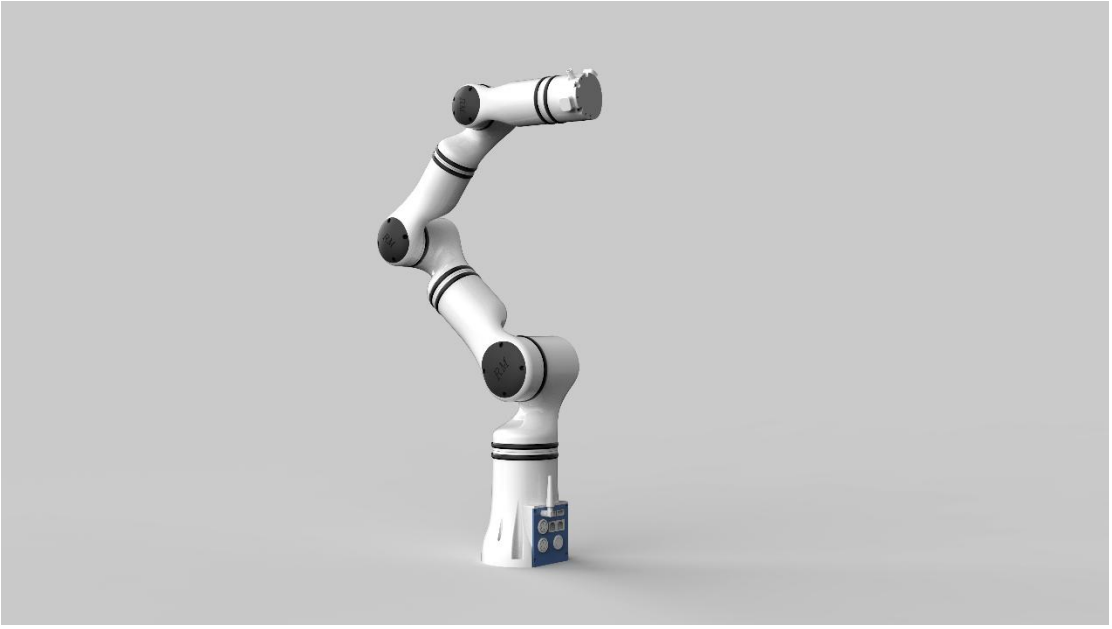
睿尔曼机械臂 SDK(C)学习资料

（内部学习）



睿尔曼智能科技（北京）有限公司

目录

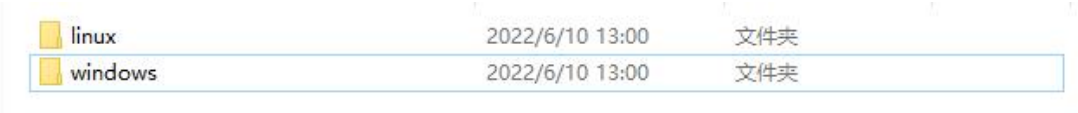
1. SDK 包简介

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。



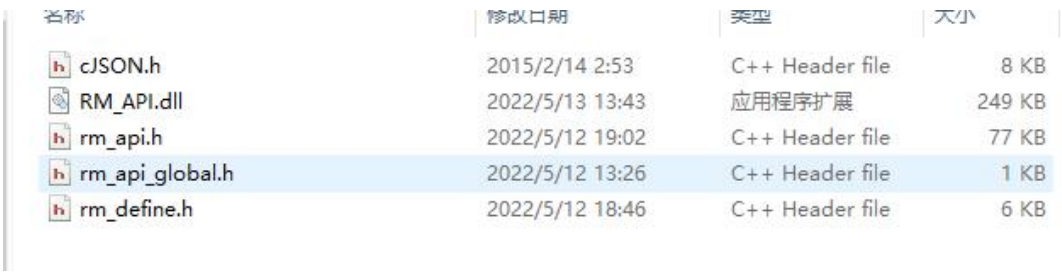
API 组成如下图所示:

（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。

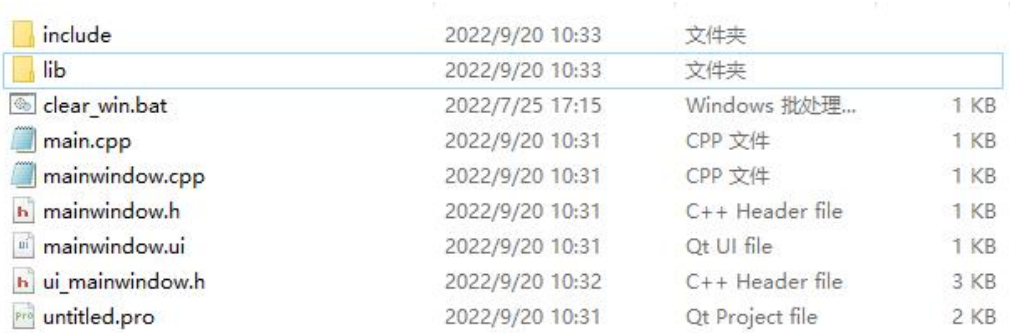


2. 使用说明

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 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;

    //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。

（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。

3. 接口函数示例

3.1. 连接相关函数

3.1.1. API 初始化 RM_API_Init

int RM_API_Init(int devMode, RM_Callback pCallback);	
函数功能	该函数用于 API 初始化
参数	（1）dev_mode 目标设备型号 ARM_65/ ARM_75/ARM_63_2/ ARM_ECO65/ ARM_GEN72/ARM_ECO62/ARM_ECO63。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。
示例	//初始化 API RM_API_Init(dev_mode,NULL);

3.1.2. API 反初始化 RM_API_UnInit

int RM_API_UnInit();	
函数功能	该函数用于 API 反初始化
示例	//API 反初始化 RM_API_UnInit();

3.1.3. 查询 API 版本信息 RM_API

char * API_Version();	
函数功能	该函数用于查询 API 版本信息
返回值	API 版本号
示例	//返回 API 版本 API_Version();

3.1.4. 连接机械臂 Arm_Socket_Start

SOCKHANDLE Arm_Socket_Start(char* arm_ip,int arm_port, int recv_timeout);	
函数功能	该函数用于连接机械臂

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。
返回值	成功返回：Socket 句柄。失败返回：错误码，rm_define.h 查询。
示例	// 连接服务器 返回全局句柄 m_sockhand = Arm_Socket_Start("192.168.1.18",8080,50000);

