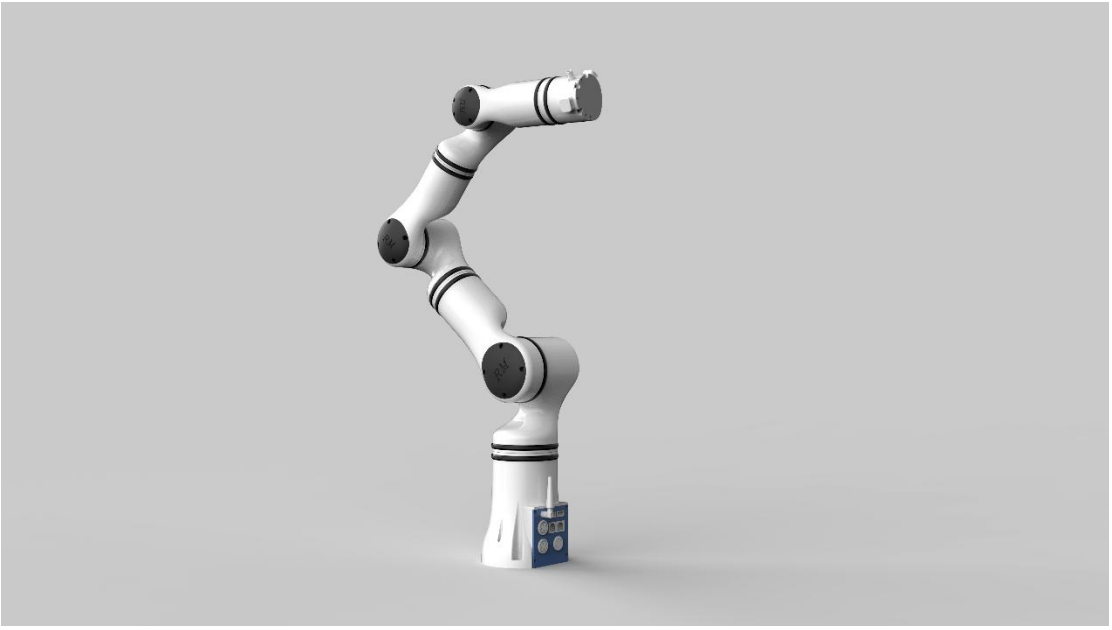
睿尔曼机械臂接口函数说明（C）V4.3.4



睿尔曼智能科技（北京）有限公司

文件修订记录：

版本号	时间	备注
V1.0	2020-05-01	拟制
V1.1	2020-05-10	修订
V1.2	2020-05-15	修订（通用化修订）
V1.3	2020-05-17	格式及说明整理
V1.4	2020-05-25	格式整理
V1.5	2020-06-05	修改 WIFI 配置流程
V1.6	2020-06-30	加入 IO 部分
V1.7	2020-07-03	修改部分协议名称
V1.8	2020-08-04	加入末端接口板部分协议
V1.9	2021-03-12	加入路径点控制等相关函数
V2.0	2021-05-20	加入 Movej_P；末端 PWM 设置；末端一维力设置
V2.1	2021-09-17	增加 Modbus 协议内容
V2.2	2021-09-27	增加防碰撞等级设置；移动平台及升降机构内容
V2.3	2021-10-19	勘误
V2.4	2022-01-28	更新六维力一维力函数代码
V2.5	2022-04-12	勘误
V2.6	2022-04-25	新增位姿透传等
V3.0	2022-05-13	合并 RM65 与 RM75 接口；新增正逆解接口；优化部分接口
V3.1	2022-06-09	勘误
V3.2	2022-07-12	Modbus 协议部分增加多圈和多寄存器操作
V4.0	2022-09-23	重构 API 架构
V4.0.2	2022-10-31	增加算法工具接口

V4.0.4	2022-11-31	修改初始化 API 设置型号，优化部分接口
V4.0.6	2023-02-08	优化夹爪相关接口阻塞模式，优化获取机械臂全部状态接口
V4.1.0	2023-03-20	配合控制器优化接口通讯效率；完善头文件注释；优化部分接口
V4.1.3	2023-03-22	优化透传、夹爪、获取力数据等接口
V4.2.0	2023-06-19	格式及说明修改 UART_USB 及 RS_485 接口新增波特率 19200 新增保存所有参数、清除系统错误、配置高速网口网络 IP、读多圈及读多个寄存器等接口 修正算法方面接口（改为引用算法库，同步修改相关引用和实现）
V4.2.2	2023-08-21	新增适配第三代控制器接口 新增 UDP 机械臂状态主动上报接口 新增扩展关节、保存拖动示教轨迹接口
V4.2.5	2023-10-19	UDP 接口新增目标 IP 地址、力数据坐标系的设置和查询 获取无线网络信息返回结构体新增 wifi 名称、密码、信道 获取力数据新增当前工作和工具坐标系下受力 新增在线编程轨迹列表等相关接口 C++部分接口适配 MATLAB API 重构
V4.2.6	2023-11-27	修复 movel、 movec 等运动接口 Linux 环境下报错机械臂异常停止 优化连接、逆解、保存拖动示教轨迹等接口
V4.2.8	2023-12-27	新增电子围栏相关接口 增加 modbus-TCP 主站和 modbus-RTU 从站接口 新增查询示教参考坐标系接口 新增自碰撞安全检测相关接口 新增读取软件信息接口 新增关节驱动器转速、加速度以及最大最

		小限位设置查询接口 优化开始复合模式拖动示教、movej、movel、movec、movej_p、一键设置关节限位等接口 新增查询夹爪信息接口 新增设置、查询机械臂仿真/真实模式
V4.2.9	2024-1-30	文档勘误
V4.3.0	2024-3-30	适配 GEN72、ECO63 机械臂 运动接口优化，适配控制器 1.5.0 软件版本 修复灵巧手相关接口阻塞模式不生效问题 修复夹爪夹取松开速度低时执行成功返回 7 的问题，新增超时时间设置 修改管理电子围栏等接口名称为管理几何参数模型，单位由 mm 改为 m 修改 SensorType 枚举项名称 B、ZF、SF 为_B、_ZF、_SF 优化电子围栏与虚拟墙数据管理接口 新增工作坐标系和工具坐标系的更新接口 新增工具坐标系包络参数相关接口 新增虚拟墙相关接口 新增在线编程文件信息修改接口 新增 IO 控制默认运行在线编程文件接口 新增全局点位相关接口
V4.3.1	2024-5-30	新增样条曲线运动 Moves 新增读多个输入寄存器接口 Read_Multiple_Input_Registers 新增使用结构体传递参数的逆解接口 Send_TrajectoryFile 发送在线编程文件 新增保存到控制器 id 参数
V4.3.2	2024-7-3	适配 GEN72 型号机械臂
V4.3.3	2024-8-5	修复使用控制器 1.6.0 时无法使用 UDP 的问题 修改运动接口交融半径

V4.3.4	2024-8-19	修复交融半径调用失败 新增计算平移、旋转运动位姿算法接口 添加交融半径使用注意事项 适配控制器 1.6.0 主动上报接口自定义项配置、获取及上报信息获取

目录

1. 简介

为了方便用户通过上位机自主开发程序控制机械臂，睿尔曼提供了基于 TCP/IP socket 接口函数，用户可通过 WIFI（AP 模式或者 STA 模式）、以太网口与机械臂建立通信，并控制机械臂。

2. 功能介绍

接口函数分为 Windows 版本和 Linux 版本，可直接加载到 C、C++、C#或者 Python 工程中使用，接口函数将用户指令封装成标准的 JSON 格式下发给机械臂，并解析机械臂回传的数据提供给用户。

接口函数基于 TCP/IP 协议编写，其中：

机械臂默认 IP 地址：192.168.1.18，端口号：8080

无论是 WIFI 模式还是以太网口模式，机械臂均以该 IP 和端口号对外进行 socket 通信，机械臂为 Server 模式，用户为 Client 模式。

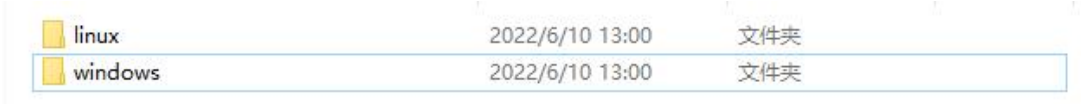
3. 使用说明

接口函数包内包括两个文件夹：（两个常用版本使用说明）

（1）linux：Linux 操作系统接口函数；

（2）windows：Windows 操作系统接口函数。

两部分除了调用系统 Socket 库稍有区别之外，其他部分完全相同。



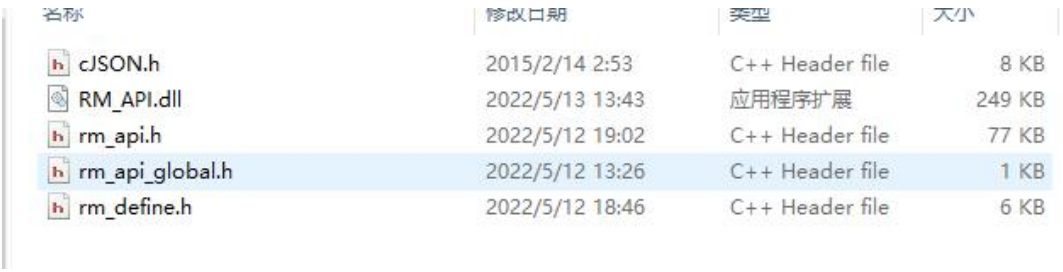
API 组成如下图所示：

（1）cJSON.h：cJSON 库的头文件，定义了 cJSON 库的结构体、数据类型和函数。

（2）rm_define.h：机械臂自定义头文件，包含了定义的数据类型、结构体和错误代码。

（3）rm_api.h ； rm_api_global.h：接口函数定义。

（4）rm_API.dll：接口函数实现。

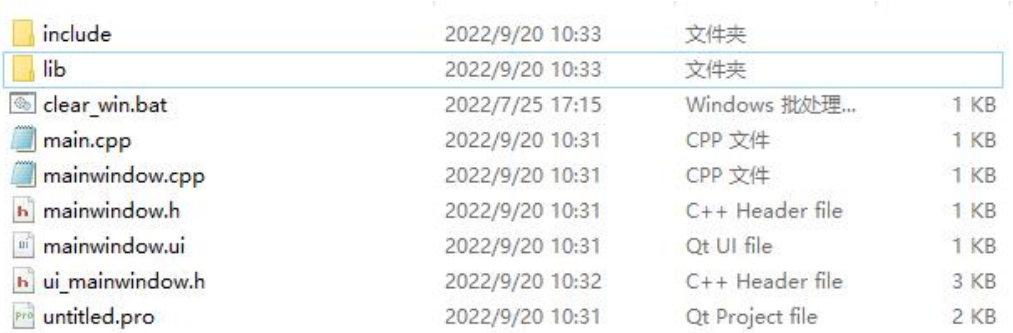


名称	修改日期	类型	大小
cJSON.h	2015/2/14 2:53	C++ Header file	8 KB
RM_API.dll	2022/5/13 13:43	应用程序扩展	249 KB
rm_api.h	2022/5/12 19:02	C++ Header file	77 KB
rm_api_global.h	2022/5/12 13:26	C++ Header file	1 KB
rm_define.h	2022/5/12 18:46	C++ Header file	6 KB

API 的头文件中已做了处理，可直接加载到工程中使用。

以下是在 QT 开发环境使用步骤：

步骤一：将我们所提供的 include 以及 lib 文件夹拷贝到工程目录下。



include	2022/9/20 10:33	文件夹	
lib	2022/9/20 10:33	文件夹	
clear_win.bat	2022/7/25 17:15	Windows 批处理...	1 KB
main.cpp	2022/9/20 10:31	CPP 文件	1 KB
mainwindow.cpp	2022/9/20 10:31	CPP 文件	1 KB
mainwindow.h	2022/9/20 10:31	C++ Header file	1 KB
mainwindow.ui	2022/9/20 10:31	Qt UI file	1 KB
ui_mainwindow.h	2022/9/20 10:32	C++ Header file	3 KB
untitled.pro	2022/9/20 10:31	Qt Project file	2 KB

步骤二：在 pro 文件中写入头文件以及 dll 文件路径。

```
INCLUDEPATH += $$PWD/include
LIBS += -L$$PWD/lib -lRM_Base
```

步骤三：添加完成后在文件中引入相应头文件即可使用。

```
#include "rm_base.h"
```

代码使用示例：

```
// 初始化API，注册回调函数
RM_API_Init(65, MCallback);

// 连接服务器
m_sockhand =  Arm_Socket_Start((char*)"192.168.1.18", 8080, 5000);
qDebug() << "m_sockhand:" << m_sockhand;
```

```
{
    // 初始化API，注册回调函数
    RM_API_Init(65, MCallback);

    // 连接服务器
    m_sockhand =  Arm_Socket_Start((char*)"192.168.1.18", 8080, 5000);
    qDebug() << "m_sockhand:" << m_sockhand;

    //获取API版本号
    char* version;
    version = API_Version();
    qDebug() << "version:" << version;

    //关节空间运动
    float joint[6] = {0,20,70,0,90,0};
    int ret;
    ret = Movej_Cmd(m_sockhand,joint,20,0,1);
    qDebug() << "ret:" << ret;

    // 关闭连接
    Arm_Socket_Close(m_sockhand);
    m_sockhand = -1;
}
```

```
m_sockhand: 1008
version: 4.1.6
ret: 0
```

使用流程如下所示：

（1）通过 WIFI 或者以太网口与机械臂连接，保证上位机与机械臂控制器在同一网段内；

（2）调用 RM_API_Init()函数初始化 API。设置接收透传接口回调函数， 不需要可以传入 NULL。

（3）调用 Arm_Socket_Start()函数，与机械臂进行 socket 连接。注意，经测试在 Windows 操作系统下，可能需要 2～3 次才能建立连接，因此根据该函数的返回值判断是否需要再次调用来保证 socket 连接成功，成功：返回句柄，

失败： < 0。

（3）连接成功后，用户根据需要调用接口函数。

（4）使用结束后，调用 Arm_Socket_Close()函数关闭 socket 连接，释放系统资源。

4. 数据类型说明

4.1. 控制器错误类型

序号	错误代码（16 进制）	错误内容
1	0x0000	系统正常
2	0x1001	关节通信异常
3	0x1002	目标角度超过限位
4	0x1003	该处不可达，为奇异点
5	0x1004	实时内核通信错误
6	0x1005	关节通信总线错误
7	0x1006	规划层内核错误
8	0x1007	关节超速
9	0x1008	末端接口板无法连接
10	0x1009	超速度限制
11	0x100A	超加速度限制
12	0x100B	关节抱闸未打开
13	0x100C	拖动示教时超速
14	0x100D	机械臂发生碰撞

15	0x100E	无该工作坐标系
16	0x100F	无该工具坐标系
17	0x1010	关节发生掉使能错误

4.2. 关节错误类型

序号	错误代码（16 进制）	错误内容
1	0x0000	关节正常
2	0x0001	FOC 错误
3	0x0002	过压
4	0x0004	欠压
5	0x0008	过温
6	0x0010	启动失败
7	0x0020	编码器错误
8	0x0040	过流
9	0x0080	软件错误
10	0x0100	温度传感器错误
11	0x0200	位置超限错误
12	0x0400	关节 ID 非法
13	0x0800	位置跟踪错误
14	0x1000	电流检测错误
15	0x2000	抱闸打开失败
16	0x4000	位置指令阶跃警告
17	0x8000	多圈关节丢圈数

