.

⑤ **sec**

Second.

Return

0-Succeeded，1-Failed

5.16.11 Clear_System_Runtime

This function is used to clean up controller's cumulative run time.

int Clear_System_Runtime(bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.16.12 Get_Joint_Odom

This function is used to read the cumulative rotation angle of the joint.

int Get_Joint_Odom(string state,float* odom);

Argument

① **state**

Error prompts. The system is normal if nothing returned.

② **odom**

The cumulative rotation angle of each joint.

Return

0-Succeeded， 1-Failed

5.16.13Clear_Joint_Odom

This function is used to clean up the cumulative rotation angle of the joint.

int Clear_Joint_Odom(bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.16.14 Set_High_Speed_Eth

To configure the high speed Ethernet.

int Set_High_Speed_Eth (byte num, bool block);

Argument

① **num**

0-Close；1-Open

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.17 IO configuration

RM-65 robot arm controller IO configurations are shown as below.

Digital output：DO	4 channels，configurable 0~12V
Digital input：DI	3 channels，configurable 0~12V
Analog output：AO	4 channels，output voltage 0~10V
Analog input：AI	4 channels，input voltage 0~10V

5.17.1 Set_IO_State

This function is used to configure the specified IO output state.

int Set_IO_State(int IO,byte num, byte *state);

Argument

①**IO**

0-specifies to set digital IO,1-specifies to set analog IO.

②**num**

The specified IO output channel, range: 1~4

③**state**

true-output high voltage，false-output low voltage

④ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.17.2 Get_IO_State

This function is used to query the specified IO state.

int Get_IO_State(int IO,byte num, byte *state);

Argument

①**IO**

The specified IO type. 0-the state set to digital output ;1-the state set to digital input ;2-the state set to analog output ;3-the state set to analog input.

②**num**

The specified digital output channel, range: 1~4

③**state**

The specified digital output channel state，0x01-high，0x00-low

Return

0-Succeeded，1-Failed

5.17.3 Get_IO_Input

This function is used to query all the IO input state.

int Get_IO_Input(byte *DI_state, float *AI_voltage);

Argument

①**DI_state**

The digital input array address，0x01-input high，0x00-input low

②**AI_voltage**

The analog input voltage array address，range：0~10v

Return

0-Succeeded，1-Failed

5.17.4 Get_IO_Output

This function is used to query all the IO output state.

int Get_IO_Output(byte *DO_state, float *AO_voltage);

Argument

①DO_state

The digital output array address，0x01-input high，0x00-input low

②AO_voltage

The analog output voltage array address, range：0~10v

Return

0-Succeeded，1-Failed

5.18 Robot arm-end IO configuration

The quantity and types of the IO interface at the tool end are listed as follows.

Power output	1 channel, configurable 0V/5V/12V/24V
Digital output：DO	2 channels，the reference level corresponds to the power output
Digital input：DI	2 channels，the reference level corresponds to the power output
Analog output：AO	1 channel，output voltage 0~10V
Analog input：AI	1 channel，input voltage 0~10V
Communication interface	1 channel, configurable RS485/RS232/CAN

5.18.1 Set_Tool_DO_State

This function is used to configure the tool-end specified digital output state.

int Set_Tool_DO_State(byte num, bool state, bool block);

Argument

①num

Specified digital output channel, range: 1~2

②state

true-output high voltage，false-output low voltage

③ block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.18.2 Set_Tool_IO_Mode

This function is used to set IO mode.

int Set_Tool_IO_Mode(byte num, bool state,bool block);

Argument

①**num**

To specify the digital channel, range 1~2

②**state**

To specify the current status/state of the digital IO channel, 0x01-output，0x00-input.

③**block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.18.3 Get_Tool_IO_State

This function is used to query tool end digital IO status/state.

int Get_Tool_IO_State(float* IO_Mode, float *IO_state);

Argument

①**IO_Mode**

To specify digital IO channel mode (range 1~2), 0-input mode, 1-output mode

②**IO_state**

To specify the current input state of the digital IO channel (range 1~2), 0x01-High, 0x00-Low

Return

0-Succeeded，1-Failed

5.18.4 Set_Tool_Voltage

This function is used to configure the tool-end power output.

int Set_Tool_Voltage(byte type, bool block);

Argument

① type

Power output options：0-0V，1-5V，2-12V，3-24V

② block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.18.5 Get_Tool_Voltage

This function is used to get the tool-end power output.

int Get_Tool_Voltage(byte *voltage);

Argument

① voltage

Queried output voltage：0-0V，1-5V，2-12V，3-24V

Return

0-Succeeded，1-Failed

5.19 Robot arm-end gripper control (optional)

RealMan RM-65 robot arm end can be equipped with EG2-4B1 gripper by Inspire Robots Company. To facilitate the user to operate the gripper, the robot arm controller opens the API function of the gripper to the user.