3.1.5. 查询连接状态 Arm_Socket_State

int Arm_Socket_State(SOCKHANDLE ArmSocket);	
函数功能	该函数用于查询机械臂连接状态
参数	（1）ArmSocket Socket 句柄。
返回值	正常返回：SYS_NORMAL。失败返回：错误码， rm_define.h 查询。
示例	// 网络连接状态 Arm_Socket_State(m_sockhand);

3.1.6. 关闭连接 Arm_Socket_Close

void Arm_Socket_Close(SOCKHANDLE ArmSocket);

函数功能	该函数用于关闭与机械臂的 socket 连接。
参数	（1）ArmSocket Socket 句柄。
示例	//关闭 socket 连接 Arm_Socket_Close(m_sockhand);

3.2. 关节配置函数

对机械臂的关节参数进行设置，如果关节发生错误，则无法修改关节参数，必须先清除关节错误代码。另外设置关节另外之前，必须先将关节掉使能，否则会设置不成功。

注意：关节所有参数在修改完成后，会自动保存到关节 Flash，立即生效，之后关节处于掉使能状态，修改完参数后必须发送指令控制关节上使能。

注意：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定位报错。关节正常上使能后，用户方可控制关节运动。

3.2.1. 设置关节最大速度 Set_Joint_Speed

int Set_Joint_Speed(SOCKHANDLE ArmSocket, byte joint_num, float speed, bool block);	
函数功能	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。
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。
示例	//设置关节 2，最大速度 30°/s byte joint_num = 2; float speed = 30; ret=Set_Joint_Speed(m_sockhand,joint_num,speed ,RM_BLOCK);

3.2.2. 设置关节最大加速度 Set_Joint_Acc

int Set_Joint_Acc(SOCKHANDLE ArmSocket, byte joint_num, float acc, bool block);	
函数功能	该函数用于设置关节最大加速度，单位：°/s²。建议使用默认最大加速度，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。
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。
示例	//设置关节 2 最大加速度 600°/s² byte joint_num = 2; float acc = 600; ret = Set_Joint_Acc(m_sockhand,joint_num,acc,RM_BLOCK);

3.2.3. 设置关节最小限位 Set_Joint_Min_Pos

int Set_Joint_Min_Pos(SOCKHANDLE ArmSocket, byte joint_num, float joint, bool block);	
函数功能	该函数用于设置关节所能到达的最小限位，单位：度。
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。
示例	//设置关节 6，最小限位度数-360° byte joint_num = 6; float joint = -360; ret=Set_Joint_Min_Pos(m_sockhand,joint_num,joint,RM_BLOCK);

3.2.4. 设置关节最大限位 Set_Joint_Max_Pos

int Set_Joint_Max_Pos(SOCKHANDLE ArmSocket, byte joint_num, float joint, bool block);	
函数功能	该函数用于设置关节所能到达的最大位置，单位：度。
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。
示例	//设置关节 6，最大限位度数 360° byte joint_num = 6; float joint = 360; ret=Set_Joint_Max_Pos(m_sockhand,joint_num,joint,RM_BLOCK);

3.2.5. 设置关节使能 Set_Joint_EN_State

int Set_Joint_EN_State(SOCKHANDLE ArmSocket, byte joint_num, bool state, bool block);	
函数功能	该函数用于设置关节使能状态。
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。
示例	//设置关节 1 上使能 byte joint_num = 1; ret=Set_Joint_EN_State(m_sockhand,joint_num,true,RM_BLOCK);

3.2.6. 设置关节零位 Set_Joint_Zero_Pos

int Set_Joint_Zero_Pos(SOCKHANDLE ArmSocket, byte joint_num, bool block);	
函数功能	该函数用于将当前位置设置为关节零位。
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。
示例	//设置关节 3 位置为零位 byte joint_num = 3; ret=Set_Joint_Zero_Pos(m_sockhand,joint_num ,RM_BLOCK);

3.2.7. 清除关节错误代码 Set_Joint_Err_Clear

int Set_Joint_Err_Clear(SOCKHANDLE ArmSocket, byte joint_num, bool block);	
函数功能	该函数用于清除关节错误代码。
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。
示例	//清除关节 2 错误代码 byte joint_num = 2; ret=Set_Joint_Err_Clear(m_sockhand,joint_num,RM_BLOCK);

3.3. 关节参数查询函数

3.3.1. 获取关节最大速度 Get_Joint_Speed

int Get_Joint_Speed(SOCKHANDLE ArmSocket, float *speed);	
函数功能	该函数用于查询关节最大速度。

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。
示例	//查询关节最大速度 float speed[6] = {0,1,2,3,4,5}; ret = Get_Joint_Speed(m_sockhand,speed);

3.3.2. 获取关节最大加速度 Get_Joint_Acc

int Get_Joint_Acc(SOCKHANDLE ArmSocket, float *acc);	
函数功能	该函数用于获取关节最大加速度。
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。
示例	//查询关节最大加速度 float acc[6] = {0,1,2,3,4,5}; ret = Get_Joint_Acc(m_sockhand,acc);

3.3.3. 获取关节最小位置 Get_Joint_Min_Pos

int Get_Joint_Min_Pos(SOCKHANDLE ArmSocket, float *min_joint);	
函数功能	该函数用于获取关节最小位置。
参数	（1）ArmSocket Socket 句柄 （2）min_joint 存放关节 1~7 最小位置数组，单位：°
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//查询关节最小位置 float min_joint[6] = {0,1,2,3,4,5}; ret = Get_Joint_Min_Pos(m_sockhand,min_joint);

3.3.4. 获取关节最大位置 Get_Joint_Max_Pos

int Get_Joint_Max_Pos(SOCKHANDLE ArmSocket, float *max_joint);	
函数功能	该函数用于获取关节最大位置。
参数	（1）ArmSocket Socket 句柄 （2）max_joint 存放关节 1~7 最大位置数组，单位：°
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//查询关节最大位置

	float max_joint[6] = {0,1,2,3,4,5}; ret = Get_Joint_Max_Pos(m_sockhand,max_joint);

3.3.5. 获取关节使能状态 Get_Joint_EN_State

int Get_Joint_EN_State(SOCKHANDLE ArmSocket, byte *state);	
函数功能	该函数用于获取关节使能状态。
参数	（1）ArmSocket Socket 句柄 （2）state 存放关节 1~7 使能状态数组，1-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。
示例	//查询关节使能状态 byte sta[6] = {0,1,2,3,4,5}; ret = Get_Joint_EN_State(m_sockhand,sta);