18	0xF000	通信丢帧

4.3. API 错误类型

序号	错误代码（16 进制）	错误内容
1	0x0000	系统运行正常
2	0x0001	消息请求返回 FALSE
3	0x0002	机械臂未初始化或输入型号非法
4	0x0003	非法超时时间
5	0x0004	Socket 初始化失败
6	0x0005	Socket 连接失败
7	0x0006	Socket 发送失败
8	0x0007	Socket 通讯超时
9	0x0008	未知错误
10	0x0009	数据不完整
11	0x000A	数组长度错误
12	0x000B	数据类型错误
13	0x000C	型号错误
14	0x000D	缺少回调函数
15	0x000E	机械臂异常停止
16	0x000F	轨迹文件名称过长
17	0x0010	轨迹文件校验失败
18	0x0011	轨迹文件读取失败

19	0x0012	控制器忙，请稍后再试
20	0x0013	非法输入
21	0x0014	数据队列已满
22	0x0015	计算失败
23	0x0016	文件打开失败
24	0x0017	力控标定手动停止
25	0x0018	没有可保存轨迹
26	0x0019	UDP 监听接口运行报错

4.4. 数据结构定义

数据结构定义列举了一些常用结构体，其他结构体和数据类型请查看 rm_define.h 和 robot_define.h

4.4.1. 位姿结构体 Pose

```
typedef struct
{
    Pos position;
    Quat quaternion;
    Euler euler;
} Pose;
```

结构体成员

position

位置坐标，float 类型，单位：m

quaternion

四元数

euler

姿态角，float 类型，单位：rad

4.4.2. 坐标系结构体 FRAME

```
typedef struct
{
    FRAME_NAME frame_name;    //坐标系名称
    Pose pose;                //坐标系位姿
    float payload;            //坐标系末端负载重量
    float x;                  //坐标系末端负载位置
    float y;                  //坐标系末端负载位置
    float z;                  //坐标系末端负载位置
}FRAME;
```

结构体成员

frame_name

坐标系名称

Pose

坐标系位姿

payload

坐标系末端负载重量　单位 g

x,y,z

坐标系末端负载位置，单位：m

4.4.3. 关节状态结构体 JOINT_STATE

```
typedef struct
{
    float joint[ARM_DOF];              //关节角度
    float temperature[ARM_DOF];     //关节温度
    float voltage[ARM_DOF];          //关节电压
    float current[ARM_DOF];          //关节电流
    byte en_state[ARM_DOF];          //使能状态
    uint16_t err_flag[ARM_DOF];     //关节错误代码
    uint16_t sys_err;              //机械臂系统错误代码
}JOINT_STATE;
```

结构体成员

joint

关节角度，每个关节角度，float 类型，单位：°

temperature

关节温度，float 类型，单位：摄氏度

voltage

关节电压，float 类型，单位：V

current

关节电流，float 类型，单位：mA

en_state

使能状态

err_flag

关节错误代码，unsigned int 类型

sys_err

机械臂系统错误代码，unsigned int 类型

4.4.4. 机械臂控制模式枚举 ARM_CTRL_MODES

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

枚举成员

None_Mode

无规划模式

Joint_Mode

关节空间规划模式

Line_Mode

笛卡尔空间直线规划模式

Circle_Mode

笛卡尔空间圆弧规划模式

4.4.5. 机械臂位置示教模式 POS_TEACH_MODES

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

枚举成员

X_Dir

X 轴方向，默认 0

Y_Dir

Y 轴方向，默认 1

Z_Dir

Z 轴方向，默认 2

4.4.6. 机械臂姿态示教模式 ORT_TEACH_MODES

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

枚举成员

RX_Rotate

RX 轴方向，默认 0

RY_Rotate

RY 轴方向，默认 1

RZ_Rotate

RZ 轴方向，默认 2

4.4.7. 控制器通讯方式选择 ARM_COMM_TYPE

控制器通讯方式选择

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

枚举成员

WIFI_AP

WIFI AP 模式

WIFI_STA

WIFI STA 模式

BlueTeeth

蓝牙模式

USB

通过控制器 UART-USB 接口通信

Ethernet

以太网口

5. 接口库函数说明

5.1. 连接相关函数

该部分函数用来控制 socket 连接的打开与关闭。

5.1.1. API 初始化 RM_API_Init

该函数用于初始化 API。

```
int RM_API_Init(int devMode, RM_Callback pCallback);
```

参数：

（1）dev_mode

目标设备型号 ARM_65/ ARM_75/ARM_63_1/ ARM_63_2/ ARM_ECO65/ ARM_GEN72。若传入型号非法，则默认机械臂为六轴。

（2）pCallback

用于接收透传接口回调函数， 不需要可以传入 NULL。

5.1.2. API 反初始化 RM_API_UnInit

该函数用于 API 反初始化。

```
int RM_API_UnInit();
```

5.1.3. 查询 API 版本信息 API_Version

该函数用于查询 API 当前版本。

```
char * API_Version();
```

返回值：

API 版本号。

5.1.4. 连接机械臂 Arm_Socket_Start

该函数用于连接机械臂。

SOCKHANDLE Arm_Socket_Start(char* arm_ip, int arm_port, int recv_timeout);

参数：

（1）arm_ip

用于传入要连接机械臂的 IP 地址，机械臂 IP 地址默认为 192.168.1.18。

（2）arm_port

用于传入要连接机械臂的端口，机械臂端口默认为 8080。

（3）recv_timeout

Socket 连接超时时间，单位：ms

返回值：

成功返回：Socket 句柄；失败返回：错误码，rm_define.h 查询。

5.1.5. 查询连接状态 Arm_Socket_State

用于查询连接状态。

int Arm_Socket_State(SOCKHANDLE ArmSocket);

（1）ArmSocket

Socket 句柄。

返回值：

正常返回：0；失败返回：错误码， rm_define.h 查询。

5.1.6. 关闭连接 Arm_Socket_Close

该函数用于关闭与机械臂的 socket 连接。

void Arm_Socket_Close(SOCKHANDLE ArmSocket);

（1）ArmSocket

Socket 句柄。

5.2. 关节配置函数

睿尔曼机械臂在出厂前所有参数都已经配置到最佳状态，一般不建议用户修改关节的底层参数。若用户确需修改，首先应使机械臂处于非使能状态，然后再发送修改参数指令，参数设置成功后，发送关节恢复使能指令。需要注意的是，关节恢复使能时，用户需要保证关节处于静止状态，以免上使能过程中关节发生报错。关节正常上使能后，用户方可控制关节运动。

5.2.1. 设置关节最大速度 Set_Joint_Speed

该函数用于设置关节最大速度，单位：°/s。建议使用默认最大速度，如需更改，设置的关节最大加速度与最大速度的比值需要≥1.5，否则可能出现运动异常。

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

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）speed

关节转速，单位：°/s

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待

控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.2. 设置关节最大加速度 Set_Joint_Acc

该函数用于设置关节最大加速度，单位：°/s²。建议使用默认关节最大加速度，如需更改，设置的关节最大加速度与最大速度的比值需要≥1.5，否则可能出现运动异常

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

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）acc

关节转速，单位：°/s²

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.3. 设置关节最小限位 Set_Joint_Min_Pos

该函数用于设置关节所能到达的最小限位，单位：°

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

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）joint

关节最小限位，单位：°

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.4. 设置关节最大限位 Set_Joint_Max_Pos

该函数用于设置关节所能达到的最大限位，单位：°

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

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）joint

关节最大限位，单位：°

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.5. 设置关节最大速度（驱动器)Set_Joint_Drive_Speed

该函数用于设置关节最大速度，单位：°/s。建议使用默认最大速度，如需更改，设置的关节最大加速度与最大速度的比值需要≥1.5，否则可能出现运动异常。

```
int Set_Joint_Drive_Speed(SOCKHANDLE ArmSocket, byte joint_num, float speed);
```

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）speed

关节转速，单位：°/s

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.6. 设置关节最大加速度（驱动器)Set_Joint_Drive_Acc

该函数用于设置关节最大加速度，单位：°/s²。建议使用默认关节最大加速度，如需更改，设置的关节最大加速度与最大速度的比值需要≥1.5，否则可能出现运动异常。

int Set_Joint_Drive_Acc(SOCKHANDLE ArmSocket, byte joint_num, float acc);

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）acc

关节转速，单位：°/s²

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.7. 设置关节最小限位（驱动器)Set_Joint_Drive_Min_Pos

该函数用于设置关节所能到达的最小限位，单位：°

int Set_Joint_Drive_Min_Pos(SOCKHANDLE ArmSocket, byte joint_num, float joint);

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）joint

关节最小限位，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.8. 设置关节最大限位（驱动器)Set_Joint_Drive_Max_Pos

该函数用于设置关节所能达到的最大限位，单位：°

int Set_Joint_Drive_Max_Pos(SOCKHANDLE ArmSocket, byte joint_num, float joint);

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）joint

关节最大限位，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.9. 设置关节使能 Set_Joint_EN_State

该函数用于设置关节使能状态。

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

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）state

true-上使能，false-掉使能

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.10. 设置关节零位 Set_Joint_Zero_Pos

该函数用于将当前位置设置为关节零位。

```
int Set_Joint_Zero_Pos(SOCKHANDLE ArmSocket, byte joint_num, bool block);
```

参数

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.11. 清除关节错误代码 Set_Joint_Err_Clear

该函数用于清除关节错误代码。

int Set_Joint_Err_Clear(SOCKHANDLE ArmSocket, byte joint_num, bool block);

参数：

（1）ArmSocket

Socket 句柄。

（2）joint_num

关节序号，1~7

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.2.12. 自动设置限位 Auto_Set_Joint_Limit

该函数用于自动设置限位。

int Auto_Set_Joint_Limit(SOCKHANDLE ArmSocket, byte limit_mode);

参数：

（1）ArmSocket

Socket 句柄。

（2）limit_mode

设置类型，1－正式模式，各关节限位为规格参数中的软限位和硬限位

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3. 关节参数查询函数

5.3.1. 查询关节最大速度 Get_Joint_Speed

该函数用于查询关节最大速度。

int Get_Joint_Speed(SOCKHANDLE ArmSocket, float *speed);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

存放关节 1~7 转速数组，单位：°/s

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.2. 查询关节最大加速度 Get_Joint_Acc

该函数用于查询关节最大加速度。

int Get_Joint_Acc(SOCKHANDLE ArmSocket, float *acc);

参数：

（1）ArmSocket

Socket 句柄

（2）acc

关节 1~7 加速度数组，单位：$°/s^2$

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.3. 获取关节最小限位 Get_Joint_Min_Pos

该函数用于获取关节最小限位。

```
int Get_Joint_Min_Pos(SOCKHANDLE ArmSocket, float *min_joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）min_joint

存放关节 1~7 最小限位数组，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.4. 获取关节最大限位 Get_Joint_Max_Pos

该函数用于获取关节最大限位。

```
int Get_Joint_Max_Pos(SOCKHANDLE ArmSocket, float *max_joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）max_joint

存放关节 1~7 最大限位数组，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.5. 查询关节最大速度（驱动器)Get_Joint_Drive_Speed

该函数用于查询关节最大速度。

int Get_Joint_Drive_Speed(SOCKHANDLE ArmSocket, float *speed);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

存放关节 1~7 转速数组，单位：°/s

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.6. 查询关节最大加速度（驱动器)Get_Joint_Drive_Acc

该函数用于查询关节最大加速度。

int Get_Joint_Drive_Acc(SOCKHANDLE ArmSocket, float *acc);

参数：

（1）ArmSocket

Socket 句柄

（2）acc

关节 1~7 加速度数组，单位：$°/s^2$

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.7. 获取关节最小限位（驱动器)Get_Joint_Drive_Min_Pos

该函数用于获取关节最小限位。

int Get_Joint_Drive_Min_Pos(SOCKHANDLE ArmSocket, float *min_joint);

参数：

（1）ArmSocket

Socket 句柄

（2）min_joint

存放关节 1~7 最小限位数组，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.8. 获取关节最大限位（驱动器)Get_Joint_Drive_Max_Pos

该函数用于获取关节最大限位。

```
int Get_Joint_Drive_Max_Pos(SOCKHANDLE ArmSocket, float *max_joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）max_joint

存放关节 1~7 最大限位数组，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.9. 获取关节使能状态 Get_Joint_EN_State

该函数用于获取关节使能状态。

```
int Get_Joint_EN_State(SOCKHANDLE ArmSocket, byte *state);
```

参数：

（1）ArmSocket

Socket 句柄

（2）state

存放关节 1~7 使能状态数组，1一使能状态，0-掉使能状态

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.10. 获取关节错误代码 Get_Joint_Err_Flag

该函数用于获取关节错误代码。

```
int Get_Joint_Err_Flag(SOCKHANDLE ArmSocket, uint16_t* state, uint16_t* bstate);
```

参数

（1）ArmSocket

Socket 句柄

（2）state

存放关节错误码（请参考 api 文档中的关节错误码）

（3）bstate

关节抱闸状态（1 代表抱闸未打开，0 代表抱闸已打开）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.3.11. 查询关节软件版本号 Get_Joint_Software_Version

该函数用于查询关节软件版本号。

```
int Get_Joint_Software_Version(SOCKHANDLE ArmSocket, float* version);
```

参数

（1）ArmSocket

Socket 句柄

（2）version

存放关节 1~7 关节软件版本号

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4. 机械臂末端运动参数配置

5.4.1. 设置末端最大线速度 Set_Arm_Line_Speed

该函数用于设置机械臂末端最大线速度。建议使用默认最大线速度，如需更改，设置的机械臂末端最大线加速度与最大线速度的比值需要≥3，否则可能出现运动异常。

```
int Set_Arm_Line_Speed(SOCKHANDLE ArmSocket, float speed, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）speed

末端最大线速度，单位 m/s

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.2. 设置末端最大线加速度 Set_Arm_Line_Acc

该函数用于设置机械臂末端最大线加速度。建议使用默认最大线加速度，如需更改，设置的机械臂末端最大线加速度与最大线速度的比值需要≥3，否则可能出现运动异常。

int Set_Arm_Line_Acc(SOCKHANDLE ArmSocket, float acc, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）acc

末端最大线加速度，单位 m/s^2

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.3. 设置末端最大角速度 Set_Arm_Angular_Speed

该函数用于设置机械臂末端最大角速度。建议使用默认最大角速度，如需更改，设置的机械臂末端最大角加速度与最大角速度的比值需要≥3，否则可能出现运动异常。

int Set_Arm_Angular_Speed(SOCKHANDLE ArmSocket, float speed, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

机械臂末端最大角速度，单位 rad/s

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.4. 设置末端最大角加速度 Set_Arm_Angular_Acc