5.19.1 Set_Gripper_Route

This function is used to configure the opening of the gripper.

int Set_Gripper_Route(int min_limit, int max_limit, bool block);

Argument

① min

The minimum opening of the gripper，range：0~1000，no unit

②max

The maximum opening of the gripper，range：0~1000，no unit

③ block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.19.2 Set_Gripper_Release

This function is used to open the gripper at the specified speed to the maximum opening.

int Set_Gripper_Release (int speed, bool block);

Argument

①speed

The opening speed of the gripper，range: 1~1000，no unit

② block

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.19.3 Set_Gripper_Pick

This function is used to control the gripper to grab at the specified speed. When the torque of the gripper is greater than the specified torque threshold, the gripper stops moving.

int Set_Gripper_Pick(int speed, int force, bool block);

Argument

①speed

The grasping speed of the gripper，range: 1~1000，no unit

②force

The torque threshold of the gripper， range: 50~1000， no unit

③block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.19.4 Set_Gripper_Pick_On

This function is used to control the gripper to keep clamping at the specified speed. When the torque of the gripper is greater than the specified torque threshold, the gripper stops moving. After that, when the torque of the gripper is less than the specified torque, the gripper will continue to clip until the torque of the gripper is greater than the specified torque threshold again, and the movement will stop.

int Set_Gripper_Pick_On(int speed, int force, bool block);

Argument

①speed

The grasping speed of the gripper， range: 1~1000， no unit

②force

The torque threshold of the gripper， range: 50~1000， no unit

③ block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.19.5 Set_Gripper_Position

This function is used to control the hand to reach the specified opening position.

int Set_Gripper_Position (int position, bool block);

Argument

①position

The specified opening position of the gripper, range: 1~1000, no unit

② block

0-Non-blocking, returning immediately after sent; 1-Blocking, waiting for the controller to return a successful setup instruction.

Return

0- Succeeded, 1-Failed

5.20 Drag teaching and trajectory reproduction

RealMan RM-65 robot adopts joint current loop to realize the drag teaching, and the configuration functions of the drag teaching and trajectory reproduction are shown as follows.

5.20.1 Start_Drag_Teach

This function is used to set the robot to the drag teaching mode.

int Start_Drag_Teach (bool block);

Argument

① block

0-Non-blocking, returning immediately after sent; 1-Blocking, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded, 1-Failed

5.20.2 Stop_Drag_Teach

This function is used to stop the drag teaching mode.

int Stop_Drag_Teach (bool block);

Argument

① block

0-Non-blocking, returning immediately after sent; 1-Blocking, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.3 Run_Drag_Trajectory

This function is used to run the trajectory repetition. It can only be used after the drag teaching is finished, and the robot arm should be located at the starting point of the drag teaching. If the current position is not at the starting point of trajectory repetition, please call 5.20.7 first, otherwise an error message will be returned.

int Run_Drag_Trajectory (bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.4 Pause_Drag_Trajectory

This function is used to pause the trajectory repetition.

int Pause_Drag_Trajectory (bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.5 Continue_Drag_Trajectory

This function is used to continue the paused trajectory repetition process. When the trajectory continues, the robot arm must be at the position when it stopped. Otherwise, an error will be reported, and the user can only reproduce the trajectory from the beginning position

int Continue_Drag_Trajectory (bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.6 Stop_Drag_Trajectory

This function is used to stop the trajectory repetition process.

int Stop_Drag_Trajectory (bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.7 Drag_Trajectory_Origin

Before the trajectory is reproduced, the robot arm must be at the starting point of the trajectory. If set correctly, the robot arm will move to the starting point of the trajectory at a speed of 20%.

int Drag_Trajectory_Origin (bool block);

Argument

① **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.20.8 Start_Multi_Drag_Teach

To start the composite drag teaching mode.

int Start_Multi_Drag_Teach(int mode,bool block);

Argument

① **mode**

The drag teaching mode.

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction .

Return

0-Succeeded，1-Failed

5.20.9 Set_Force_Postion

To set the force position mixing control.

int Set_Force_Postion (int mode,int force,bool block);

Argument

① **mode**

1-Base coordinate system Z-axis force control; 2-operation surface normal force control; 3-operation surface radial force control.