3.3.6. 获取关节错误代码 Get_Joint_Err_Flag

int Get_Joint_Err_Flag(SOCKHANDLE ArmSocket, uint16_t* state, uint16_t* bstate);	
函数功能	该函数用于获取关节使能状态。
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。
示例	//获取关节错误代码 uint16_t sta[6] = {0,1,2,3,4,5}; uint16_t bsta[6] = {0,1,2,3,4,5}; ret = Get_Joint_Err_Flag(m_sockhand,sta,bsta);

3.3.7. 查询关节软件版本号 Get_Joint_Software_Version

int Get_Joint_Software_Version(SOCKHANDLE ArmSocket, float* version);	
函数功能	该函数用于查询关节软件版本号。
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。
示例	//获取关节软件版本号 float ver[6] = {0}; ret = Get_Joint_Software_Version(m_sockhand,ver);

3.4. 机械臂末端运动参数配置

3.4.1. 设置末端最大线速度 Set_Arm_Line_Speed

int Set_Arm_Line_Speed(SOCKHANDLE ArmSocket, float speed, bool block);	
函数功能	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。
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。
示例	//设置机械臂末端最大线速度 0.1m/s float speed = 0.1; ret = Set_Arm_Line_Speed(m_sockhand,speed ,RM_BLOCK);

3.4.2. 设置末端最大线加速度 Set_Arm_Line_Acc

int Set_Arm_Line_Acc(SOCKHANDLE ArmSocket, float acc, bool block);	
函数功能	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。
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。
示例	//设置机械臂末端最大线加速度 $2m/s^2$ float acc = 2; ret = Set_Arm_Line_Acc(m_sockhand,acc,RM_BLOCK);

3.4.3. 设置末端最大角速度 Set_Arm_Angular_Speed

int Set_Arm_Angular_Speed(SOCKHANDLE ArmSocket, float speed, bool block);	
函数功能	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。
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。
示例	//设置机械臂末端最大角速度 0.2rad/s float speed = 0.2; ret=Set_Arm_Angular_Speed(m_sockhand,speed,RM_BLOCK);

3.4.4. 设置末端最大角加速度 Set_Arm_Angular_Acc

int Set_Arm_Angular_Acc(SOCKHANDLE ArmSocket, float acc, bool block);	
函数功能	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。
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。
示例	//设置机械臂末端最大角加速度 4rad/s² float acc = 4; ret = Set_Arm_Angular_Acc(m_sockhand,acc,RM_BLOCK);

3.4.5. 获取末端最大线速度 Get_Arm_Line_Speed

int Get_Arm_Line_Speed(SOCKHANDLE ArmSocket, float *speed);	
函数功能	该函数用于获取机械臂末端最大线速度。
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。
示例	//获取机械臂末端线速度 float speed = 0; ret = Get_Arm_Line_Speed(m_sockhand,&speed);

3.4.6. 获取最大末端线加速度 Get_Arm_Line_Acc

int Get_Arm_Line_Acc(SOCKHANDLE ArmSocket, float *acc);	
函数功能	该函数用于获取机械臂末端最大线加速度。
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。
示例	//获取机械臂末端线加速度 float acc = 0; ret = Get_Arm_Line_Acc(m_sockhand,&acc);

3.4.7. 获取末端最大角速度 Get_Arm_Angular_Speed

int Get_Arm_Angular_Speed(SOCKHANDLE ArmSocket, float *speed);	
函数功能	该函数用于获取机械臂末端最大角速度。
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。

示例	//获取机械臂末端角速度 float speed = 0; ret = Get_Arm_Angular_Speed(m_sockhand,&speed);

3.4.8. 获取末端最大角加速度 Get_Arm_Angular_Acc

int Get_Arm_Angular_Acc(SOCKHANDLE ArmSocket, float *speed);	
函数功能	该函数用于获取机械臂末端最大角加速度。
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。
示例	//获取末端角加速度 float acc = 0; ret = Get_Arm_Angular_Acc(m_sockhand,&acc);

3.4.9. 设置机械臂末端参数为初始值 Set_Arm_Tip_Init

int Set_Arm_Tip_Init(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于设置机械臂末端参数为初始值。
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。
示例	//初始化机械臂参数，机械臂的末端参数回复到默认值。其中 末端线速度：0.1m/s 末端线加速度：0.5m/s^2 末端角速度：0.2rad/s 末端角加速度：1rad/s^2 ret = Set_Arm_Tip_Init(m_sockhand,RM_BLOCK);

3.4.10. 设置碰撞等级 Set_Collision_Stage

int Set_Collision_Stage(SOCKHANDLE ArmSocket, int stage, bool block);	
函数功能	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。
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。
示例	//设置机械臂碰撞防护等级为 1 int stage = 1; ret=Set_Collision_Stage(m_sockhand,stage,RM_BLOCK);

3.4.11. 查询碰撞等级 Get_Collision_Stage

int Get_Collision_Stage(SOCKHANDLE ArmSocket, int *stage);	
函数功能	该函数用于查询机械臂动力学碰撞等级，等级 0~8，数值越高，碰撞检测越灵敏，同时也更易发生误碰撞检测。机械臂上电后默认碰撞等级为 0，即不检测碰撞。
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。
示例	//查询机械臂动力学碰撞等级 int stage = -1; ret = Get_Collision_Stage(m_sockhand,&stage);

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

int Set_Joint_Zero_Offset (SOCKHANDLE ArmSocket, float *offset, bool block);	
函数功能	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。
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：度 （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。
示例	//设置关节零位偏移 //关节 1~6 的零位补偿角度：1°，-2°，3°，-4°，5°，-6° float offset[7] = {1,-2,3,-4,5,-6} ; ret = Set_Joint_Zero_Offset(m_sockhand,offset,RM_BLOCK);
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！

3.5. 机械臂末端接口板

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

int Get_Tool_Software_Version(SOCKHANDLE ArmSocket, int* version);	
函数功能	该函数用于查询末端接口板软件版本号
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。
示例	//查询末端接口板软件版本号 int ver = 0; ret = Get_Tool_Software_Version(m_sockhand,&ver);

3.6. 工具坐标系设置

3.6.1. 标定点位 Auto_Set_Tool_Frame

int Auto_Set_Tool_Frame(SOCKHANDLE ArmSocket, byte point_num, bool block);	
函数功能	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。