该函数用于设置机械臂末端最大角加速度。建议使用默认最大角加速度，如需更改，设置的机械臂末端最大角加速度与最大角速度的比值需要≥3，否则可能出现运动异常。

```
int Set_Arm_Angular_Acc(SOCKHANDLE ArmSocket, float acc, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）acc

末端最大角加速度，单位 rad/s^2

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.5. 获取末端最大线速度 Get_Arm_Line_Speed

该函数用于获取机械臂末端最大线速度。

int Get_Arm_Line_Speed(SOCKHANDLE ArmSocket, float *speed);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

末端最大线速度，单位 m/s

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.6. 获取末端最大线加速度 Get_Arm_Line_Acc

该函数用于获取机械臂末端最大线加速度。

int Get_Arm_Line_Acc(SOCKHANDLE ArmSocket, float *acc);

参数：

（1）ArmSocket

Socket 句柄

（2）acc

末端最大线加速度，单位 m/s^2

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.7. 获取末端最大角速度 Get_Arm_Angular_Speed

该函数用于获取机械臂末端最大角速度。

int Get_Arm_Angular_Speed(SOCKHANDLE ArmSocket, float *speed);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

末端最大角速度，单位 rad/s

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.8. 获取末端最大角加速度 Get_Arm_Angular_Acc

该函数用于获取机械臂末端最大角加速度。

int Get_Arm_Angular_Acc(SOCKHANDLE ArmSocket, float *acc);

参数：

（1）ArmSocket

Socket 句柄

（2）acc

末端最大角加速度，单位 rad/s^2

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.9. 设置机械臂末端参数为初始值 Set_Arm_Tip_Init

该函数用于设置机械臂末端参数为初始值。

int Set_Arm_Tip_Init(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block：

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.10. 设置碰撞等级 Set_Collision_Stage

该函数用于设置机械臂动力学碰撞等级，等级 0~8，等级越高，碰撞检测越灵敏，同时也更容易发生误碰撞检测。机械臂上电后默认碰撞等级为 0，即不检测碰撞。

```
int Set_Collision_Stage(SOCKHANDLE ArmSocket, int stage, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）stage

碰撞等级：0~8，0－无碰撞，8－碰撞最灵敏

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.11. 查询碰撞等级 Get_Collision_Stage

该函数用于查询机械臂动力学碰撞等级，等级 0~8，等级越高，碰撞检测越灵敏，同时也更容易发生误碰撞检测。机械臂上电后默认碰撞等级为 0，即不检测碰撞。

```
int Get_Collision_Stage(SOCKHANDLE ArmSocket, int *stage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）stage

碰撞等级，等级：0~8

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.4.12. 设置关节零位补偿角度 Set_Joint_Zero_Offset

该函数用于设置机械臂各关节零位补偿角度，一般在对机械臂零位进行标定后调用该函数。

```
int Set_Joint_Zero_Offset (SOCKHANDLE ArmSocket, float *offset, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）offset

关节 1~7 零位补偿角度数组，角度单位：°

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：该指令用户不可自行使用，必须配合测量设备进行绝对精度补偿时方可使用，否则会导致机械臂参数错误！

5.5. 机械臂末端接口板

5.5.1. 查询末端接口板软件版本号 Get_Tool_Software_Version

该函数用于查询末端接口板软件版本号

int Get_Tool_Software_Version(SOCKHANDLE ArmSocket, int* version);

参数：

（1）ArmSocket

Socket 句柄

（2）version

末端接口板软件版本号

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.6. 工具坐标系设置

5.6.1. 标定点位 Auto_Set_Tool_Frame

该函数用于六点法自动设置工具坐标系（标记点位），机械臂控制器最多只能存储 10 个工具坐标系信息，在建立新的工具坐标系之前，请确认工具坐标系数量没有超过限制，否则建立新工具坐标系无法成功。

int Auto_Set_Tool_Frame(SOCKHANDLE ArmSocket, byte point_num, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）point_num

1~6 代表 6 个标定点。

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

说明：控制器只能存储十个工具坐标系，超过十个控制器不予响应，请在标定前查询已有工具

5.6.2. 生成工具坐标系 Generate_Auto_Tool_Frame

该函数用于六点法自动设置工具坐标系（生成坐标系），机械臂控制器最多只能存储 10 个工具坐标系信息，在建立新的工具坐标系之前，请确认工具坐标系数量没有超过限制，否则建立新工具坐标系无法成功。

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

参数：

（1）ArmSocket

Socket 句柄

（2）name

工具坐标系名称，不能超过十个字节。

（3）payload

新工具坐标系执行末端负载重量　单位 kg

（4）x,y,z

新工具坐标系执行末端负载位置 位置 x，y，z 单位 mm

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

说明：控制器只能存储十个工具坐标系，超过十个控制器不予响应，请在标定前查询已有工具坐标系

5.6.3. 手动设置工具坐标系 Manual_Set_Tool_Frame

该函数用于手动设置工具坐标系，机械臂控制器最多只能存储 10 个工具坐标系信息，在建立新的工具坐标系之前，请确认工具坐标系数量没有超过限制，否则建立新工具坐标系无法成功。

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

参数：

（1）ArmSocket

Socket 句柄

（2）name

工具坐标系名称，不能超过十个字节。

（3）pose

新工具坐标系执行末端相对于机械臂法兰中心的位姿

（4）payload

新工具坐标系执行末端负载重量　单位 kg

（5）x,y,z

新工具坐标系执行末端负载位置 位置 x，y，z 单位 mm

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

说明：控制器只能存储十个工具坐标系，超过十个控制器不予响应，请在标定前查询已有工具

5.6.4. 切换当前工具坐标系 Change_Tool_Frame

该函数用于切换当前工具坐标系

```
int Change_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）name

目标工具坐标系名称

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.6.5. 删除指定工具坐标系 Delete_Tool_Frame

该函数用于删除指定工具坐标系

```
int Delete_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）name

要删除的工具坐标系名称

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：删除工具坐标系后，机械臂将切换到机械臂法兰末端工具坐标系。

5.6.6. 修改指定工具坐标系 Update_Tool_Frame

该函数用于修改指定工具坐标系

int Update_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose, float payload,float x,float y,float z);

参数：

（1）ArmSocket

Socket 句柄

（2）name

要修改的工具坐标系名称

（3）pose

更新执行末端相对于机械臂法兰中心的位姿

（4）payload

末端负载 单位 kg。

（5）x

质心位置 x 单位 m。

（6）y

质心位置 y 单位 m。

（7）z

质心位置 z 单位 m。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.6.7. 设置工具坐标系的包络参数 Set_Tool_Envelope

该函数用于设置工具坐标系的包络参数

int Set_Tool_Envelope(SOCKHANDLE ArmSocket, ToolEnvelopeList list);

参数：

（1）ArmSocket

Socket 句柄

（2）list

包络参数列表，每个工具最多支持 5 个包络球，可以没有包络

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.6.8. 获取工具坐标系的包络参数 Get_Tool_Envelope

该函数用于获取工具坐标系的包络参数

int Get_Tool_Envelope(SOCKHANDLE ArmSocket, const char *name, ToolEnvelopeList *list);

参数：

（1）ArmSocket

Socket 句柄

（2）name

控制器中已存在的工具坐标系名称

（3）list

包络参数列表，每个工具最多支持 5 个包络球，可以没有包络。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.7. 工具坐标系查询

5.7.1. 获取当前工具坐标系 Get_Current_Tool_Frame

该函数用于获取当前工具坐标系。

int Get_Current_Tool_Frame(SOCKHANDLE ArmSocket, FRAME *tool);

参数：

（1）ArmSocket

Socket 句柄

（2）tool

返回的工具坐标系参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.7.2. 获取指定工具坐标系 Get_Given_Tool_Frame

该函数用于获取指定工具坐标系

int Get_Given_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, FRAME *tool);

参数：

（1）ArmSocket

Socket 句柄

（2）name

指定的工具坐标系名称

（3）tool

返回的工具坐标系参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.7.3. 获取所有工具坐标系名称 Get_All_Tool_Frame

该函数用于获取所有工具坐标系名称

int Get_All_Tool_Frame(SOCKHANDLE ArmSocket, FRAME_NAME *names, int *len);

参数：

（1）ArmSocket

Socket 句柄

（2）names

返回的工具坐标系名称数组

（3）len

返回工具坐标系数量。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.8. 工作坐标系设置

5.8.1. 自动设置工作坐标系 Auto_Set_Work_Frame

该函数用于三点法自动设置工作坐标系，机械臂控制器最多只能存储 10 个工作坐标系信息，在建立新的工作坐标系之前，请确认工作坐标系数量没有超过限制，否则建立新工作坐标系无法成功。

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

参数：

（1）ArmSocket

Socket 句柄

（2）name

工作坐标系名称，不能超过十个字节。

（3）point_num

1~3 代表 3 个标定点，依次为原点、X 轴上任意点、Y 轴上任意点，4 代表生成坐标系。

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

说明：控制器只能存储十个工作坐标系，超过十个控制器不予响应，请在标定前查询已有工作坐标系

5.8.2. 手动设置工作坐标系 Manual_Set_Work_Frame

该函数用于手动设置工作坐标系。

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

参数：

（1）ArmSocket

Socket 句柄

（2）name

工作坐标系名称，不能超过十个字节。

（3）pose

新工作坐标系相对于基坐标系的位姿。

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

说明：控制器只能存储十个工作坐标系，超过十个控制器不予响应，请在标定前查询已有工作坐标系

5.8.3. 切换当前工作坐标系 Change_Work_Frame

该函数用于切换当前工作坐标系

```
int Change_Work_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）name

目标工作坐标系名称

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.8.4. 删除指定工作坐标系 Delete_Work_Frame

该函数用于删除指定工作坐标系。

int Delete_Work_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）name

要删除的工作坐标系名称

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：删除工作坐标系后，机械臂将切换到机械臂基坐标系

5.8.5. 修改指定工作坐标系 Update_Work_Frame

该函数用于修改指定工作坐标系。

int Update_Work_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose);

参数：

（1）ArmSocket

Socket 句柄

（2）name

要修改的工作坐标系名称

（3）pose

更新工作坐标系相对于基坐标系的位姿

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.9. 工作坐标系查询

5.9.1. 获取当前工作坐标系 Get_Current_Work_Frame

该函数用于获取当前工作坐标系。

int Get_Current_Work_Frame(SOCKHANDLE ArmSocket, FRAME *frame);

参数：

（1）ArmSocket

Socket 句柄

（2）frame

返回的工作坐标系位姿参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.9.2. 获取指定工作坐标系 Get_Given_Work_Frame

该函数用于获取指定工作坐标系

int Get_Given_Work_Frame(SOCKHANDLE ArmSocket, const char *name, POSE *pose);

参数：

（1）ArmSocket

Socket 句柄

（2）name

指定的工作坐标系名称

（3）pose

返回的工作坐标系位姿参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.9.3. 获取所有工作坐标系名称 Get_All_Work_Frame

该函数用于获取所有工作坐标系名称

int Get_All_Work_Frame(SOCKHANDLE ArmSocket, FRAME_NAME *names, int *len);

参数：

（1）ArmSocket

Socket 句柄

（2）names

返回的工作坐标系的名称数组

（3）len

返回的工作坐标系的数量。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10. 机械臂状态查询

5.10.1. 获取机械臂当前状态 Get_Current_Arm_State

该函数用于获取机械臂当前状态

int Get_Current_Arm_State(SOCKHANDLE ArmSocket, float *joint, Pose *pose, uint16_t *Arm_Err, uint16_t *Sys_Err);

参数：

（1）ArmSocket

Socket 句柄

（2）joint

关节 1~7 角度数组

（3）pose

机械臂当前位姿

（4）Arm_Err

机械臂运行错误代码

（5）Sys_Err

控制器错误代码

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.2. 获取关节温度 Get_Joint_Temperature

该函数用于获取关节当前温度。

int Get_Joint_Temperature(SOCKHANDLE ArmSocket, float *temperature);

参数：

（1）ArmSocket

Socket 句柄

（2）temperature

关节 1~7 温度数组

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.3. 获取关节电流 Get_Joint_Current

该函数用于获取关节当前电流。

```
int Get_Joint_Current(SOCKHANDLE ArmSocket, float *current);
```

参数：

（1）ArmSocket

Socket 句柄

（2）current

关节 1~7 电流数组

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.4. 获取关节电压 Get_Joint_Voltage

该函数用于获取关节当前电压。

```
int Get_Joint_Voltage(SOCKHANDLE ArmSocket, float *voltage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）voltage

关节 1~7 电压数组

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.5. 获取关节当前角度 Get_Joint_Degree

该函数用于获取机械臂各关节的当前角度

```
int Get_Joint_Degree (SOCKHANDLE ArmSocket, float *joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）joint

关节 1~7 当前角度数组，单位：°；

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.6. 获取所有状态 Get_Arm_All_State

该函数用于获取机械臂所有状态

```
int Get_Arm_All_State(SOCKHANDLE ArmSocket, JOINT_STATE *joint_state);
```

参数：

（1）ArmSocket

Socket 句柄

（2）joint_state

机械臂所有状态；

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.10.7. 获取轨迹规划计数 Get_Arm_Plan_Num

该函数用于获取机械臂轨迹规划计数

int Get_Arm_Plan_Num(SOCKHANDLE ArmSocket, int* plan);

参数：

（1）ArmSocket

Socket 句柄

（2）plan

查询到的轨迹规划计数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.11. 机械臂初始位姿

5.11.1. 设置初始位姿角度 Set_Arm_Init_Pose

该函数用于设置机械臂的初始位姿角度。

int Set_Arm_Init_Pose(SOCKHANDLE ArmSocket, const float *target, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）target

机械臂初始位置关节角度数组，单位：°

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.11.2. 获取初始位姿角度 Get_Arm_Init_Pose

该函数用于获取机械臂初始位姿角度。

```
int Get_Arm_Init_Pose(SOCKHANDLE ArmSocket, float *joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）joint

机械臂初始位姿关节角度数组，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.11.3. 设置安装角度 Set_Install_Pose

该函数用于设置机械臂安装方式。

```
int Set_Install_Pose(SOCKHANDLE ArmSocket, float x, float y, float z, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）x

旋转角，单位 °

（3）y

俯仰角，单位 °

（4）z

方位角，单位 °

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.11.4. 查询安装角度 Get_Install_Pose

该函数用于查询机械臂安装角度。

```
int Get_Install_Pose(SOCKHANDLE ArmSocket, float *fx, float *fy, float *fz);
```

参数：

（1）ArmSocket

Socket 句柄

（2）fx

旋转角（out），单位 °

（3）fy

俯仰角（out），单位 °

（4）fz

方位角（out），单位 °

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12. 机械臂运动规划

5.12.1. 关节空间运动 Movej_Cmd

该函数用于关节空间运动。

int Movej_Cmd(SOCKHANDLE ArmSocket, const float *joint, byte v, float r, int trajectory_connect, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）joint

目标关节 1~7 角度数组，单位：°

（3）v

速度百分比系数，1~100。

（4）r

交融半径百分比系数，0~100。

（5）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

注意：trajectory_connect 参数为 1 交融半径才生效，如果为 0 则交融半径不生效

5.12.2. 笛卡尔空间直线运动 Movel_Cmd

该函数用于笛卡尔空间直线运动。

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

参数：

（1）ArmSocket

Socket 句柄

（2）pose

目标位姿，位置单位：米，姿态单位：弧度

（3）v

速度百分比系数，1~100

（4）r

交融半径百分比系数，0~100。

（5）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

注意：trajectory_connect 参数为 1 交融半径才生效，如果为 0 则交融半径不

生效

5.12.3. 笛卡尔空间圆弧运动 Movec_Cmd

该函数用于笛卡尔空间圆弧运动

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

参数：

（1）ArmSocket

Socket 句柄

（2）pose_vai

中间点位姿，位置单位：米，姿态单位：弧度

（3）pose_to

终点位姿，位置单位：米，姿态单位：弧度

（4）v

速度比例 1~100，即规划速度和加速度占机械臂末端最大角速度和角加速度的百分比

（5）r

交融半径百分比系数，0~100。

（6）loop

规划圈数，目前默认 0。

（7）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（8）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

注意：trajectory_connect 参数为 1 交融半径才生效，如果为 0 则交融半径不生效

5.12.4. 关节角度 CANFD 透传 Movej_CANFD

该函数用于角度不经规划，直接通过 CANFD 透传给机械臂，使用透传接口时，请勿使用其他运动接口。