② **force**

Force control level, the higher the level, the smaller the force.

③ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.20.10 Stop_Force_Postion

To stop the force position mixing control.

int Stop_Force_Postion (bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction．

Return

0-Succeeded，1-Failed

5.21 The use of six-axis force/torque sensor (optional)

RealMan RM-65F robot arm end is equipped with an integrated six-axis force sensor, which requires no external wiring. Users can operate the six-axis force directly through the protocol and obtain the six-axis force data.

5.21.1 Get_Force_Data

Query the force and torque information obtained by the current six-axis force sensor. If the force data is to be obtained periodically, the period should not be less than 50ms.

int Get_Force_Data(float *Force);

Argument

① **Force**

Returns the address of an array of forces and torques. The array has six elements, Fx, Fy, Fz, Mx, My, Mz. Force unit is N; Torque unit is Nm.

Return

0-Succeeded，1-Failed

5.21.2 Clear_Force_Data

The six-axis force data is cleared to zero, that is, all subsequent data obtained are based on the offset of the current data.

int Clear_Force_Data(bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the

controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.21.3 Set_Force_Sensor

Set the 6-dimensional force barycenter parameter. After the 6- dimensional force is reinstalled, the initial force and barycenter received by the 6- dimensional force must be recalculated. Under different postures, the data of the six-axis forces are obtained and used to calculate the position of the center of gravity. After the command is issued, the robot moves to each calibration point at a speed of 20%. The process cannot be interrupted. After the interruption, it must be recalibrated.

Important note: Calibration of the robot must be guaranteed at stationary state.

int Set_Force_Sensor (bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.21.4 Manual_Set_Force

Manual calibration of six-axis force data. It needs four waypoints in total, requiring four function calls.

int Manual_Set_Force (int type,float *joint);

Argument

① **type**

Waypoints sent 1~4 in sequence using this function.

② **joint**

Joint angle

Return

0-Succeeded，1-Failed

5.21.5 Stop_Set_Force_Sensor

In the process of calibrating the six-axis or one-axis force, if there is an accident, the command will be sent to stop the movement of the robot and exit the calibration process.

int Stop_Set_Force_Sensor (bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.22 Control of the dexterous (five fingers) hand (optional)

RealMan RM-65 robot arm end is equipped with a five-fingered dexterous hand, which can be controlled by protocol.

5.22.1 Set_Hand_Posture

Set the gesture number of the dexterous hand. After successful setting, the dexterous book will move according to the gestures pre-saved in Flash.

int Set_Hand_Posture (int posture_num, bool block);

Argument

① **posture_num**

Pre-saved gesture sequence number in the dexterous，range：1~40

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.22.2 Set_Hand_Seq

Set the sequence number of dexterous hand movements. After successful setting, the dexterous book moves according to the pre-saved action sequence in Flash.

```
int Set_Hand_Seq (int seq_num, bool block);
```

Argument

① seq_num

Pre-saved action sequence number in the dexterous， range： 1~40

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.22.3 Set_Hand_Angle

Set the angle of dexterous hand. The dexterous hand has 6 degrees of freedom. They are the little finger, ring finger, middle finger, index finger, thumb bending, thumb rotation from 1 to 6, respectively.

```
int Set_Hand_Angle (int *angle, bool block);
```

Argument

① angle

Finger angle array with 6 elements representing angles of 6 degrees of freedom. Range: 0~1000. In addition, -1 means that the degree of freedom does not perform any operation and remains in the current state.

② block

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.22.4 Set_Hand_Speed

Set the speed of each joint of dexterous hand.

int Set_Hand_Speed (int speed, bool block);

Argument

① **speed**

Speed setting for each joint of dexterous hand, range：1~1000

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.22.5 Set_Hand_Force

Set the force threshold of each joint of dexterous hand.

int Set_Hand_Force (int force, bool block);

Argument

① **force**

Setting of force threshold of each joint of dexterous hand, range: 1~1000, representing torque threshold of each joint (four fingers grip force 0~10N, thumb grip force 0~15N)

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.23 Arm-End PWM (optional)

The digital output channel 2 of RM robot arm end interface board can be multiplexed as PWM output. Its output high level is consistent with the current end output voltage.