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。
示例	//自动计算工具坐标系标记点位，标定当前位置为参考点 6 byte point_num = 6; ret = Auto_Set_Tool_Frame(m_sockhand,point_num,RM_BLOCK);
备注	机械臂控制器最多只能存储 10 个工具信息，在建立新的工具之前，请确认工具数量没有超过限制，否则建立新工具无法成功。

3.6.2. 生成工具坐标系 Generate_Auto_Tool_Frame

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

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。
示例	//自动计算工具坐标系生成坐标系，名称 t1，末端负载 5kg，质心位置：x-1mm,y-2mm,z-3mm const char *name = "t1"; float payload = 5; float x = 1; float y = 2; float z = 3; ret=Generate_Auto_Tool_Frame(m_sockhand,name ,payload,x,y,z,RM_BLOCK);
备注	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。

3.6.3. 手动设置工具坐标系 Manual_Set_Tool_Frame

int Manual_Set_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose,float payload,float x,float y,float z, bool block);	
函数功能	该函数用于手动设置工具坐标系。
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。
示例	//手动输入工具坐标系，名称 t2 //工具位置：x：331.78mm，y:49.12mm，z：98.98mm

	//工具姿态：rx：3.11249rad，ry：0.0rad，rz：-3.00656rad //末端负载 5kg，质心位置：x-1mm,y-2mm,z-3mm const char *name1 = "t2"; Pose pose; pose.position.x = 0.33178; pose.position.y = 0.04912; pose.position.z = 0.09898; pose.euler.rx = 3.11249; pose.euler.ry = 0; pose.euler.rz = -3.00656; float payload = 5; float x = 1; float y = 2; float z = 3 ; ret=Manual_Set_Tool_Frame(m_sockhand,name1,pose,payload,x,y,z,RM_BLOCK);
备注	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。

3.6.4. 切换当前工具坐标系 Change_Tool_Frame

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

函数功能	该函数用于切换当前工具坐标系
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。
示例	//切换为 t1 工具坐标系 const char *name = "t1"; ret=Change_Tool_Frame(m_sockhand,name,RM_BLOCK);

3.6.5. 删除指定工具坐标系 Delete_Tool_Frame

int Delete_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);
函数功能	该函数用于删除指定工具坐标系
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。
示例	//删除 t1 工具坐标系 const char *name = "t1"; ret=Delete_Tool_Frame(m_sockhand,name,RM_BLOCK);
备注	删除坐标系后，机械臂将切换到机械臂法兰末端工具坐标系。

3.6.6. 修改指定工具坐标系 Update_Tool_Frame

int Update_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose, float payload,float x,float y,float z);	
函数功能	该函数用于修改指定工具坐标系。
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。
示例	//修改 t2 坐标系 //工具位置：x：331.78mm，y:49.12mm，z：98.98mm //工具姿态：rx：3.11249rad，ry：0.0rad，rz：-3.00656rad //末端负载 2kg，质心位置：0, 0, 1 const char *name1 = "t2"; Pose pose; pose.position.x = 0.33178; pose.position.y = 0.04912; pose.position.z = 0.09898; pose.euler.rx = 3.11249; pose.euler.ry = 0; pose.euler.rz = -3.00656; float payload = 2; float x = 0; float y = 0; float z = 1 ; ret=Update_Tool_Frame(m_sockhand,name1,pose,payload,x,y,z,

	RM_BLOCK);

3.6.7. 设置工具坐标系的包络参数 Set_Tool_Envelope

int Set_Tool_Envelope(SOCKHANDLE ArmSocket, ToolEnvelopeList list);	
函数功能	该函数用于设置工具坐标系的包络参数
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。
示例	// 设置 test_1 工具坐标系包络参数 // 2 个包络球，名称分别为”left”,”right” ToolEnvelopeList tool; strcpy(tool.tool_name,"test_1"); strcpy(tool.list[0].name, "right"); tool.list[0].radius = 0.01; tool.list[0].x = 0.1; tool.list[0].y = 0.1; tool.list[0].z = 0.1; strcpy(tool.list[1].name, "left"); tool.list[1].radius = 0.03;

	tool.list[1].x = 0.1; tool.list[1].y = 0.1; tool.list[1].z = 0.1; tool.count = 2; ret = Set_Tool_Envelope(handle, tool);

3.6.8. 获取工具坐标系的包络参数 Get_Tool_Envelope

int Get_Tool_Envelope(SOCKHANDLE ArmSocket, const char *name, ToolEnvelopeList *list);	
函数功能	该函数用于获取指定工具坐标系的包络参数
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。
示例	// 获取 test_1 工具坐标系的包络参数 const char *name = "test_1"; ToolEnvelopeList tool; ret = service.Service_Get_Tool_Envelope(handle, name ,&tool);

3.7. 工具坐标系查询

3.7.1. 获取当前工具坐标系 Get_Current_Tool_Frame

int Get_Current_Tool_Frame(SOCKHANDLE ArmSocket, FRAME *tool);	
函数功能	该函数用于获取当前工具坐标系
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。
示例	//获取当前工具坐标系 FRAME tool; ret = Get_Current_Tool_Frame(m_sockhand,&tool);

3.7.2. 获取指定工具坐标系 Get_Given_Tool_Frame

int Get_Given_Tool_Frame(SOCKHANDLE ArmSocket, const char *name, FRAME *tool);	
函数功能	该函数用于获取指定工具坐标系
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。
示例	//获取 t2 工具坐标系参数 const char *name = "t2"; FRAME tool; ret = Get_Given_Tool_Frame(m_sockhand,name,&tool);

3.7.3. 获取所有工具坐标系名称 Get_All_Tool_Frame

int Get_All_Tool_Frame(SOCKHANDLE ArmSocket, FRAME_NAME *names, int *len);	
函数功能	该函数用于获取所有工具坐标系名称
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。
示例	//获取所有工具坐标系名称 FRAME_NAME names[10]={0};

	int len; ret = Get_All_Tool_Frame(m_sockhand,names,&len);

3.8. 工作坐标系设置

3.8.1. 自动设置工作坐标系 Auto_Set_Work_Frame

int Auto_Set_Work_Frame(SOCKHANDLE ArmSocket, const char *name, byte point_num, bool block);	
函数功能	该函数用于三点法自动设置工作坐标系。
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。
示例	//自动设置工作坐标系，名称 w1，标记当前位置为点 3 const char *name3 = "w1";

	byte point_num=3; ret=Auto_Set_Work_Frame(m_sockhand,name3,point_num,RM_BLOCK);
备注	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。