```
int Movej_CANFD(SOCKHANDLE ArmSocket, const float *joint, bool follow, float expand);
```

参数：

（1）ArmSocket

Socket 句柄

（2）joint

关节 1~7 目标角度数组，单位：°

（3）follow

true-高跟随，false-低跟随。若使用高跟随，透传周期要求不超过 10ms。

（4）expand

扩展关节目标位置，单位°。如果存在通用扩展轴，并需要进行透传，可使用该参数进行透传发送。不需要时传入 0 即可

备注：

透传周期越快，控制效果越好，越平顺。基础系列 WIFI 和网口模式透传周期最快 20ms，USB 和 RS48 模式透传周期最快 10ms。高速网口的透传周期最快也可到 10ms，不过在使用该高速网口前，需要使用指令打开配置。另外 I 系列有线网口周期最快可达 5ms。

用户使用该函数时请做好轨迹规划，轨迹规划的平滑成都决定了机械臂的运行状态，帧与帧之间关节的角度差不能超过 10°，并保证关节规划的速度不超过 180°/s，否则关节不会响应。

由于该模式直接下发给机械臂，不经控制器规划，因此只要控制器运行正常并且目标角度在可达范围内，机械臂立即返回成功指令，此时机械臂可能仍在运行；若有错误，立即返回失败指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.5. 位姿 CANFD 透传 Movep_CANFD

该函数用于位姿不经规划，直接通过 CANFD 透传给机械臂，使用透传接口时，请勿使用其他运动接口。

```
int Movep_CANFD(SOCKHANDLE ArmSocket, Pose pose, bool follow);
```

参数：

（1）ArmSocket

Socket 句柄

（2）pose

位姿（优先采用四元数表达）

（3）follow

true-高跟随，false-低跟随。若使用高跟随，透传周期要求不超过 10ms。

备注：

透传周期越快，控制效果越好、越平顺。基础系列 WIFI 和网口模式透传周期最快 20ms，USB 和 RS485 模式透传周期最快 10ms。高速网口的透传周期最快也可到 10ms，不过在使用该高速网口前，需要使用指令打开配置。另外 I 系列有线网口周期最快可达 5ms。

由于该模式直接下发给机械臂，不经控制器规划，因此只要控制器运行正常并且目标角度在可达范围内，机械臂立即返回成功指令，此时机械臂可能仍在运行；若有错误，立即返回失败指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.6. 计算环绕运动位姿 MoveRotate_Cmd

该函数用于计算环绕运动位姿并按照结果运动

int MoveRotate_Cmd(SOCKHANDLE ArmSocket, int rotateAxis, float rotateAngle,Pose choose_axis, byte v, float r, int trajectory_connect, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）rotateAxis

旋转轴： 1：x 轴， 2：y 轴， 3：z 轴

（3）rotateAngle

旋转角度： 旋转角度， 单位：°

（4）choose_axis

指定计算时使用的坐标系

（5）v

速度

（6）r

交融半径百分比系数，0~100

（7）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（8）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.7. 沿工具端位姿移动 MoveCartesianTool_Cmd

该函数用于沿工具端位姿移动

int MoveCartesianTool_Cmd(SOCKHANDLE ArmSocket, float *Joint_Cur, float movelengthx,float movelengthy, float movelengthz, int m_dev, byte v, float r, int trajectory_connect, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）Joint_Cur

当前关节角度

（3）movelengthx

沿 X 轴移动长度，米为单位

（4）Movelengthy

沿 X 轴移动长度，米为单位

（5）movelengthz

沿 Z 轴移动长度，米为单位

（6）m_dev

机械臂型号

（7）v

速度

（8）r

交融半径百分比系数，0~100

（9）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（10）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.8. 快速急停 Move_Stop_Cmd

该函数用于突发状况，机械臂以最快速度急停，轨迹不可恢复。

int Move_Stop_Cmd(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.9. 暂停当前规划 Move_Pause_Cmd

该函数用于轨迹暂停，暂停在规划轨迹上，轨迹可恢复。

int Move_Pause_Cmd(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.10. 继续当前轨迹 Move_Continue_Cmd

该函数用于轨迹暂停后，继续当前轨迹运动

int Move_Continue_Cmd(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.11. 清除当前轨迹 Clear_Current_Trajectory

该函数用于清除当前轨迹，必须在暂停后使用，否则机械臂会发生意外！！!!

int Clear_Current_Trajectory(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.12. 清除所有轨迹 Clear_All_Trajectory

该函数用于清除所有轨迹，必须在暂停后使用，否则机械臂会发生意外！！！！

int Clear_All_Trajectory(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.12.13. 关节空间运动 Movej_P_Cmd

该函数用于关节空间运动到目标位姿

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

参数：

（1）ArmSocket

Socket 句柄

（2）pose

目标位姿，位置单位：米，姿态单位：弧度。

注意：该目标位姿必须是机械臂当前工具坐标系相对于当前工作坐标系的位，

用户在使用该指令前务必确保，否则目标位姿会出错！！！

（3）v

速度百分比系数，1~100

（4）r

交融半径百分比系数，0~100。

（5）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

注意：该运动暂不支持轨迹交融。

5.12.14. 样条曲线运动 Moves_Cmd

该函数用于样条曲线运动到目标位姿

int Moves_Cmd(SOCKHANDLE ArmSocket, POSE pose, byte v, float r, int trajectory_connect, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）pose

目标位姿，位置单位：米，姿态单位：弧度。

（3）v

速度百分比系数，1~100

（4）r

交融半径百分比系数，0~100。

（5）trajectory_connect

代表是否和下一条运动一起规划，0 代表立即规划，1 代表和下一条轨迹一起规划，当为 1 时，轨迹不会立即执行，样条曲线运动需至少连续下发三个点位，否则运动轨迹为直线。

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待机械臂到达位置或者规划失败。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

注意：该运动暂不支持轨迹交融。

5.13. 机械臂示教

5.13.1. 关节示教 Joint_Teach_Cmd

该函数用于关节示教，关节从当前位置开始按照指定方向转动，接收到停止指令或者到达关节限位后停止。

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

参数：

（1）ArmSocket

Socket 句柄

（2）num

示教关节的序号，1~7

（3） direction

示教方向，0—负方向，1—正方向

（4）v

速度比例 1~100，即规划速度和加速度占关节最大线转速和加速度的百分比

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.13.2. 位置示教 Pos_Teach_Cmd

该函数用于当前坐标系下（默认为当前工作坐标系下，调用 5.13.5 切换示教运动坐标系 Set_Teach_Frame 可切换为工具坐标系），笛卡尔空间位置示教。机械臂在当前工作坐标系下，按照指定坐标轴方向开始直线运动，接收到停止指令或者该处无逆解时停止。

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

参数：

（1）ArmSocket

Socket 句柄

（2）type

示教类型

（3） direction

示教方向，0-负方向，1-正方向

（4）v

速度百分比系数，1~100

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.13.3. 姿态示教 Ort_Teach_Cmd

该函数用于当前工作坐标系下（默认为当前工作坐标系下，调用 5.13.5 切换示教运动坐标系 Set_Teach_Frame 可切换为工具坐标系），笛卡尔空间末端姿态示教。机械臂在当前工作坐标系下，绕指定坐标轴旋转，接收到停止指令或者该处无逆解时停止。

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

参数：

（1）ArmSocket

Socket 句柄

（2）type

示教类型

（3）direction

示教方向，0-负方向，1-正方向

（4）v

速度比例 1~100，即规划速度和加速度占机械臂末端最大角速度和角加速度的百分比

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.13.4. 示教停止 Teach_Stop_Cmd

该函数用于示教停止。

```
int Teach_Stop_Cmd(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.13.5. 切换示教运动坐标系 Set_Teach_Frame

该函数用于切换示教运动坐标系。

```
int Set_Teach_Frame(SOCKHANDLE ArmSocket, int type, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）type

0： 基坐标系运动, 1： 工具坐标系运动

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.13.6. 获取示教运动坐标系 Get_Teach_Frame

该函数用于获取示教运动坐标系。

```
int Get_Teach_Frame(SOCKHANDLE ArmSocket, int* type);
```

参数：

（1）ArmSocket

Socket 句柄

（2）type

0： 基坐标系运动, 1： 工具坐标系运动

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.14. 机械臂步进

5.14.1. 关节步进 Joint_Step_Cmd

该函数用于关节步进。关节在当前位置下步进指定角度。

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

参数：

（1）ArmSocket

Socket 句柄

（2）num

关节序号，1~7

（3）step

步进的角度

（4）v

速度比例 1~100，即规划速度和加速度占指定关节最大关节转速和关节加速度的百分比

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.14.2. 位置步进 Pos_Step_Cmd

该函数用于当前坐标系下（默认为当前工作坐标系下，调用 5.13.5 切换示教运动坐标系 Set_Teach_Frame 可切换为工具坐标系），位置步进。机械臂末端在工作坐标系下，朝指定坐标轴方向步进指定距离，到达位置返回成功指令，规划错误返回失败指令。

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

参数：

（1）ArmSocket

Socket 句柄

（2）type

示教类型

（3）step

步进的距离，单位 m，精确到 0.001mm

（4）v

速度百分比系数，1~100

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.14.3. 姿态步进 Ort_Step_Cmd

该函数用于当前坐标系下（默认为当前工作坐标系下，调用 5.13.5 切换示教运动坐标系 Set_Teach_Frame 可切换为工具坐标系），姿态步进。机械臂末端在当前坐标系下，绕指定坐标轴方向步进指定弧度，到达位置返回成功指令，规划错误返回失败指令。

int Ort_Step_Cmd(SOCKHANDLE ArmSocket, ORT_TEACH_MODES type, float step,byte v,bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）type

示教类型

（3）step

步进的弧度，单位 rad，精确到 0.001rad

（4）v

速度比例 1~100，即规划速度和加速度占机械臂末端最大角速度和角加速度的百分比

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15. 控制器配置

5.15.1. 获取控制器状态 Get_Controller_State

该函数用于获取控制器状态。

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

参数：

（1）ArmSocket

Socket 句柄

（2）voltage

返回的电压

（3）current

返回的电流

（4）temperature

返回的温度

（5）sys_err

控制器运行错误代码

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.2. 设置 WiFi AP 模式设置 Set_WiFi_AP_Data

该函数用于控制器 WiFi AP 模式设置，非阻塞模式，机械臂收到后更改参数，蜂鸣器响后代表更改成功，控制器重启，以 WIFI AP 模式通信。

int WiFi_AP_Data(SOCKHANDLE ArmSocket, const char *wifi_name, const char*

password);

参数：

（1）ArmSocket

Socket 句柄

（2）wifi_name

控制器 wifi 名称

（3）password

wifi 密码

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.3. 设置 WiFi STA 模式设置 Set_WIFI_STA_Data

该函数用于控制器 WiFi STA 模式设置，非阻塞模式，机械臂收到后更改参数，蜂鸣器响后代表更改成功，控制器重启，以 WIFI STA 模式通信。

int Set_WIFI_STA_Data(SOCKHANDLE ArmSocket, const char *router_name,const char* password);

参数：

（1）ArmSocket

Socket 句柄

（2）router_name

路由器名称

（3）password

路由器 Wifi 密码

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

/// **注意：**非阻塞模式：设置成功后，机械臂进入 WIFI STA 通信模式

5.15.4. 设置 UART_USB 接口波特率 Set_USB_Data

该函数用于控制器 UART_USB 接口波特率设置非阻塞模式，机械臂收到后更改参数，然后立即通过 UART-USB 接口与外界通信。

该指令下发后控制器会记录当前波特率，断电重启后仍会使用该波特率对外通信。

```
int Set_USB_Data(SOCKHANDLE ArmSocket, int baudrate);
```

参数：

（1）ArmSocket

Socket 句柄

（2）baudrate

波特率：9600，19200，38400，115200 和 460800，若用户设置其他数据，控制器会默认按照 460800 处理。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：

该指令下发后控制器会记录当前波特率，需要请求保存参数函数进行保存，否则断电重启后恢复为修改之前的波特率。

5.15.5. 设置 RS485 配置 Set_RS485

该函数用于控制器设置 RS485 配置。

```
int Set_RS485(SOCKHANDLE ArmSocket, int baudrate);
```

参数：

（1）ArmSocket

Socket 句柄

（2）baudrate

波特率：9600，19200，38400，115200 和 460800，若用户设置其他数据，控制器会默认按照 460800 处理。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：

该指令下发后该指令下发后，若 Modbus 模式为打开状态，则会自动关闭，同时控制器会记录当前波特率，需要请求保存参数函数进行保存，否则断电重启后恢复为修改之前的波特率。

5.15.6. 设置机械臂电源 Set_Arm_Power

该函数用于设置机械臂电源

```
int Set_Arm_Power(SOCKHANDLE ArmSocket, bool cmd, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）cmd

true-上电， false-断电

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待

控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.7. 获取机械臂电源 Get_Arm_Power_State

该函数用于获取机械臂电源

int Get_Arm_Power_State(SOCKHANDLE ArmSocket, int* power);

参数：

（1）ArmSocket

Socket 句柄

（2）power

获取到的机械臂电源状态：1-上电， 0-断电

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.8. 读取机械臂软件版本 Get_Arm_Software_Version

该函数用于读取机械臂软件版本

int Arm_Software_Version(SOCKHANDLE ArmSocket, char* plan_version, char* ctrl_version,char *kernal1, char *kernal2, char* product_version);

参数：

（1）ArmSocket

Socket 句柄

（2）plan_version

读取到的用户接口内核版本号

（3）ctrl_version

实时内核版本号

（4）kernal1

实时内核子核心 1 版本号

（5）kernal2

实时内核子核心 2 版本号

（6）product_version

机械臂型号，仅 I 系列机械臂支持[-I]

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.9. **获取控制器的累计运行时间** Get_System_Runtime

读取控制器的累计运行时间。

int Get_System_Runtime(SOCKHANDLE ArmSocket,int* day,int* hour,int* min,int* sec);

参数：

（1）ArmSocket

Socket 句柄

（2）day

天

（3）hour

小时

（4）min

分

（5）sec

秒

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.10. 清空控制器累计运行时间 Clear_System_Runtime

该函数用于清空控制器累计运行时间

```
int Clear_System_Runtime(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.11. 获取关节累计转动角度 Get_Joint_Odom

该函数用于读取关节的累计转动角度

```
int Get_Joint_Odom(SOCKHANDLE ArmSocket, float* odom);
```

参数：

（1）ArmSocket

Socket 句柄

（2）odom

各关节累计的转动角度值

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.12. 清除关节累计转动角度 Clear_Joint_Odom

该函数用于清空关节累计转动角度

```
int Clear_Joint_Odom(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.13. 配置高速网口 Set_High_Speed_Eth--基础系列