5.23.1 Set_PWM

This function is used to set arm-end PWM output.

int Set_PWM (int Frq, int Dpulse, bool block);

Argument

①Frq

PWM output frequency, range: 100~10000Hz

②Dpulse

PWM output duty ratio, range: 0~100, the accuracy does not exceed 1%

③ block

0-Non-blocking, returned immediately after sending; 1-Block, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.23.2 Stop_PWM

This function is used to stop the arm-end PWM output.

int Stop_PWM (bool block);

Argument

①block

0-Non-blocking, returned immediately after sending; 1-Block, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.24 Arm-End Sensor: One-Axis Force (optional)

The interface plate at RM robot arm end is integrated with a one-axis force sensor, which can obtain the force in the Z direction. The measuring range is 200N, and the accuracy is 0.5%FS.

5.24.1 Get_Fz

This function is used to query the arm-end one-axis force data.

int Get_Fz (float *data);

Argument

①**data**

Returned one-axis force data

Return

0-Succeeded，1-Failed

Note

Note: After the first command is issued, the one-axis force data is updated, and the returned first data frame at this time has a lag. Please start with the valid data from the second data frame. If FZ data is queried periodically, the frequency cannot be higher than 40Hz.

5.24.2 Clear_Fz

This function is used to clear the arm-end one-axis force data. After the one-axis force data is cleared, all subsequent data obtained are based on the current bias.

int Clear_Fz (bool block);

Argument

①**block**

0-Non-blocking, returned immediately after sending; 1-Block, waiting for the

controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.24.3 Auto_Set_Fz

This function is used to automatically calibrate robot end one-axis force data.

int Auto_Set_Fz (bool block);

Argument

① block

0-Non-blocking, returned immediately after sending; 1-Block, waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.24.4 Manual_Set_Fz

This function is used to manually calibrate robot end one-axis force data.

int Manual_Set_Fz (float* joint,float* joint2);

Argument

① joint

Joint angle at point 1

② joint2

Joint angle at point 2

Return

0-Succeeded， 1-Failed

5.25 Modbus RTU Configuration

RealMan robot arm has one RS485 communication each at the controller's 26-core and end interface board 9-core aviation plugs, both of which can be configured in standard Modbus RTU mode via the JSON protocol. The peripherals of the port connection are then read and written through the JOSN protocol.

5.25.1 Set_Modbus_Mode

To set the communication in Modbus RTU mode.

Set_Modbus_Mode (int port,int baudrate,int timeout,bool block);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② baudrate

Baud rate， supporting 9600,115200,and 460800 baud rates.

③ timeout

Timeout in seconds.

④ **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.25.2 Close_Modbus_Mode

To close Modbus RTU mode.

Close_Modbus_Mode (int port , bool block);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② **block**

0-Non-blocking， returning immediately after sent； 1-Blocking， waiting for the controller to return a successful setup instruction.

Return

0-Succeeded， 1-Failed

5.25.3 Get_Read_Coils

To read coil(s).

Get_Read_Coils (int port, int address, int num, int device, int* coils_data);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ num

The number of coils to be read. This command supports reading up to 8 coil data at a time, i.e., the data returned will not be one byte.

④ device

Peripheral device address.

⑤ coils_data

Return discrete data.

Return

0-Succeeded， 1-Failed

5.25.4 Get_Read_Input_Status

To read the input status.

Get_Read_Input_Status (int port, int address, int num, int device, int* coils_data);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ num

The number of coils to be read. This command supports reading up to 8 coil data at a time, i.e., the data returned will not be one byte.

④ device

Peripheral device address.

⑤ coils_data

Return discrete data.

Return

0-Succeeded， 1-Failed

5.25.5 Get_Read_Holding_Registers

To read the holding registers.

Get_Read_Holding_Rsgisters (int port, int address, int device, int* coils_data);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ device

Peripheral device address.

④ coils_data

Return discrete data.

Return

0-Succeeded， 1-Failed

5.25.6 Get_Read_Input_Registers

To read the input registers.

Get_Read_Input_Rsgisters (int port, int address, int device, int* coils_data);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ device

Peripheral device address.

④ coils_data

Return discrete data.

Return

0-Succeeded， 1-Failed

5.25.7 Write_Single_Coil

To write a single coil data.

Write_Single_Coil (int port, int address, int data, int device, bool block);

Argument

① port

Communication port， 0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ data

Data that is written to the coil.

④ device

Peripheral device address.

⑤ block

0-Non-blocking， returning immediately after sent； 1-Blocking.

Return

0-Succeeded， 1-Failed

5.25.8 Write_Single_Register

To write a single register.

Write_Single_Register (int port, int address, int data, int device, bool block);

Argument

① port

Communication port，0-controller RS485, 1-end interface board RS485.

② address

Coil start address.

③ data

Data that is written to the coil.

④ device

Peripheral device address.