3.8.2. 手动设置工作坐标系 Manual_Set_Work_Frame

int Manual_Set_Work_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose, bool block);	
函数功能	该函数用于手动设置工作坐标系。
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。
示例	// 手动输入工作坐标系，名称 w2

	坐标系位置：x：0.1m，y:0.2m，z：0.03m 坐标系姿态：rx：0.4rad，ry：0.5rad，rz：0.6rad const char *name = "w2"; Pose pose; pose.position.x = 0.1; pose.position.y = 0.2; pose.position.z = 0.03; pose.euler.rx = 0.4; pose.euler.ry = 0.5; pose.euler.rz = 0.6; ret=Manual_Set_Work_Frame(m_sockhand,name,pose,RM_BLOCK);
备注	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。

3.8.3. 切换当前工作坐标系 Change_Work_Frame

int Change_Work_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);	
函数功能	该函数用于切换当前工作坐标系
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。
示例	//切换为 w1 工作坐标系 const char *name = "w1"; ret=Change_Work_Frame(m_sockhand,name,RM_BLOCK);

3.8.4. 删除指定工作坐标系 Delete_Work_Frame

int Delete_Work_Frame(SOCKHANDLE ArmSocket, const char *name, bool block);	
函数功能	该函数用于删除指定工作坐标系。
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。
示例	//删除 w1 工作坐标系

	const char *name = "w1"; ret=Delete_Work_Frame(m_sockhand,name,RM_BLOCK);
备注	删除坐标系后，机械臂将切换到机械臂基坐标系

3.8.5. 修改指定工作坐标系 Update_Work_Frame

int Update_Work_Frame(SOCKHANDLE ArmSocket, const char *name, Pose pose);	
函数功能	该函数用于修改指定工作坐标系。
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。
示例	// 修改 w1 工作坐标系 const char *name = "w1"; Pose pose; pose.position.x = 0.02; ret = service.Service_Update_Work_Frame(handle, name, pose);

3.9. 工作坐标系查询

3.9.1. 获取当前工作坐标系 Get_Current_Work_Frame

int Get_Current_Work_Frame(SOCKHANDLE ArmSocket, FRAME *frame);	
函数功能	该函数用于获取当前工作坐标系。
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。
示例	//获取当前工作坐标系 FRAME work; ret = Get_Current_Work_Frame(m_sockhand,&work);

3.9.2. 获取指定工作坐标系 Get_Given_Work_Frame

int Get_Given_Work_Frame(SOCKHANDLE ArmSocket, const char *name, Pose *pose);	
函数功能	该函数用于获取指定工作坐标系
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。
示例	//获取 w2 工作坐标系 char name[12] = "w2"; Pose pose; ret = Get_Given_Work_Frame(m_sockhand,name,&pose);

3.9.3. 获取所有工作坐标系名称 Get_All_Work_Frame

int Get_All_Work_Frame(SOCKHANDLE ArmSocket, FRAME_NAME *names, int *len);	
函数功能	该函数用于获取所有工作坐标系名称。
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。
示例	//获取所有工作坐标系名称 FRAME_NAME names[10]={0}; int len;

	ret = Get_All_Work_Frame(m_sockhand,names,&len);

3.10. 机械臂状态查询

3.10.1. 获取机械臂当前状态 Get_Current_Arm_State

int Get_Current_Arm_State(SOCKHANDLE ArmSocket, float *joint, Pose *pose, uint16_t *Arm_Err, uint16_t *Sys_Err);	
函数功能	该函数用于获取机械臂当前状态
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。
示例	//获取机械臂当前状态 float joint[6] ; Pose pose;

	uint16_t arm_Err; uint16_t sys_Err; ret=Get_Current_Arm_State(m_sockhand,joint,&pose,&arm_err,&sys_err);

3.10.2. 获取关节温度 Get_Joint_Temperature

int Get_Joint_Temperature(SOCKHANDLE ArmSocket, float *temperature);	
函数功能	该函数用于获取关节当前温度。
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。
示例	//获取关节当前温度 float temperature[6] = {0,1,2,3,4,5}; ret = Get_Joint_Temperature(m_sockhand,temperature);

3.10.3. 获取关节电流 Get_Joint_Current

int Get_Joint_Current(SOCKHANDLE ArmSocket, float *current);	
函数功能	该函数用于获取关节当前电流。
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。
示例	//获取关节当前电流 float current[6] = {0,1,2,3,4,5}; ret = Get_Joint_Current(m_sockhand,current);

3.10.4. 获取关节电压 Get_Joint_Voltage

int Get_Joint_Voltage(SOCKHANDLE ArmSocket, float *voltage);	
函数功能	该函数用于获取关节当前电压。
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。
示例	//获取关节当前电压 float voltage[6] = {0,1,2,3,4,5}; ret = Get_Joint_Voltage(m_sockhand,voltage);

3.10.5. 获取关节当前角度 Get_Joint_Degree

int Get_Joint_Degree (SOCKHANDLE ArmSocket, float *joint);

函数功能	该函数用于获取机械臂各关节的当前角度
参数	（1）ArmSocket Socket 句柄 （2）joint 关节 1~7 当前角度数组；
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//获取机械臂各关节的当前角度 float joint[6] = {0,1,2,3,4,5}; ret = Get_Joint_Degree(m_sockhand,joint);

3.10.6. 获取所有状态 Get_Arm_All_State

int Get_Arm_All_State(SOCKHANDLE ArmSocket, JOINT_STATE *joint_state);	
函数功能	该函数用于获取机械臂所有状态
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。
示例	//获取机械臂所有状态 JOINT_STATE joint_state = {0}; ret = Get_Arm_All_State(m_sockhand,&joint_state);

3.10.7. 获取轨迹规划计数 Get_Arm_Plan_Nume

int Get_Arm_Plan_Num(SOCKHANDLE ArmSocket, int* plan);	
函数功能	该函数用于获取机械臂轨迹规划计数
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。
示例	//获取机械臂轨迹规划计数 int plan; ret = Get_Arm_Plan_Num(m_sockhand,&plan);