该函数用于配置高速网口

```
int Set_High_Speed_Eth(SOCKHANDLE ArmSocket, byte num, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）num

0-关闭　1-开启

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.14. 设置高速网口网络配置 Set_High_Ethernet--基础系列

该函数用于设置高速网口网络配置[配置通讯内容]

int Set_High_Ethernet(SOCKHANDLE ArmSocket, const char * ip, const char * mask, const char * gateway);

参数：

（1）ArmSocket

Socket 句柄

（2）ip

网络地址

（3）mask

子网掩码

（4）gateway

网关

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.15. 获取高速网口网络配置 Get_High_Ethernet--基础系列

该函数用于获取高速网口网络配置[配置通讯内容]

int Get_High_Ethernet(SOCKHANDLE ArmSocket, char * ip, char * mask, char * gateway, char * mac);

参数：

（1）ArmSocket

Socket 句柄

（2）ip

网络地址

（3）mask

子网掩码

（4）gateway

网关

（5）mac

MAC 地址

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.16. 保存参数 Save_Device_Info_All--基础系列

该函数用于保存所有参数

int Save_Device_Info_All(SOCKHANDLE ArmSocket);

参数：

（1）ArmSocket

Socket 句柄

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.17. 配置有线网卡 IP 地址 Set_NetIP--I 系列

该函数用于配置有线网卡 IP 地址[-I]

int Set_NetIP(SOCKHANDLE ArmSocket, const char * ip);

参数：

（1）ArmSocket

Socket 句柄

（2）ip

网络地址

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.18. 查询有线网卡网络信息 Get_Wired_Net--I 系列

该函数用于查询有线网卡网络信息[-I]

int Get_Wired_Net(SOCKHANDLE ArmSocket, char * ip, char * mask, char * mac);

参数：

（1）ArmSocket

Socket 句柄

（2）ip

网络地址

（3）mask

子网掩码

（4）mac

MAC 地址

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.19. 查询无线网卡网络信息 Get_Wifi_Net--I 系列

该函数用于查询无线网卡网络信息[-I]

```
int Get_Wifi_Net(SOCKHANDLE ArmSocket, WiFi_Info* network);
```

参数：

（1）ArmSocket

Socket 句柄

（2）net_work

无线网卡网络信息结构体

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.20. 恢复网络出厂设置 Set_Net_Default--I 系列

该函数用于恢复网络出厂设置[-I]

```
int Set_Net_Default(SOCKHANDLE ArmSocket);
```

参数：

（1）ArmSocket

Socket 句柄

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.21. 清除系统错误代码 Clear_System_Err

该函数用于清除系统错误代码

int Clear_System_Err(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.22. 读取机械臂软件信息 Get_Arm_Software_Info

该函数用于读取机械臂软件信息。

int Get_Arm_Software_Info(SOCKHANDLE ArmSocket, ArmSoftwareInfo* software_info);

参数：

（1）ArmSocket

Socket 句柄

（2）software_info

机械臂软件信息。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.23. 设置机械臂模式（仿真/真实)Set_Arm_Run_Mode

该函数用于设置机械臂模式（仿真/真实)。

```
int Set_Arm_Run_Mode(SOCKHANDLE ArmSocket, int mode);
```

参数：

（1）ArmSocket

Socket 句柄

（2）mode

模式 0:仿真 1:真实

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.15.24. 获取机械臂模式（仿真/真实)Get_Arm_Run_Mode

该函数用于获取机械臂模式（仿真/真实)。

```
int Get_Arm_Run_Mode(SOCKHANDLE ArmSocket, int* mode);
```

参数：

（1）ArmSocket

Socket 句柄

（2）mode

模式 0:仿真 1:真实

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16. IO 配置

机械臂具有 IO 端口，基础系列数量和分类如下所示：

数字输出：DO	4 路，可配置为 0~12V
数字输入：DI	3 路，可配置为 0~12V
模拟输出：AO	4 路，输出电压 0~10V
模拟输入：AI	4 路，输入电压 0~10V

I 系列数量和分类如下所示：

数字 IO：DO/DI 复用	4 路，可配置为 0~24V

5.16.1. 设置控制器端数字 IO 模式 Set_IO_Mode--I 系列

该函数用于设置数字 IO 模式[-I]。

int Set_IO_Mode(SOCKHANDLE ArmSocket, byte io_num, byte io_mode);

参数：

（1）ArmSocket

Socket 句柄

（2）io_num

IO 端口号，范围：1~4

（3）Io_mode

模式，0-通用输入模式，1-通用输出模式、2-输入开始功能复用模式，3-输入暂停功能复用模式，4-输入继续功能复用模式，5-输入急停功能复用模式，6-输入进入电流环拖动复用模式，7-输入进入力只动位置拖动模式，8-输入进入力只动姿态拖动模式，9-输入进入力位姿结合拖动复用模式，10-输入外部轴最大软限位复用模式，11-输入外部轴最小软限位复用模式

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16.2. 设置数字 IO 输出状态 Set_DO_State

该函数用于设置数字 IO 输出。

int Set_DO_State(SOCKHANDLE ArmSocket, byte io_num, bool state, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）io_num

IO 端口号，范围：1~4

（3）state

IO 状态，true-高， false-低

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16.3. 查询指定 IO 输出状态 Get_IO_State--I 系列

该函数用于查询指定 IO 输出状态。

int Get_IO_State(SOCKHANDLE ArmSocket, byte num, byte *state, byte *mode);

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定数字 IO 通道号，范围 1~4

（3）state

IO 状态

（4）mode

0-通用输入模式，1-通用输出模式、2-输入开始功能复用模式，3-输入暂停功能复用模式，4-输入继续功能复用模式，5-输入急停功能复用模式，6-输入进入电流环拖动复用模式，7-输入进入力只动位置拖动模式，8-输入进入力只动姿态拖动模式，9-输入进入力位姿结合拖动复用模式，10-输入外部轴最大软限位复用模式，11-输入外部轴最小软限位复用模式

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.4. 查询数字 IO 输出状态 Get_DO_State--基础系列

该函数用于获取数字 IO 输出状态。

```
int Get_DO_State(SOCKHANDLE ArmSocket, byte num, byte *state);
```

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定数字 IO 通道号，范围 1~4

（3）state

指定数字 IO 通道返回的状态，1-高，　0-低

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.5. 查询数字 IO 输入状态 Get_DI_State--基础系列

该函数用于获取数字 IO 输入状态。

int Get_DI_State(SOCKHANDLE ArmSocket, byte num, byte *state);

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定数字 IO 通道号，范围 1~3

（3）state

指定数字 IO 通道返回的状态，1-高，　0-低

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.6. 设置模拟 IO 输出状态 Set_AO_State--基础系列

该函数用于获取数字 IO 输出状态。

int Set_AO_State(SOCKHANDLE ArmSocket, byte io_num, float voltage, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定通道号，1~4

（3）state

IO 输出电压，分辨率 0.001V，范围：0~10000，代表输出电压 0v~10v

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.7. 查询模拟 IO 输出状态 Get_AO_State--基础系列

该函数用于获取模拟 IO 输出状态。

```
int Get_AO_State(SOCKHANDLE ArmSocket, byte num, byte *voltage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定通道号，1~4

（3）state

IO 输出电压，分辨率 0.001V，范围：0~10000，代表输出电压 0v~10v

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.8. 查询模拟 IO 输入状态 Get_AI_State--基础系列

该函数用于获取模拟 IO 输入状态。

```
int Get_AI_State(SOCKHANDLE ArmSocket, byte num, byte *voltage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定通道号，1~4

（3）state

IO 输入电压，分辨率 0.001V，范围：0~10000，代表输出电压 0v~10v

返回值：

成功返回：0；失败返回：错误码，rm_define.h 查询。

5.16.9. 查询所有 IO 的输入状态 Get_IO_Input

该函数用于查询所有 IO 的输入状态。

```
int Get_IO_Input(SOCKHANDLE ArmSocket, int *DI_state, float *AI_voltage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）DI_state

数字输入状态数组，1-高，0-低

（3） AI_voltage

模拟 IO 输入通道 1~4 输入电压数组

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16.10. 查询所有 IO 的输出状态 Get_IO_Output

该函数用于查询所有 IO 的输出状态。

int Get_IO_Output(SOCKHANDLE ArmSocket, int *DO_state, float *AO_voltage);

参数：

（1）ArmSocket

Socket 句柄

（2）DO_state

数字输入状态数组，1-高，0-低

（3）AO_voltage

模拟 IO 输出通道 1~4 输出电压数组

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16.11. 设置电源输出 Set_Voltage--I 系列

该函数用于设置控制器端电源输出。

int Set_Voltage(SOCKHANDLE ArmSocket, byte voltage_type, bool start_enable);

参数：

（1）ArmSocket

Socket 句柄

（2）voltage_type

电源输出类型，范围：0~3(0-0V，2-12V，3-24V)

（3）start_enable

true-开机启动时即输出此配置电压，false-取消开启启动配置电压

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.16.12. 获取电源输出 Get_Voltage--I 系列

该函数用于获取控制器端电源输出。

int Get_Voltage(SOCKHANDLE ArmSocket, byte * voltage_type);

参数：

（1）ArmSocket

Socket 句柄

（2）voltage_type

电源输出类型，范围：0~3(0-0V，2-12V，3-24V)

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.17. 末端工具 IO 配置

机械臂末端工具端具有 IO 端口，数量和分类如下所示：

电源输出	1 路，可配置为 0V/5V/12V/24V
数字 IO	2 路，输入输出可配置 输入：参考电平 12V~24V 输出：5~24V，与输出电压一致
通讯接口	1 路，可配置为 RS485

5.17.1. 设置工具端数字 IO 输出状态 Set_Tool_DO_State

该函数用于配置工具端指定数字 IO 输出状态。

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

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定数字 IO 输出通道号，范围 1~2

（3）state

输入参数 true-高， false-低

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.17.2. 设置工具端数字 IO 模式 Set_Tool_IO_Mode

该函数用于设置数字 IO 模式。

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

参数：

（1）ArmSocket

Socket 句柄

（2）num

指定数字 IO 端口号，范围 1~2

（3）state

模式，0-输入状态，1-输出状态

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.17.3. 查询工具端数字 IO 状态 Get_Tool_IO_State

该函数用于查询工具端数字 IO 状态。

```
int Get_Tool_IO_State(SOCKHANDLE ArmSocket, float* IO_Mode, float* IO_State);
```

参数：

（1）ArmSocket

Socket 句柄

（2）IO_Mode

指定数字 IO 通道模式（范围 1~2）， 0-输入模式，1-输出模式

（3）IO_state

指定数字 IO 通道当前输入状态（范围 1~2），1-高电平，0-低电平

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.17.4. 设置工具端电源输出 Set_Tool_Voltage

该函数用于设置工具端电源输出。

```
int Set_Tool_Voltage(SOCKHANDLE ArmSocket, byte type, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）type

电源输出类型，0-0V，1-5V，2-12V，3-24V

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.17.5. 获取工具端电源输出 Get_Tool_Voltage

该函数用于获取工具端电源输出。

```
int Get_Tool_Voltage(SOCKHANDLE ArmSocket, byte *voltage);
```

参数：

（1）ArmSocket

Socket 句柄

（2）voltage

读取回来的电源输出类型：0-0V，1-5V，2-12V，3-24V

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18. 末端手爪控制（选配）

睿尔曼机械臂末端配备了因时机器人公司的 EG2-4C2 手爪，为了便于用户操作手爪，机械臂控制器对用户开放了手爪的 API 函数（手爪控制 API 与末端 modbus 功能互斥）。

5.18.1. 配置手爪的开口度 Set_Gripper_Route

该函数用于配置手爪的开口度。

```
int Set_Gripper_Route(SOCKHANDLE ArmSocket, int min_limit, int max_limit, bool
```

block);

参数：

（1）ArmSocket

Socket 句柄

（2）min

手爪开口最小值，范围：0~1000，无单位量纲

（3）max

手爪开口最大值，范围：0~1000，无单位量纲

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18.2. 设置夹爪松开到最大位置 Set_Gripper_Release

该函数用于控制手爪以指定速度张开到最大开口处

int Set_Gripper_Release(SOCKHANDLE ArmSocket, int speed, bool block, int timeout);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

手爪松开速度 ，范围 1~1000，无单位量纲

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

（4）timeout

设置返回超时时间，阻塞模式生效，单位：秒

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18.3. 设置夹爪夹取 Set_Gripper_Pick

该函数用于控制手爪以设定的速度去夹取，当手爪所受力矩大于设定的力矩阈值时，停止运动。

int Set_Gripper_Pick(SOCKHANDLE ArmSocket, int speed, int force, bool block, int timeout);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

手爪夹取速度 ，范围：1~1000，无单位量纲

（3）force

手爪夹取力矩阈值，范围 ：50~1000，无单位量纲

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

（5）timeout

设置返回超时时间，阻塞模式生效，单位：秒

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18.4. 设置夹爪持续夹取 Set_Gripper_Pick_On

该函数用于控制手爪以设定的速度去持续夹取，当手爪所受力矩大于设定的力矩阈值时，停止运动。之后当手爪所受力矩小于设定力矩后，手爪继续持续夹取，直到再次手爪所受力矩大于设定的力矩阈值时，停止运动。

int Set_Gripper_Pick_On(SOCKHANDLE ArmSocket, int speed, int force, bool block, int timeout);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

手爪夹取速度 ，范围：1~1000，无单位量纲

（3）force

手爪夹取力矩阈值，范围 ：50~1000，无单位量纲

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

（5）timeout

设置返回超时时间，阻塞模式生效，单位：秒

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18.5. 设置夹爪到指定开口位置 Set_Gripper_Position

该函数用于控制手爪到达指定开口度位置

int Set_Gripper_Position(SOCKHANDLE ArmSocket, int position, bool block, int timeout);

参数：

（1）ArmSocket

Socket 句柄

（2）position

手爪指定开口度，范围 ：1~1000，无单位量纲

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

（4）timeout

设置返回超时时间，阻塞模式生效，单位：秒

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.18.6. 获取夹爪状态 Get_Gripper_State

该函数用于获取夹爪状态。

int Get_Gripper_State(SOCKHANDLE ArmSocket, GripperState* gripper_state);

参数：

（1）ArmSocket

Socket 句柄

（2）gripper_state

夹爪状态

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：此接口需升级夹爪最新固件方可使用。

5.19. 拖动示教及轨迹复现

睿尔曼机械臂采用关节电流环实现拖动示教，拖动示教及轨迹复现的配置函数如下所示。

5.19.1. 进入拖动示教模式 Start_Drag_Teach

该函数用于控制机械臂进入拖动示教模式

```
int Start_Drag_Teach(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.2. 退出拖动示教模式 Stop_Drag_Teach

该函数用于控制机械臂退出拖动示教模式

int Stop_Drag_Teach(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.3. 拖动示教轨迹复现 Run_Drag_Trajectory

该函数用于控制机械臂复现拖动示教的轨迹，必须在拖动示教结束后才能使用，同时保证机械臂位于拖动示教的起点位置。若当前位置没有位于轨迹复现起点，请先调用 5.19.7，否则会返回报错信息。

int Run_Drag_Trajectory(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.4. 拖动示教轨迹复现暂停 Pause_Drag_Trajectory

该函数用于控制机械臂在轨迹复现过程中的暂停。

```
int Pause_Drag_Trajectory(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.5. 拖动示教轨迹复现继续 Continue_Drag_Trajectory

该函数用于控制机械臂在轨迹复现过程中暂停之后的继续，轨迹继续时，必须保证机械臂位于暂停时的位置，否则会报错，用户只能从开始位置重新复现轨迹。

```
int Continue_Drag_Trajectory(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待