⑤ block

0-Non-blocking，returning immediately after sent；1-Blocking.

Return

0-Succeeded，1-Failed

5.26 Mobile Platform and Lift

RM robot arm can integrate the C100 industrial mobile platform by Sensorobotics company and our independently developed lifting mechanisms.

5.26.1 Set_Lift

To set lift platform movement.

Set_Lift (int height,int speed,bool block);

Argument

① **height**

Lifting height.

② **speed**

Lifting speed.

③ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.26.2 Set_Lift_Speed

To set lift platform moving speed.

Set_Lift_Speed (int speed,bool block);

Argument

① **speed**

Percentage of lifting speed.

② **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.26.3 Set_Lift_Height

To closed-loop control of the position of the lif.

Set_Lift_Height (int height,int speed,bool block);

Argument

① **height**

Target height.

② **speed**

Percentage of the lifting speed.

③ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.26.4 Get_Lift_State

The status of the lifting mechanism.

Get_Lift_Height (int *height,int *current,int *err);

Argument

① **height**

Target height. Unit: mm. Accuracy: 1mm. Range：0~2300.

② **current**

Current lift drive current, unit: mA and accuracy：1mA.

③ **err**

Lift drive error code.

Return

0-Succeeded，1-Failed

5.27 Passthrough force-position hybrid control compensation

For RM robot arms with one-axis force sensor or six-axis force sensor, the user can not only call the underlying force-position hybrid control module directly using the teach pendant software, but can also compensate for custom trajectories in the form of periodic passthrough in combination with the underlying force-position hybrid control algorithm.

5.27.1 Start_Force_Position_Move

To start the passthrough force-position hybrid control compensation mode.

Start_Force_Position_Move (bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.27.2 Force_Position_Move_Joint

The force-position hybrid control compensation data.

Force_Position_Move_Joint(const float *joint,byte sensor,byte mode,int dir,float force,bool block);

Argument

① **joint**

Joint angle.

② **sensor**

The type of sensor, 0-one-axis force sensor, 1- six-axis force sensor

③ **mode**

0-based on the Base coordinate system， 1-based on the Tool coordinate system.

④ **dir**

Force control direction, 0~5 represent X/Y/Z/Rx/Ry/Rz respectively, where the default direction is Z direction when one-axis force sensor is used.

⑤ **force**

The force (N)

⑥ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.27.3 Force_Position_Move_POSE

The force-position hybrid control compensation data.

Force_Position_Move_POSE(POSE pose,byte sensor,byte mode,int dir,float force,bool block);

Argument

① **pose**

The position and posture in the current coordinate system.

② **sensor**

The type of sensor, 0-one-axis force sensor, 1- six-axis force sensor

③ **mode**

0-based on the Base coordinate system， 1-based on the Tool coordinate system.

④ **dir**

Force control direction, 0~5 represent X/Y/Z/Rx/Ry/Rz respectively, where the default direction is Z direction when one-axis force sensor is used.

⑤ **force**

The force (N)

⑥ **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.27.4 Stop_Force_Position_Move

To stop the passthrough force-position hybrid control compensation mode.

Stop_Force_Position_Move (bool block);

Argument

① **block**

0-Non-blocking，returning immediately after sent；1-Blocking，waiting for the controller to return a successful setup instruction.

Return

0-Succeeded，1-Failed

5.28 Algorithm Tool Interface

5.28.1 setAngle

Set the installation angle parameters of the algorithm.

setAngle(float x,float y);

Argument

① **x**

X axis installation angle.

② **y**

Y axis installation angle.

5.28.2 setLwt

Set theTerminal DH parameter of the algorithm.

setAngle(float x,float y);

Argument

① **type**

0-Standard, 1-Integrated 1-axis force, 2-Integrated 6-axis force.

5.28.3 Forward_Kinematics

This function is used for forward kinematics of RM robot arm .

Forward_Kinematics(const float * joint);

Argument

① **joint**

0-Standard, 1-Integrated 1-axis force, 2-Integrated 6-axis force.

Return

Forward kinematics results.

5.28.4 Inverse_Kinematics

This function is used for inverse kinematics of RM robot arm .

inverse_kinematics(const float *q_in, const KINEMATIC *q_pose, float *q_out,const uint8_t flag);

Argument

① **q_in**

Joint angle at last moment.

② **q_pose**

Target pose and position.

③ **q_out**

Output joint angle

④ **flag**

Pose parameter type :0-quaternion;1-eulerian angle

Return

1：Normal operation; -9:Unreachable；-3:Joint 1 overrun position ~~ -8：Joint 6 overrun position；