3.11. 机械臂初始位姿

3.11.1. 设置初始位姿角度 Set_Arm_Init_Pose

int Set_Arm_Init_Pose(SOCKHANDLE ArmSocket, const float *target, bool block);	
函数功能	该函数用于设置机械臂的初始位姿角度。
参数	（1）ArmSocket Socket 句柄 （2）target 机械臂初始位置关节角度数组

	（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。
示例	//设置初始位置关节角度 0°,0°,0°,0°,0°,0°,0° const float target[7] = {0,0,0,0,0,0,0}; ret = Set_Arm_Init_Pose(m_sockhand,target,RM_BLOCK);

3.11.2. 获取初始位姿角度 Get_Arm_Init_Pose

int Get_Arm_Init_Pose(SOCKHANDLE ArmSocket, float *joint);	
函数功能	该函数用于获取机械臂初始位姿角度。
参数	（1）ArmSocket Socket 句柄 （2）joint 机械臂初始位姿关节角度数组
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//获取初始位置角度 float joint[7] = {0,1,2,3,4,5,6}; ret = Get_Arm_Init_Pose(m_sockhand,joint);

3.11.3. 设置安装角度 Set_Install_Pose

int Set_Install_Pose(SOCKHANDLE ArmSocket, float x, float y, float z, bool

block);	
函数功能	该函数用于设置机械臂安装方式。
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。
示例	//设置安装角度旋转角 30°，俯仰角 60°,方位角 90° float x = 30; float y = 60; float z = 90; ret = Set_Install_Pose(m_sockhand,x,y,z,RM_BLOCK);

3.11.4. 查询安装角度 Get_Install_Pose

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

函数功能	该函数用于查询机械臂安装角度。
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。
示例	//获取安装角度 float fx,fy,fz; ret = Get_Install_Pose(m_sockhand,&fx,&fy,&fz);

3.12. 机械臂运动规划

3.12.1. 关节空间运动 Movej_Cmd

int Movej_Cmd(SOCKHANDLE ArmSocket, float *joint, byte v, float r, int trajectory_connect, bool block);
函数功能	该函数用于关节空间运动。
参数	（1）ArmSocket Socket 句柄

	（2）joint 目标关节 1~7 角度数组 （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。
示例	//关节运动，关节角度[0°,0°,0°,0°,90°,0°,0°],速度系数 20%，交融半径：0，立即规划执行 float joint[6] = {0,0,0,90,0,0}; ret = Movej_Cmd(m_sockhand,joint,20,0,0,RM_BLOCK);
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

3.12.2. 笛卡尔空间直线运动 Movel_Cmd

int Movel_Cmd(SOCKHANDLE ArmSocket, Pose pose, byte v, float r, int

trajectory_connect, bool block);	
函数功能	该函数用于笛卡尔空间直线运动。
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。
示例	//直线运动，目标位置：x：-0.3m，y:-0.03m，z：0.215m；目标姿态：rx:3rad，ry:0.1rad，rz:0.1rad；速度系数 20%，不交融，立即规划执行 Pose pose; pose.position.x=-0.300;

	pose.position.y=-0.030; pose.position.z=0.215; pose.euler.rx=3.0; pose.euler.ry=0.1; pose.euler.rz=0.1; ret = Movel_Cmd(m_sockhand,pose, 20,0,0,RM_BLOCK);
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

3.12.3. 笛卡尔空间圆弧运动 Movec_Cmd

int Movec_Cmd(SOCKHANDLE ArmSocket, Pose pose_via, Pose pose_to, byte v, float r, byte loop, int trajectory_connect, bool block);	
函数功能	该函数用于笛卡尔空间圆弧运动
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 速度百分比系数，1~100

	（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。
示例	/*圆弧运动:中间点位置：x：-0.3m，y:-0.03m，z：0.215m;中间点姿态：rx:3rad，ry:0.1rad，rz:0.1rad;终点位置：x：0.4m，y:-0.03m，z：0.215m;终点姿态：rx:3rad，ry:0.1rad，rz:0.1rad;速度系数 20%，;不交融;不循环，立即规划执行*/ Pose povia; povia.position.x=-0.300; povia.position.y=-0.03; povia.position.z=0.215; povia.euler.rx=3.0; povia.euler.ry=0.1; povia.euler.rz=0.1;

	Pose poto; poto.position.x=-0.4; poto.position.y=-0.030; poto.position.z=0.215; poto.rx=3.0; poto.ry=0.1; poto.rz=0.1; ret = Movec_Cmd(m_sockhand,povia,poto,20,0,0,0,RM_BLOCK);
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

3.12.4. 关节角度 CANFD 透传 Movej_CANFD

int Movej_CANFD(SOCKHANDLE ArmSocket, float *joint, bool follow, float expand);	
函数功能	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。
参数	（1）ArmSocket Socket 句柄 （2）joint 关节 1~7 目标角度数组 （3）follow 是否高跟随,true--高跟随，false--低跟随

	（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
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//角度透传到 CANFD，目标关节角度：[1°,0°,20°,30°,0°,20°] float fl1[6] = { 1, 0, 20, 30, 0, 20}; Movej_CANFD(m_sockhand,fl1,true,0);
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。

3.12.5. 位姿 CANFD 透传 Movep_CANFD

int Movep_CANFD(SOCKHANDLE ArmSocket, Pose pose, bool follow);	
函数功能	该函数用于角度不经规划，直接通过 CANFD 透传给机械臂，使用透传接口是，请勿使用其他运动接口。

参数	（1）ArmSocket Socket 句柄 （2）pose 位姿（优先采用四元数表达） （3）follow 是否高跟随,true--高跟随，false--低跟随
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	/*pose：目标位姿，位置精度：0.001mm，姿态精度：0.001rad 目标位置：x：0m，y:0m，z：0.85049m 目标姿态：rx:0rad，ry:0rad，rz:3.142rad 目标位姿为当前工具在当前工作坐标系下的数值。*/ Pose pose; pose.position.x=0; pose.position.y=0; pose.position.z=0.85049; pose.euler.rx=0; pose.euler.ry=0; pose.euler.rz=3.142; Movep_CANFD(m_sockhand,pose,true);