控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.6. 拖动示教轨迹复现停止 Stop_Drag_Trajectory

该函数用于控制机械臂在轨迹复现过程中停止，停止后，不可继续。若要再次轨迹复现，只能从第一个轨迹点开始。

```
int Stop_Drag_Trajectory(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.7. 运动到轨迹起点 Drag_Trajectory_Origin

该函数用于机械臂运动到拖动示教轨迹起点。轨迹复现前，必须控制机械臂运动到轨迹起点，如果设置正确，机械臂将以 20%的速度运动到轨迹起点

```
int Drag_Trajectory_Origin(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.8. 复合模式拖动示教 Start_Multi_Drag_Teach

该函数用于复合模式拖动示教

int Start_Multi_Drag_Teach(SOCKHANDLE ArmSocket, int mode,int singular_wall,bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）mode

拖动示教模式 0-电流环模式，1-使用末端六维力，只动位置，2-使用末端六维力 ，只动姿态，3-使用末端六维力，位置和姿态同时动

（3）singular_wall

仅在六维力模式拖动示教中生效，用于指定是否开启拖动奇异墙，0 表示关闭拖动奇异墙，1 表示开启拖动奇异墙

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.9. 保存拖动示教轨迹 Save_Trajectory

该函数用于保存拖动示教轨迹。

int Save_Trajectory(SOCKHANDLE ArmSocket, char * filename, int* num, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）filename

轨迹要保存路径及名称，例: c:/rm_test.txt

（3）num

轨迹点数

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.10. 设置力位混合控制 Set_Force_Postion

该函数用于设置力位混合控制。在笛卡尔空间轨迹规划时，使用该功能可保证机械臂末端接触力恒定，使用时力的方向与机械臂运动方向不能在同一方向。开启力位混合控制，执行笛卡尔空间运动，接收到运动完成反馈后，需要等待 2S 后继续下发下一条运动指令。

注意：在进行力的操作之前，如果未进行力数据标定，可使用清空一维力、六维力数据接口对零位进行标定。

int Set_Force_Postion(SOCKHANDLE ArmSocket, int sensor, int mode, int direction, int N, bool block);

参数：

（1）ArmSocket

socket 句柄

（2）sensor

0-一维力；1-六维力

（3）mode

0-基坐标系力控；1-工具坐标系力控

（4）direction

力控方向；0-沿 X 轴；1-沿 Y 轴；2-沿 Z 轴；3-沿 RX 姿态方向；4-沿 RY 姿态方向；5-沿 RZ 姿态方向

（5）N

力的大小，单位 N，精确到 0.1N

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.19.11. 结束力位混合控制 Stop_Force_Postion

该函数用于结束力位混合控制

int Stop_Force_Postion(SOCKHANDLE ArmSocket, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.20. 末端六维力传感器的使用（选配）

睿尔曼 RM-65F 机械臂末端配备集成式六维力传感器，无需外部走线，用户可直接通过 API 对六维力进行操作，获取六维力数据。

5.20.1. 获取六维力数据 Get_Force_Data

该函数用于获取当前六维力传感器得到的力和力矩信息，若要周期获取力数据，周期不能小于 50ms。

int Get_Force_Data(SOCKHANDLE ArmSocket, float *Force, float *zero_force, float *work_zero, float *tool_zero);

参数：

（1）ArmSocket

Socket 句柄

（2）Force

返回的力和力矩数组地址，数组 6 个元素，依次为 Fx，Fy，Fz， Mx，My， Mz。其中，力的单位为 N；力矩单位为 Nm。

（3）zero_force

系统受到的外力数据

（4）work_zero

当前工作坐标系下系统受到的外力数据

（5）tool_zero

当前工具坐标系下系统受到的外力数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.20.2. 清空六维力数据 Clear_Force_Data

该函数用于空六维力数据，即后续获得的所有数据都是基于当前数据的偏移量。

```
int Clear_Force_Data(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.20.3. 设置六维力重心参数 Set_Force_Sensor

设置六维力重心参数，六维力重新安装后，必须重新计算六维力所收到的初始力和重心。分别在不同姿态下，获取六维力的数据，用于计算重心位置。该指令下发后，机械臂以 20%的速度运动到各标定点，该过程不可中断，中断后必须重新标定。

重要说明：必须保证在机械臂静止状态下标定。

```
int Set_Force_Sensor(SOCKHANDLE ArmSocket);
```

参数：

（1）ArmSocket

Socket 句柄

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.20.4. 手动标定六维力数据 Manual_Set_Force

该手动标定流程，适用于空间狭窄工作区域，以防自动标定过程中机械臂发生碰撞，用户手动标定六维力时，需要选择四个点位的数据，连续调用函数四次，机械臂开始自动沿用户设置的目标运动，并在此过程中计算六维力重心。

```
int Manual_Set_Force(SOCKHANDLE ArmSocket, int type,const float* joint);
```

参数：

（1）ArmSocket

Socket 句柄

（2）type

点位，依次调用四次发送 1~4；

（3）joint

关节角度，单位：°

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.20.5. 退出标定流程 Stop_Set_Force_Sensor

在标定六/一维力过程中，如果发生意外，发送该指令，停止机械臂运动，退出标定流程

```
int Stop_Set_Force_Sensor(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.21. 末端五指灵巧手控制（选配）

睿尔曼 RM-65 机械臂末端配备了五指灵巧手，可通过 API 对灵巧手进行设置。

5.21.1. 设置灵巧手手势序号 Set_Hand_Posture

设置灵巧手手势序号，设置成功后，灵巧手按照预先保存在 Flash 中的手势运动。

int Set_Hand_Posture(SOCKHANDLE ArmSocket, int posture_num, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）posture_num

预先保存在灵巧手内的手势序号，范围：1~40

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.21.2. 设置灵巧手动作序列序号 Set_Hand_Seq

设置灵巧手动作序列序号，设置成功后，灵巧手按照预先保存在 Flash 中的动作序列运动。

int Set_Hand_Seq(SOCKHANDLE ArmSocket, int seq_num, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）seq_num

预先保存在灵巧手内的动作序列序号，范围：1~40

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待

控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.21.3. 设置灵巧手角度 Set_Hand_Angle

设置灵巧手角度，灵巧手有 6 个自由度，从 1~6 分别为小拇指，无名指，中指，食指，大拇指弯曲，大拇指旋转。

```
int Set_Hand_Angle(SOCKHANDLE ArmSocket, const int *angle, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）angle

手指角度数组，6 个元素分别代表 6 个自由度的角度。范围：0~1000。另外，-1 代表该自由度不执行任何操作，保持当前状态

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.21.4. 设置灵巧手各关节速度 Set_Hand_Speed

设置灵巧手各关节速度

```
int Set_Hand_Speed(SOCKHANDLE ArmSocket, int speed, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）speed

灵巧手各关节速度设置，范围：1~1000

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.21.5. 设置灵巧手各关节力阈值 Set_Hand_Force

设置灵巧手各关节力阈值

```
int Set_Hand_Force(SOCKHANDLE ArmSocket, int force, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）force

灵巧手各关节力阈值设置，范围：1~1000，代表各关节的力矩阈值（四指握力 0~10N，拇指握力 0~15N）。

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.22. 末端传感器-一维力（选配）

睿尔曼机械臂末端接口板集成了一维力传感器，可获取 Z 方向的力，量程 200N，准度 0.5%FS。

5.22.1. 查询一维力数据 Get_Fz

该函数用于查询末端一维力数据。

```
int Get_Fz(SOCKHANDLE ArmSocket, float *Fz);
```

参数

（1）ArmSocket

Socket 句柄

（2）Fz

反馈的一维力原始数据 单位：N

（3）zero_force

系统受到的外力数据

（4）work_zero

当前工作坐标系下系统受到的外力数据

（5）tool_zero

当前工具坐标系下系统受到的外力数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注

第一帧指令下发后，开始更新一维力数据，此时返回的数据有滞后性；请从第二帧的数据开始使用。若周期查询 Fz 数据，频率不能高于 40Hz。

5.22.2. 清空一维力数据 Clear_Fz

该函数用于清空末端一维力数据。清空一维力数据后，后续所有获取到的数据都是基于当前的偏置。

```
int Clear_Fz(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.22.3. 自动标定末端一维力数据 Auto_Set_Fz

该函数用于自动标定末端一维力数据。一维力重新安装后，必须重新计算一维力所受到的初始力和重心。分别在不同姿态下，获取一维力的数据，用于计算

重心位置，该步骤对于基于一维力的力位混合控制操作具有重要意义。

int Auto_Set_Fz(SOCKHANDLE ArmSocket);

参数：

（1）ArmSocket

Socket 句柄

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.22.4. 手动标定末端一维力数据 Manual_Set_Fz

该函数用于手动标定末端一维力数据。一维力重新安装后，必须重新计算一维力所受到的初始力和重心。该手动标定流程，适用于空间狭窄工作区域，以防自动标定过程中机械臂发生碰撞，用户可以手动选取 2 个位姿下发，当下发完后，机械臂开始自动沿用户设置的目标运动，并在此过程中计算一维力重心。

int Manual_Set_Fz(SOCKHANDLE ArmSocket, const float* joint1,const float* joint2);

参数：

（1）ArmSocket

Socket 句柄

（2）joint

点位 1 关节角度

（3）joint2

点位 2 关节角度

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23. Modbus 配置

睿尔曼机械臂在控制器的航插和末端接口板航插处，各有 1 路 RS485 通讯接口，这两个 RS485 端口可通过接口配置为标准的 ModbusRTU 模式。然后通过接口对端口连接的外设进行读写操作。

注意：控制器的 RS485 接口在未配置为 Modbus RTU 模式的情况下，可用于用户对机械臂进行控制，这两种模式不可兼容。若要恢复机械臂控制模式，必须将该端口的 Modbus RTU 模式关闭。Modbus RTU 模式关闭后，系统会自动切换回机械臂控制模式，波特率 460800BPS，停止位 1，数据位 8，无检验。

同时，I 系列控制器支持 modbus-TCP 主站配置，可配置使用 modbus-TCP 主站，用于连接外部设备的 modbus-TCP 从站。

5.23.1. 设置通讯端口 Modbus RTU 模式 Set_Modbus_Mode

该函数用于配置通讯端口 Modbus RTU 模式。机械臂启动后，要对通讯端口进行任何操作，必须先启动该指令，否则会返回报错信息。

另外，机械臂会对用户的配置方式进行保存，机械臂重启后会自动恢复到用户断电之前配置的模式。

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

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口为 RTU 主站，1-末端接口板 RS485

接口为 RTU 主站，2-控制器 RS485 端口为 RTU 从站

（3）baudrate

波特率，支持 9600，115200，460800 三种常见波特率

（4）timeout

超时时间，单位百毫秒。对 Modbus 设备所有的读写指令，在规定的超时时间内未返回响应数据，则返回超时报错提醒。超时时间若设置为 0，则机械臂按 1 进行配置。

（5）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.2. 关闭通讯端口 Modbus RTU 模式 Close_Modbus_Mode

该函数用于关闭通讯端口 Modbus RTU 模式。

```
int Close_Modbus_Mode(SOCKHANDLE ArmSocket, int port, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口为 RTU 主站，1-末端接口板 RS485 接口为 RTU 主站，2-控制器 RS485 端口为 RTU 从站

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.3. 配置连接 ModbusTCP 从站 Set_Modbustcp_Mode--I 系列

该函数用于配置连接 ModbusTCP 从站。

int Set_Modbustcp_Mode(SOCKHANDLE ArmSocket, const char* ip, int port, int timeout);

参数：

（1）ArmSocket

Socket 句柄

（2）ip

从机 IP 地址

（3）port

端口号

（4）timeout

超时时间，单位秒。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.4. 配置关闭 ModbusTCP 从站 Close_Modbustcp_Mode--I 系列

该函数用于配置关闭 ModbusTCP 从站。

int Close_Modbustcp_Mode(SOCKHANDLE ArmSocket);

参数：

（1）ArmSocket

Socket 句柄

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.5. 读线圈 Get_Read_Coils

该函数用于读线圈。

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

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

线圈起始地址

（4）num

要读的线圈的数量，该指令最多一次性支持读 8 个线圈数据，即返回的数据不会超过一个字节

（5）device

外设设备地址

（6）coils_data

返回线圈数量

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.6. 读离散量输入 Get_Read_Input_Status

该函数用于读离散量输入。

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

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

数据起始地址

（4）num

要读的数据的数量，该指令最多一次性支持读 8 个离散量数据，即返回的数据不会超过一个字节

（5）device

外设设备地址

（6）coils_data

返回离散量

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.7. 读保持寄存器 Get_Read_Holding_Registers

该函数用于读保持寄存器。该函数每次只能读 1 个寄存器，即 2 个字节的数据，不可一次性读取多个寄存器数据

int Get_Read_Holding_Registers(SOCKHANDLE ArmSocket, int port,int address,int device,int* coils_data);

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

数据起始地址

（4）device

外设设备地址

（5）coils_data

返回寄存器数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.8. 读输入寄存器 Get_Read_Input_Registers

该函数用于读输入寄存器。

int Get_Read_Input_Registers(SOCKHANDLE ArmSocket, int port,int address,int device,int* coils_data);

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

数据起始地址

（4）device

外设设备地址

（5）coils_data

返回寄存器数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.9. 写单圈数据 Write_Single_Coil

该函数用于写单圈数据。

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

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

线圈起始地址

（4）data

要写入线圈的数据

（5）device

外设设备地址

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.10. 写单个寄存器 Write_Single_Register

该函数用于写单个寄存器。

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

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

寄存器起始地址。

（4）data

要写入寄存器的数据。

（5）device

外设设备地址

（6）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.11. 写多个寄存器 Write_Registers

该函数用于写多个寄存器。

```
int Write_Registers(SOCKHANDLE ArmSocket, int port,int address,int num,byte *single_data, int device, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

寄存器起始地址

（4）num

写寄存器个数，寄存器每次写的数量不超过 10 个

（5）single_data

要写入寄存器的数据数组，类型：byte

（6）device

外设设备地址

（7）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.12. 写多圈数据 Write_Coils

该函数用于写多圈数据

```
int Write_Coils(SOCKHANDLE ArmSocket, int port,int address,int num, byte *coils_data,int device,bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

线圈起始地址。

（4）num

写线圈个数，每次写的数量不超过 160 个

（5）coils_data

要写入线圈的数据数组，类型：byte。若线圈个数不大于 8，则写入的数据为 1 个字节；否则，则为多个数据的数组。

（6）device

外设设备地址

（7）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.13. 读多圈数据 Get_Read_Multiple_Coils

该函数用于读取多圈数据。

int Get_Read_Multiple_Coils(SOCKHANDLE ArmSocket, int port,int address,int num,int device,int *coils_data);

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

线圈起始地址

（4）num

8< num <= 120 要读的线圈的数量，该指令最多一次性支持读 120 个线圈数据， 即 15 个 byte

（5）device

外设设备地址

（6）coils_data

返回线圈状态

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.14. 读多个保持寄存器 Read_Multiple_Holding_Registers

该函数用于读多个保持寄存器。

int Read_Multiple_Holding_Registers(SOCKHANDLE ArmSocket, byte port, int address,byte num, int device, int8_t *coils_data);

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

寄存器起始地址

（4）num

2 < num < 13 要读的寄存器的数量，该指令最多一次性支持读 12 个寄存器数据， 即 24 个 byte

（5）device

外设设备地址

（6）coils_data

返回寄存器数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.23.15. 读多个输入寄存器 Read_Multiple_Input_Registers

该函数用于读多个保持寄存器。

int Read_Multiple_Input_Registers(SOCKHANDLE ArmSocket, int port, int address,byte num, int device, int *coils_data);

参数：

（1）ArmSocket

Socket 句柄

（2）port

通讯端口，0-控制器 RS485 端口，1-末端接口板 RS485 接口，3-控制器 ModbusTCP 设备

（3）address

寄存器起始地址

（4）num

2 < num < 13 要读的寄存器的数量，该指令最多一次性支持读 12 个寄存器数据， 即 24 个 byte

（5）device

外设设备地址

（6）coils_data

返回寄存器数据

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.24. 升降机构

睿尔曼机械臂可集成自主研发升降机构。

5.24.1. 升降机构速度开环控制 Set_Lift_Speed

该函数用于升降机构速度开环控制。

int Set_Lift_Speed(SOCKHANDLE ArmSocket, int speed);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

升降机速度百分比，-100 ~100

speed<0：升降机构向下运动

speed>0：升降机构向上运动

Speed=0: 升降机构停止运动

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.24.2. 设置升降机构高度 Set_Lift_Height

该函数用于设置升降机构高度。

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

参数：

（1）ArmSocket

Socket 句柄

（2）height

目标高度，单位 mm，范围：0~2600

（3）speed

升降机速度百分比，1~100

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.24.3. 获取升降机构状态 Get_Lift_State

该函数用于获取升降机构状态。

int Get_Lift_State(SOCKHANDLE ArmSocket, int* height,int* current,int* err_flag, int *mode);

参数：

（1）ArmSocket

Socket 句柄

（2）height

当前升降机构高度，单位：mm，精度：1mm，范围：0~2300

（3）current

当前升降驱动电流，单位：mA，精度：1mA

（4）err

升降驱动错误代码，错误代码类型参考关节错误代码

（5）mode

当前升降状态，0-空闲，1-正方向速度运动，2-正方向位置运动，3-负方向速度运动，4-负方向位置运动

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.25. 透传力位混合控制补偿

针对睿尔曼带一维力和六维力版本的机械臂，用户除了可直接使用示教器调用底层的力位混合控制模块外，还可以将自定义的轨迹以周期性透传的形式结合底层的力位混合控制算法进行补偿。

在进行力的操作之前，如果未进行力数据标定，可使用清空一维力、六维力数据接口对零位进行标定。

5.25.1. 开启透传力位混合控制补偿模式 Start_Force_Position_Move

该函数用于开启透传力位混合控制补偿模式。在下发透传轨迹前必须下发该指令开启功能

```
int Start_Force_Position_Move(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.25.2. 力位混合控制补偿透传模式（关节角度）Force_Position_Move_Joint

该函数用于力位混合控制补偿透传模式（关节角度）。

```
int Force_Position_Move_Joint(SOCKHANDLE ArmSocket, const float *joint,byte
```

sensor,byte mode,int dir,float force, bool follow);

参数：

（1）ArmSocket

Socket 句柄

（2）joint

目标关节角度 单位 °

（3）sensor

所使用传感器类型，0-一维力，1-六维力

（4）mode

模式，0-沿基坐标系，1-沿工具端坐标系

（5）dir

力控方向，0~5 分别代表 X/Y/Z/Rx/Ry/Rz，其中一维力类型时默认方向为 Z 方向

（6）force

力的大小 单位 0.1N

（7）follow

是否高跟随

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注 1：该功能只适用于一维力传感器和六维力传感器机械臂版本

备注 2：透传周期越快，力位混合控制效果越好。基础系列 WIFI 和网口模式透传周期最快 20ms，USB 和 RS485 模式透传周期最快 10ms。高速网口的透传

周期最快也可到 10ms，不过在使用该高速网口前，需要使用指令打开配置。另外 I 系列有线网口周期最快可达 5ms。

5.25.3. 力位混合控制补偿透传模式（位姿）Force_Position_Move_Pose

该函数用于力位混合控制补偿透传模式（位姿）

int Force_Position_Move_Pose(SOCKHANDLE ArmSocket, Pose pose,byte sensor,byte mode,int dir,float force, bool follow);

参数：

（1）ArmSocket

Socket 句柄

（2）pose

当前坐标系下目标位姿，位姿中包括姿态欧拉角和姿态四元数，四元数合理情况下，优先使用姿态四元数

（3）sensor

所使用传感器类型，0-一维力，1-六维力

（4）mode

模式，0-沿基坐标系，1-沿工具端坐标系

（5）dir

力控方向，0~5 分别代表 X/Y/Z/Rx/Ry/Rz，其中一维力类型时默认方向为 Z 方向

（6）force

力的大小 单位 0.1N

（7）follow

是否高跟随

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

备注：

1、该功能只适用于一维力传感器和六维力传感器机械臂版本

2、透传周期越快，力位混合控制效果越好。基础系列 WIFI 和网口模式透传周期最快 20ms，USB 和 RS485 模式透传周期最快 10ms。高速网口的透传周期最快也可到 10ms，不过在使用该高速网口前，需要使用指令打开配置。另外 I 系列有线网口周期最快可达 5ms。

3、透传开始的起点务必为机械臂当前位姿，否则可能会力控补偿失败或机械臂无法运动

5.25.4. 关闭透传力位混合控制补偿模式 Stop_Force_Position_Move

该函数用于关闭透传力位混合控制补偿模式。

```
int Stop_Force_Position_Move(SOCKHANDLE ArmSocket, bool block);
```

参数：

（1）ArmSocket

Socket 句柄

（2）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.26. 算法工具接口

针对睿尔曼机械臂，提供正解、逆解等工具接口。

算法接口可单独使用，也可连接机械臂使用。连接机械臂使用时，为保证算法所用的参数是机械臂当前的数据，需调用获取工作、工具坐标系，获取安装角度等接口同步信息，否则算法将使用 API 与机械臂创建连接时机械臂的坐标系、安装角度等数据计算。

5.26.1. 初始化算法依赖数据 Algo_Init_Sys_Data

初始化算法依赖数据（不连接机械臂时调用， 连接机械臂会自动调用)

```
void Algo_Init_Sys_Data(RobotType dMode, SensorType bType);
```

参数：

（1）dMode

机械臂型号

（2）bType

传感器型号

5.26.2. 设置算法的安装角度 Algo_Set_Angle

设置算法的安装角度参数。

```
void Algo_Set_Angle(float x, float y, float z);
```

参数：

（1）x

X 轴安装角度，单位：°

（2）y

Y 轴安装角度，单位：°

（3）z

Z 轴安装角度，单位：°

5.26.3. 获取算法的安装角度 Algo_Get_Angle

获取算法的安装角度参数。

```
Algo_Get_Angle(float* x, float* y, float* z);
```

参数：

（1）x

X 轴安装角度，单位：°

（2）y

Y 轴安装角度，单位：°

（3）z

Z 轴安装角度，单位度

5.26.4. 设置算法工作坐标系 Algo_Set_WorkFrame

设置算法工作坐标系

```
Algo_Set_WorkFrame(const FRAME* const coord_work);
```

参数：

（1）coord_work

工作坐标系数据

5.26.5. 获取算法当前工作坐标系 Algo_Get_Curr_WorkFrame

获取算法当前工作坐标系。

```
Algo_Get_Curr_WorkFrame(FRAME* coord_work);
```

返回值：

当前工作坐标系

5.26.6. 设置算法工具坐标系 Algo_Set_ToolFrame

设置算法工具坐标系和负载

```
Algo_Set_ToolFrame(const FRAME* const coord_tool);
```

参数：

（1）coord_tool

工具坐标系数据

5.26.7. 获取算法当前工具坐标系 Algo_Get_Curr_ToolFrame

获取算法当前工具坐标系。

```
Algo_Get_Curr_ToolFrame(FRAME* coord_tool);
```

返回值：

当前工具坐标系

5.26.8. 正解 Algo_Forward_Kinematics

用于睿尔曼机械臂正解计算。

```
Pose Algo_Forward_Kinematics(const float* const joint);
```

参数：

（1）joint

关节 1 到关节 7 角度，单位：°

返回值：

正解结果

5.26.9. 逆解 Algo_Inverse_Kinematics

用于睿尔曼机械臂逆解计算。

int Algo_Inverse_Kinematics(const float* const q_in, const Pose* const q_pose,float* q_out, uint8_t flag);

参数：

（1）q_in

上一时刻关节角，单位：°

（2）q_pose

目标位姿。

（3）q_out

输出的关节角度，单位：°

（4）flag

姿态参数类别：0-四元数；1-欧拉角

返回值：

SYS_NORMAL：计算正常，CALCULATION_FAILED：计算失败。

5.26.10. 逆解 Algo_Inverse_Kinematics_Wrap

用于睿尔曼机械臂逆解计算。

int Algo_Inverse_Kinematics_Wrap(const IK_Params* params);

参数：

（1）params

逆解输入输出参数。

返回值：

SYS_NORMAL：计算正常，CALCULATION_FAILED：计算失败。

5.26.11. 计算平移、旋转运动位姿 Algo_PoseMove

用于计算 Pos 和 Rot 沿某坐标系有一定的位移和旋转角度后，所得到的位姿数据。

Pose Algo_PoseMove(Pose poseCurrent, const float *deltaPosAndRot, int frameMode);

参数：

（1）poseCurrent

当前位姿，输入 position 和 euler。

（2）deltaPosAndRot

沿轴位移和绕轴旋转数组（dx，dy，dz，rotx, roty, rotz），位置移动单位：m，旋转单位：度。

（3）frameMode

坐标系模式选择

0：计算相对于工作坐标系平移、旋转后的位姿，当工作坐标系为 0 时，即为计算相对基坐标系的位姿；

1：计算相对于工具坐标系平移、旋转后的位姿。

返回值：

经平移、旋转后的位姿。

5.26.12. 计算环绕运动位姿 Algo_RotateMove

用于计算环绕运动位姿。

Pose Algo_RotateMove(const float* const curr_joint, int rotate_axis,float rotate_angle, Pose choose_axis);

参数：

（1）curr_joint

当前关节角度，单位：°

（2）rotate_axis

旋转轴：1：x 轴， 2：y 轴，3：z 轴

（3）rotate_angle

旋转角度，单位：°

（4）choose_axis

指定计算时使用的坐标系

返回值：

计算位姿结果。

5.26.13. 末端位姿转成工具位姿 Algo_End2Tool

末端位姿转成工具位姿，即为：机械臂末端在基坐标系下的位姿，转化成工具坐标系末端在工作坐标系下的位姿。

```
Pose Algo_End2Tool(Pose eu_end);
```

参数：

（1）eu_end

基于世界坐标系和默认工具坐标系的末端位姿

返回值：

基于工作坐标系和工具坐标系的末端位姿

5.26.14. 工具位姿转末端位姿 Algo_Tool2End

工具位姿转末端位姿，即为：工具坐标系末端在工作坐标系下的位姿，转换

成机械臂末端在基坐标系下的位姿。

Pose Algo_Tool2End(Pose eu_tool);

参数：

（1）eu_tool

基于工作坐标系和工具坐标系的末端位姿

返回值：

基于世界坐标系和默认工具坐标系的末端位姿

5.26.15. 四元数转欧拉角 Algo_Quaternion2Euler

四元数转欧拉角 。

Euler Algo_Quaternion2Euler(Quat qua);

参数：

（1）qua

四元数

返回值：

欧拉角

5.26.16. 欧拉角转四元数 Algo_Euler2Quaternion

欧拉角转四元数。

Quat Algo_Euler2Quaternion(Euler eu);

参数：

（1）eu

欧拉角

返回值：

四元数

5.26.17. 欧拉角转旋转矩阵 Algo_Euler2Matrix

欧拉角 rx,ry,rz 转换成旋转矩阵（3*3）。

Matrix Algo_Euler2Matrix(Euler _rot3);

参数：

（1）_rot3

输入欧拉角

返回值：

（3*3）旋转矩阵。

5.26.18. 位姿转旋转矩阵 Algo_Pos2Matrix

位姿转旋转矩阵。

Matrix Algo_Pos2Matrix(Pose _point);

参数：

（1）_point

位姿（x、y、z、rx、ry、rz)

返回值：

Matrix (4*4)。

5.26.19. 旋转矩阵转位姿 Algo_Matrix2Pos

旋转矩阵转位姿。

Pose Algo_Matrix2Pos(Matrix _matPos);

参数：

（1）matrix

旋转矩阵

返回值：

位姿。

5.26.20. 基坐标系转工作坐标系 Algo_Base2WorkFrame

基坐标系转工作坐标系

```
Pose Algo_Base2WorkFrame(Matrix matWork2Base, Pose poseBase);
```

参数：

（1）matWork2Base

工作坐标系在基坐标系下的矩阵

（2）poseBase

工具端坐标在基坐标系下位姿

返回值：

工作坐标系下的位姿。

5.26.21. 工作坐标系转基坐标系 Algo_WorkFrame2Base

工作坐标系转基坐标系

```
Pose Algo_WorkFrame2Base(Matrix matWork2Base, Pose poseWork);
```

参数：

（1）matWork2Base

工作坐标系在基坐标系下的矩阵

（2）poseWork

工具端坐标在工作坐标系下位姿

返回值：

基坐标系下的位姿。

5.26.22. 计算沿工具坐标系运动位姿 Algo_Cartesian_Tool

计算沿工具坐标系运动位姿。

Pose Algo_Cartesian_Tool(const float* const curr_joint, float move_lengthx,float move_lengthy, float move_lengthz);

参数：

（1）curr_joint

当前关节角度，单位度

（2）move_lengthx

沿 X 轴移动长度，单位米

（3）move_lengthy

沿 Y 轴移动长度，单位米

（4）move_lengthz

沿 Z 轴移动长度，单位米

返回值：

基坐标系下的位姿。

5.26.23. 设置算法关节最大限位 Algo_Set_Joint_Max_Limit

设置算法关节最大限位

Algo_Set_Joint_Max_Limit(const float* const joint_limit);

参数：

（1）joint_limit

关节的最大限位数组

5.26.24. 获取算法关节最大限位 Algo_Get_Joint_Max_Limit

获取算法关节最大限位。

Algo_Get_Joint_Max_Limit(float* joint_limit);

返回值：

关节最大限位数组

5.26.25. 设置算法关节最小限位 Algo_Set_Joint_Min_Limit

设置算法关节最小限位

Algo_Set_Joint_Min_Limit(const float* const joint_limit);

参数：

（1）joint_limit

关节的最小限位数组

5.26.26. 获取算法关节最小限位 Algo_Get_Joint_Min_Limit

获取算法关节最小限位。

Algo_Get_Joint_Min_Limit(float* joint_limit);

返回值：

关节最小限位数组

5.26.27. 设置算法关节最大速度 Algo_Set_Joint_Max_Speed

设置算法关节最大速度

Algo_Set_Joint_Max_Speed(const float* const joint_slim_max);

参数：

（1）joint_slim_max

关节的最大速度数组

5.26.28. 获取算法关节最大速度 Algo_Get_Joint_Max_Speed

获取算法关节最大速度。

Algo_Get_Joint_Max_Speed(float* joint_slim_max);

返回值：

关节最大速度数组

5.26.29. 设置算法关节最大加速度 Algo_Set_Joint_Max_Acc

设置算法关节最大加速度

Algo_Set_Joint_Max_Acc(const float* const joint_alim_max);