3.12.6. 计算环绕运动位姿 MoveRotate_Cmd

int MoveRotate_Cmd(SOCKHANDLE ArmSocket, int rotateAxis, float

rotateAngle,Pose choose_axis, byte v, float r, int trajectory_connect, bool block);	
函数功能	该函数用于计算环绕运动位姿并按照结果运动。
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(度) （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。
示例	//计算沿 X 轴的环绕运动位姿并立即规划执行 Pose pose; pose.position.x = 0.012f; pose.position.y = 0.012f; pose.position.z = 0.012f; pose.euler.rx = 1; pose.euler.ry = 1; pose.euler.rz = 1; float rotateAngle = 10; int ret = -1; ret=MoveRotate_Cmd(m_sockhand,1,rotateAngle,pose,20,0,0,RM_ BLOCK);

3.12.7. 沿工具端位姿移动 MoveCartesianTool_Cmd

int MoveCartesianTool_Cmd(SOCKHANDLE ArmSocket, float movelengthx,float movelengthy, float movelengthz, int m_dev, byte v, float r, int trajectory_connect, bool block);	
函数功能	该函数用于沿工具端位姿运动。
参数	（1）ArmSocket Socket 句柄 （2）movelengthx

	沿 X 轴移动长度，米为单位 （3）Movelengthy 沿 Y 轴移动长度，米为单位 （4）movelengthz 沿 Z 轴移动长度，米为单位 （5）m_dev 机械臂型号 （6）v 速度 （7）r 交融半径百分比系数，0~100 （8）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 （9）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。
	//计算沿工具端位姿运动，沿 X 轴运动 0.01m，沿 Y 轴运动 0.01m，沿 Z 轴运动 0.01m，立即规划执行 float joint[6] = {20,20,70,30,90,120};

	float movelengthx = 0.01f; float movelengthy = 0.01f; float movelengthz = 0.01f; int ret = -1; ret=MoveCartesianTool_Cmd(m_sockhand,joint,movelengthx,mo velengthy,movelengthz,65,20,0,0,RM_BLOCK);

3.12.8. 快速急停 Move_Stop_Cmd

int Move_Stop_Cmd(SOCKHANDLE ArmSocket, bool block);	
函 数 功 能	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。
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。
示例	//轨迹急停 ret = Move_Stop_Cmd(m_sockhand,RM_BLOCK);

3.12.9. 暂停当前规划 Move_Pause_Cmd

int Move_Pause_Cmd(SOCKHANDLE ArmSocket, bool block);

函 数 功 能	该函数用于轨迹暂停，暂停在规划轨迹上，轨迹可恢复。
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。
示例	//轨迹暂停 ret = Move_Pause_Cmd(m_sockhand,RM_BLOCK);

3.12.10. 继续当前轨迹 Move_Continue_Cmd

int Move_Continue_Cmd(SOCKHANDLE ArmSocket, bool block);	
函 数 功 能	该函数用于轨迹暂停后，继续当前轨迹运动
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。
示例	//轨迹暂停后恢复

	ret = Move_Continue_Cmd(m_sockhand,RM_BLOCK);

3.12.11. 清除当前轨迹 Clear_Current_Trajectory

int Clear_Current_Trajectory(SOCKHANDLE ArmSocket, bool block);	
函数功能	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！！！！
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。
示例	//清除当前轨迹 ret = Clear_Current_Trajectory(m_sockhand,RM_BLOCK);

3.12.12. 清除所有轨迹 Clear_All_Trajectory

int Clear_All_Trajectory(SOCKHANDLE ArmSocket, bool block);	
函数功能	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！
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。
示例	//清除所有轨迹 ret = Clear_All_Trajectory(m_sockhand,RM_BLOCK);

3.12.13. 关节空间运动 Movej_P_Cmd

int Movej_P_Cmd(SOCKHANDLE ArmSocket, Pose pose, byte v, float r, int trajectory_connect, bool block);	
函数功能	该函数用于关节空间运动到目标位姿
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。
示例	//关节空间运动，目标位置：x：0.1m，y:0.2m，z：0.03m；目标姿态：rx:0.4rad，ry:0.5rad，rz:0.6rad；速度系数 20%，不交融，立即规划执行 Pose pose; pose.position.x=-0.1; pose.position.y=-0.2; pose.position.z=0.3; pose.euler.rx=0.4; pose.euler.ry=0.5; pose.euler.rz=0.6; ret = Movej_P_Cmd(m_sockhand,pose, 20,0,0,RM_BLOCK);
注意	该运动暂不支持轨迹交融。

3.12.14. 样条曲线运动 Moves_Cmd

int Moves_Cmd(SOCKHANDLE ArmSocket, Pose pose, byte v, float r, int

trajectory_connect, bool block);	
函数功能	该函数用于样条曲线运动到目标位姿
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。
示例	//样条曲线运动，目标位置：x：0.1m，y:0.2m，z：0.03m；目标姿态：rx:0.4rad，ry:0.5rad，rz:0.6rad；速度系数 20%，不交融，立即规划执行 Pose pose;

	pose.position.x=-0.1; pose.position.y=-0.2; pose.position.z=0.3; pose.euler.rx=0.4; pose.euler.ry=0.5; pose.euler.rz=0.6; ret = Moves_Cmd(m_sockhand,pose, 20,0,0,RM_BLOCK);
注意	该运动暂不支持轨迹交融。

3.12.15. 关节空间跟随运动 Movej_Follow

int Movej_Follow(SOCKHANDLE ArmSocket, const float *joint);	
函数功能	该函数用于给机械臂更新目标关节角度，机械臂会自动规划轨迹运动跟随目标点。用户使用该指令时无需自行规划轨迹。
参数	（1）ArmSocket Socket 句柄 （2）joint 关节 1~7 目标角度数组
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//目标关节角度：[1°,0°,20°,30°,0°,20°] float fl1[6] = { 1, 0, 20, 30, 0, 20}; Movej_Follow(m_sockhand,fl1);

3.12.16. 笛卡尔空间跟随运动 Movep_Follow

int Movep_Follow(SOCKHANDLE ArmSocket, Pose pose);	
函 数 功能	该函数用于给机械臂更新目标位姿，机械臂会自动规划轨迹运动跟随目标点。用户使用该指令时无需自行规划轨迹。
参数	（1）ArmSocket Socket 句柄 （2）pose 位姿 (优先采用四元数表达)
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	/*pose：目标位姿，位置精度：0.001mm，姿态精度：0.001rad 目标位置：x：0m，y:0m，z：0.85049m 目标姿态：rx:0rad，ry:0rad，rz:3.142rad 目标位姿为当前工具在当前工作坐标系下的数值。*/ Pose pose; pose.position.x=0; pose.position.y=0; pose.position.z=0.85049; pose.euler.rx=0; pose.euler.ry=0; pose.euler.rz=3.142; Movep_Follow(m_sockhand,pose);