参数：

（1）joint_alim_max

关节的最大加速度数组

5.26.30. 获取算法关节最大加速度 Get_Joint_Max_Acc

获取算法关节最大加速度。

Algo_Get_Joint_Max_Acc(float* joint_alim_max);

返回值：

关节最大加速度数组

5.27. 在线编程

5.27.1. 文件下发 Send_TrajectoryFile

在线编程文件下发。

int Send_TrajectoryFile(SOCKHANDLE ArmSocket, Send_Project_Params params, int * err_line);

参数：

（1）ArmSocket

socket 句柄

（2）params

文件下发参数

（3）err_line

下发失败时有问题的工程行数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.27.2. 轨迹规划中改变速度比例系数 Set_Plan_Speed

该函数用于轨迹规划中改变速度比例系数

int Set_Plan_Speed(SOCKHANDLE ArmSocket, int speed, bool block);

参数：

（1）ArmSocket

socket 句柄

（2）speed

当前进度条的速度数据

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.27.3. 文件树弹窗提醒 Popup

文件树弹窗提醒。本指令是控制器发送给示教器，返回值是示教器发送给控制器。

int Popup(SOCKHANDLE ArmSocket, int content, bool block);

参数：

（1）ArmSocket

socket 句柄

（2）content

弹窗提示指令所在文件树的位置

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.28. 机械臂状态主动上报

5.28.1. 设置主动上报配置 Set_Realtime_Push

该函数用于设置主动上报接口配置。

int Set_Realtime_Push(SOCKHANDLE ArmSocket, Realtime_Push_Config config);

（1）ArmSocket

socket 句柄

（2）config

主动上报接口配置

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.28.2. 获取主动上报配置 Get_Realtime_Push

该函数用于获取主动上报接口配置。

int Get_Realtime_Push(SOCKHANDLE ArmSocket, Realtime_Push_Config *config);

（1）ArmSocket

socket 句柄

（2）config

获取到的主动上报接口配置

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.28.3. 机械臂状态主动上报 Realtime_Arm_Joint_State

该函数使用 UDP 协议监听本机广播的端口号，接收机械臂状态广播数据。可注册回调函数来处理机械臂状态信息。

void Realtime_Arm_Joint_State(RobotStatusListener RobotStatuscallback);

（1）RobotStatuscallback

用于接收机械臂状态广播回调函数。

5.29. 通用扩展关节

5.29.1. 关节速度环控制 Expand_Set_Speed

该函数用于扩展关节速度环控制。

int Expand_Set_Speed(SOCKHANDLE ArmSocket, int speed, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）speed

-50 表示最大速度的百分之五十反方向运动

（3）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.29.2. 关节位置环控制 Expand_Set_Pos

该函数用于扩展关节位置环控制。

int Expand_Set_Pos(SOCKHANDLE ArmSocket, int pos, int speed, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）pos

升降关节精度 1mm 旋转关节精度 0.001°

（3）speed

50 表示最大速度的百分之五十,且速度必须大于 0

（4）block

RM_NONBLOCK-非阻塞，发送后立即返回； RM_BLOCK-阻塞，等待控制器返回设置成功指令。

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.29.3. 扩展关节状态获取 Expand_Get_State

该函数用于获取扩展关节状态。

int Expand_Get_State(SOCKHANDLE ArmSocket, int* pos, int* err_flag, int* current, int* mode);

参数：

（1）ArmSocket

Socket 句柄

（2）pos

当前升降机构高度，单位：mm，精度：1mm，如果是旋转关节则为角度 单位度，精度 0.001°

（3）err

升降驱动错误代码，错误代码类型参考关节错误代码

（4）current

当前升降驱动电流，单位：mA，精度：1mA

（5）mode

当前升降状态，0-空闲，1-正方向速度运动，2-正方向位置运动，3-负方向速度运动，4-负方向位置运动

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30. 在线编程存储列表（I 系列）

5.30.1. 获取在线编程轨迹列表 Get_Program_Trajectory_List

该函数用于查询在线编程轨迹列表。

int Get_Program_Trajectory_List(SOCKHANDLE ArmSocket, ProgramTrajectoryData *programlist);

参数：

（1）ArmSocket

Socket 句柄

（2）programlist

在线编程程序列表

page_num:页码（全部查询时此参数传 0）

page_size:每页大小（全部查询时此参数传 0）

vague_search:模糊搜索 （传递此参数可进行模糊查询）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.2. 查询在线编程程序运行状态 Get_Program_Run_State

该函数用于查询在线编程轨迹运行状态。

int Get_Program_Run_State(SOCKHANDLE ArmSocket, ProgramRunState* state);

参数：

（1）ArmSocket

Socket 句柄

（2）state

在线编程运行状态结构体

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.3. 开始运行指定编号轨迹 Set_Program_ID_Start

该函数用于开始运行指定编号轨迹。

int Set_Program_ID_Start(SOCKHANDLE ArmSocket, int id, int speed, bool block);

参数：

（1）ArmSocket

Socket 句柄

（2）id

运行指定的 ID，1-100，存在轨迹可运行

（3）speed

1-100，需要运行轨迹的速度，可不提供速度比例，按照存储的速度运行

（4）block

0-非阻塞，开始运行后返回；1-阻塞，等待在线编程程序运行结束返回

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.4. 删除指定编号轨迹 Delete_Program_Trajectory

该函数用于删除指定编号轨迹。

int Delete_Program_Trajectory(SOCKHANDLE ArmSocket, int id);

参数：

（1）ArmSocket

Socket 句柄

（2）id

删除指定的 ID 的轨迹

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.5. 修改指定编号轨迹的信息 Update_Program_Trajectory

该函数用于修改指定编号轨迹的信息。

int Update_Program_Trajectory(SOCKHANDLE ArmSocket, int id, int plan_speed, const char *project_name);

参数：

（1）ArmSocket

Socket 句柄

（2）id

指定在线编程轨迹编号

（3）speed

更新后的规划速度比例 1-100

（4）Project_name

更新后的文件名称（最大 10 个字节）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.6. 设置 IO 默认运行的在线编程文件编号 Set_Default_Run_Program

该函数用于设置 IO 默认运行的在线编程文件编号。

int Set_Default_Run_Program(SOCKHANDLE ArmSocket, int id);

参数：

（1）ArmSocket

Socket 句柄

（2）id

设置 IO 默认运行的在线编程文件编号，支持 0-100，0 代表取消设置

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.30.7. 获取 IO 默认运行的在线编程文件编号 Get_Default_Run_Program

该函数用于获取 IO 默认运行的在线编程文件编号。

int Get_Default_Run_Program(SOCKHANDLE ArmSocket, int *id);

参数：

（1）ArmSocket

Socket 句柄

（2）id

IO 默认运行的在线编程文件编号，支持 0-100，0 代表无默认

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.31. 全局路点（I 系列）

5.31.1. 新增全局路点 Add_Global_Waypoint

该函数用于新增全局路点。

int Add_Global_Waypoint(SOCKHANDLE ArmSocket, Waypoint waypoint);

参数：

（1）ArmSocket

Socket 句柄

（2）waypoint

新增全局路点参数（无需输入新增全局路点时间）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.31.2. 更新全局路点 Update_Global_Waypoint

该函数用于更新全局路点。

int Update_Global_Waypoint(SOCKHANDLE ArmSocket, Waypoint waypoint);

参数：

（1）ArmSocket

Socket 句柄

（2）waypoint

更新全局路点参数（无需输入更新全局路点时间）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.31.3. 删除全局路点 Delete_Global_Waypoint

该函数用于删除全局路点。

int Delete_Global_Waypoint(SOCKHANDLE ArmSocket, const char* name);

参数：

（1）ArmSocket

Socket 句柄

（2）name

全局路点名称

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.31.4. 查询多个全局路点 Get_Global_Point_List

该函数用于查询多个全局路点。

int Get_Global_Point_List(SOCKHANDLE ArmSocket, WaypointsList* point_list);

参数：

（1）ArmSocket

Socket 句柄

（2）point_list

全局路点列表查询结构体

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.31.5. 查询指定全局路点 Given_Global_Waypoint

该函数用于查询指定全局路点。

int Given_Global_Waypoint(SOCKHANDLE ArmSocket, const char *name, Waypoint* point);

参数：

（1）ArmSocket

Socket 句柄

（2）name

指定全局路点名称

（3）point

返回指定全局路点参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32. 电子围栏和虚拟墙（I 系列）

I 系列机械臂具备电子围栏与虚拟墙功能，并提供了针对控制器所保存的电子围栏或虚拟墙几何模型参数的操作接口。用户可以通过这些接口，实现对电子围栏或虚拟墙的新增、查询、更新和删除操作，在使用中，可以灵活的使用保存在控制器中的参数配置，需要注意的是，目前控制器支持保存的参数要求不超过 10 个。

电子围栏功能通过精确设置参数，确保机械臂的轨迹规划、示教等运动均在设定的电子围栏范围内进行。当机械臂的运动轨迹可能超出电子围栏的界限时，系统会立即返回相应的错误码，并自动中止运动，从而有效保障机械臂的安全运行。需要注意的是，电子围栏目前仅支持长方体和点面矢量平面这两种形状，并且其仅在仿真模式下生效，为用户提供一个预演轨迹与进行轨迹优化的安全环境。

虚拟墙功能支持在电流环拖动示教与力控拖动示教两种模式下，对拖动范围进行精确限制。在这两种特定的示教模式下，用户可以借助虚拟墙功能，确保机械臂的拖动操作不会超出预设的范围。但请务必注意，虚拟墙功能目前支持长方

体和球体两种形状，并仅在上述两种示教模式下有效。在其他操作模式下，此功能将自动失效。因此，请确保在正确的操作模式下使用虚拟墙功能，以充分发挥其限制拖动范围的作用。

5.32.1. 新增几何模型参数 Add_Electronic_Fence_Config

该函数用于新增几何模型参数，最多支持 10 个几何模型。

int Add_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);

参数：

（4）ArmSocket

Socket 句柄

（5）config

电子围栏参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.2. 更新几何模型参数 Update_Electronic_Fence_Config

该函数用于更新几何模型参数，最多支持 10 个几何模型。

int Update_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);

参数：

（1）ArmSocket

Socket 句柄

（2）config

电子围栏参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.3. 删除几何模型参数 Delete_Electronic_Fence_Config

该函数用于删除几何模型参数，最多支持 10 个几何模型。

int Delete_Electronic_Fence_Config(SOCKHANDLE ArmSocket, const char* name);

参数：

（1）ArmSocket

Socket 句柄

（2）name

指定电子围栏名称

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.4. 查询所有几何模型名称 Get_Electronic_Fence_List_Names

该函数用于查询所有几何模型名称，最多支持 10 个几何模型。

int Get_Electronic_Fence_List_Names(SOCKHANDLE ArmSocket, ElectronicFenceNames* names, int *len);

参数：

（1）ArmSocket

Socket 句柄

（2）names

电子围栏名称列表，长度为实际存在电子围栏

（3）len

电子围栏名称列表长度

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.5. 查询指定几何模型电子围栏参数 Given_Electronic_Fence_Config

该函数用于查询指定几何模型参数，最多支持 10 个几何模型。

int Given_Electronic_Fence_Config(SOCKHANDLE ArmSocket, const char *name, ElectronicFenceConfig* config);

参数：

（1）ArmSocket

Socket 句柄

（2）name

指定电子围栏名称

（3）config

返回电子围栏参数

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.6. 查询所有几何模型信息 Get_Electronic_Fence_List_Info

该函数用于查询所有几何模型信息，最多支持 10 个几何模型。

int Get_Electronic_Fence_List_Info(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config, int *len);

参数：

（1）ArmSocket

Socket 句柄

（2）config

电子围栏信息列表，长度为实际存在电子围栏

（3）len

电子围栏信息列表长度

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.7. 设置电子围栏使能状态 Set_Electronic_Fence_Enable

该函数用于设置电子围栏使能状态。

int Set_Electronic_Fence_Enable(SOCKHANDLE ArmSocket, bool enable_state, int in_out_side, int effective_region);

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

（3）in_out_side

0-机器人在电子围栏内部，1-机器人在电子围栏外部

（4）effective_region

0-针对整臂区域生效

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.8. 获取电子围栏使能状态 Get_Electronic_Fence_Enable

该函数用于获取电子围栏使能状态。

int Get_Electronic_Fence_Enable(SOCKHANDLE ArmSocket, bool* enable_state, int* in_out_side, int* effective_region);

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

（3）in_out_side

0-机器人在电子围栏内部，1-机器人在电子围栏外部

（4）effective_region

0-针对整臂区域生效

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.9. 设置当前电子围栏参数 Set_Electronic_Fence_Config

该函数用于设置当前电子围栏参数。

int Set_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);

参数：

（1）ArmSocket

Socket 句柄

（2）config

当前电子围栏参数（无需设置电子围栏名称）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.10. 获取当前电子围栏参数 Get_Electronic_Fence_Config

该函数用于获取当前电子围栏参数。

int Get_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config);

参数：

（1）ArmSocket

Socket 句柄

（2）config

当前电子围栏参数（返回参数中不包含电子围栏名称）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.11. 设置虚拟墙使能状态 Set_Virtual_Wall_Enable

该函数用于设置虚拟墙使能状态。

int Set_Virtual_Wall_Enable(SOCKHANDLE ArmSocket, bool enable_state, int in_out_side, int effective_region);

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

（3）in_out_side

0-机器人在虚拟墙内部

（4）effective_region

1-针对末端生效

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.12. 获取虚拟墙使能状态 Get_Virtual_Wall_Enable

该函数用于获取虚拟墙使能状态。

int Get_Virtual_Wall_Enable(SOCKHANDLE ArmSocket, bool* enable_state, int* in_out_side, int* effective_region);

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

（3）in_out_side

0-机器人在虚拟墙内部

（4）effective_region

1-针对末端生效

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.13. 设置当前虚拟墙参数 Set_Virtual_Wall_Config

该函数用于设置当前虚拟墙参数。

int Set_Virtual_Wall_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);

参数：

（1）ArmSocket

Socket 句柄

（2）config

当前虚拟墙参数（无需设置虚拟墙名称）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.32.14. 获取当前虚拟墙参数 Get_Virtual_Wall_Config

该函数用于获取当前虚拟墙参数。

int Get_Virtual_Wall_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config);

参数：

（1）ArmSocket

Socket 句柄

（2）config

当前虚拟墙参数（返回参数中不包含虚拟墙名称）

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.33. 自碰撞安全检测（I 系列）

I 系列机械臂支持自碰撞安全检测，自碰撞安全检测使能状态下，可确保在轨迹规划、示教等运动过程中机械臂的各个部分不会相互碰撞，需要注意的是，以上自碰撞安全检测功能目前只在仿真模式下生效，用于进行预演轨迹与轨迹优化。

5.33.1. 设置自碰撞安全检测使能状态 Set_Self_Collision_Enable

该函数用于设置自碰撞安全检测使能状态。

```
int Set_Self_Collision_Enable(SOCKHANDLE ArmSocket, bool enable_state);
```

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。

5.33.2. 获取自碰撞安全检测使能状态 Get_Self_Collision_Enable

该函数用于获取自碰撞安全检测使能状态。

```
int Get_Self_Collision_Enable(SOCKHANDLE ArmSocket, bool* enable_state);
```

参数：

（1）ArmSocket

Socket 句柄

（2）enable_state

true 代表使能，false 代表禁使能

返回值：

成功返回：0；失败返回：错误码， rm_define.h 查询。