3.13. 机械臂示教

3.13.1. 关节示教 Joint_Teach_Cmd

int Joint_Teach_Cmd(SOCKHANDLE ArmSocket, byte num, byte direction, byte v, bool block);	
函数功能	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。
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。
示例	//关节 6 示教，正方向，速度 20% byte num = 6;

	byte direction = 1; byte v = 20; ret = Joint_Teach_Cmd(m_sockhand,num ,direction ,v,RM_BLOCK);

3.13.2. 位置示教 Pos_Teach_Cmd

int Pos_Teach_Cmd(SOCKHANDLE ArmSocket, POS_TEACH_MODES type, byte direction, byte v, bool block);	
函数功能	该函数用于当前坐标系下（默认为当前工作坐标系下，调用”3.13.5 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。
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线速度和线加速度的百分比 （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。
示例	//位置示教，x 轴负方向，速度 20% POS_TEACH_MODES type = X_Dir; byte direction = 0; byte v = 20; ret=Pos_Teach_Cmd(m_sockhand,type,direction,v, RM_BLOCK);

3.13.3. 姿态示教 Ort_Teach_Cmd

int Ort_Teach_Cmd(SOCKHANDLE ArmSocket, ORT_TEACH_MODES type, byte direction, byte v, bool block);	
函数功能	该函数用于当前坐标系下（默认为当前工作坐标系下，调用”3.13.5 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。
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。
示例	//姿态示教，rx 轴负方向，速度 20% ORT_TEACH_MODES type = RX_Rotate; byte direction = 0; byte v = 20; ret = Ort_Teach_Cmd(m_sockhand,type, direction, v, RM_BLOCK);

3.13.4. 示教停止 Teach_Stop_Cmd

int Teach_Stop_Cmd(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于示教停止。
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。
示例	//示教停止 ret = Teach_Stop_Cmd(m_sockhand,RM_BLOCK);

3.13.5. 切换示教运动坐标系 Set_Teach_Frame

int Set_Teach_Frame(SOCKHANDLE ArmSocket, int type, int block);	
函数功能	该函数用于切换示教参考坐标系。
参数	（1）ArmSocket Socket 句柄 （2）type 0 代表工作坐标系，1 代表工具坐标系 （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。
示例	//切换示教运动坐标系为工具坐标系 int ret = -1; int type = 1; ret = Set_Teach_Frame(m_sockhand,type,RM_BLOCK);

3.13.6. 获取示教运动坐标系 Get_Teach_Frame

int Get_Teach_Frame(SOCKHANDLE ArmSocket, int* type);	
函数功能	该函数用于切换示教运动坐标系。
参数	（1）ArmSocket Socket 句柄 （2）type 0 代表工作坐标系，1 代表工具坐标系 （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。
示例	//获取示教参考坐标系 int ret = -1; int type; ret = Get_Teach_Frame(m_sockhand,&type);

3.14. 机械臂步进

3.14.1. 关节步进 Joint_Step_Cmd

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

函数功能	该函数用于关节步进。关节在当前位置下步进指定角度。
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。
示例	//关节步进，关节 1 反方向步进 10 度，速度系数 30% byte num = 1; float step = -10; byte v = 30; ret = Joint_Step_Cmd(m_sockhand,num, step, v,RM_BLOCK);

3.14.2. 位置步进 Pos_Step_Cmd

int Pos_Step_Cmd(SOCKHANDLE ArmSocket, POS_TEACH_MODES type, float step, byte v, bool block);	
函数功能	该函数用于当前坐标系下（默认为当前工作坐标系下，调用”3.13.5 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。机械臂末端在当前坐标系下，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。
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 速度比例 1~100，即规划速度和加速度占机械臂末端最大线速度和线加速度的百分比 （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。
示例	//位置步进，y 轴负方向步进 0.1m，速度 30%

	POS_TEACH_MODES type = Y_Dir; float step = -0.1; byte v = 30; ret = Pos_Step_Cmd(m_sockhand,type, step, v,RM_BLOCK);

3.14.3. 姿态步进 Ort_Step_Cmd

int Ort_Step_Cmd(SOCKHANDLE ArmSocket, ORT_TEACH_MODES type, float step, byte v, bool block);	
函数功能	该函数用于当前坐标系下（默认为当前工作坐标系下，调用”3.13.5 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。
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。
示例	//姿态步进，y 轴负方向旋转 0.5rad，速度 30% ORT_TEACH_MODES type = RY_Rotate; float step = -0.5; byte v = 30; ret = Ort_Step_Cmd(m_sockhand, type, step, v, RM_BLOCK);

3.15. 控制器配置

3.15.1. 获取控制器状态 Get_Controller_State

int Get_Controller_State(SOCKHANDLE ArmSocket, float *voltage, float *current, float *temperature, uint16_t *sys_err);	
函数功能	该函数用于获取控制器状态。
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。
示例	//获取控制器状态 float voltage = 0; float current = 0; float temperature = 0; uint16_t sys_err = 0; ret = Get_Controller_State(m_sockhand, &voltage, &current, &temperature, &sys_err);

3.15.2. 设置 WiFi AP 模式设置 Set_WiFi_AP_Data

int Set_WiFi_AP_Data(SOCKHANDLE ArmSocket, const char *wifi_name, const char* password);	
函数功能	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。
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。
示例	//配置 wifiAP 内容，wifi 名称：robot，连接密码：12345678 char *wifi_name = (char*)"robot"; char* password = (char*)"12345678"; ret = Set_WiFi_AP_Data(m_sockhand,wifi_name,password);

3.15.3. 设置 WiFi STA 模式设置 Set_WIFI_STA_Data

int Set_WIFI_STA_Data(SOCKHANDLE ArmSocket, const char *router_name, const char* password);	
函数功能	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。
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。

示例	//配置 wifiSTA 内容，目标路由器名称：robot，路由器密码：12345678 char *router_name = (char*)"robot"; char* password = (char*)"12345678"; ret = Set_WiFI_STA_Data(m_sockhand,router_name,password);

3.15.4. 设置 UART_USB 接口波特率 Set_USB_Data

int Set_USB_Data(SOCKHANDLE ArmSocket, int baudrate);	
函数功能	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。
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。
示例	//配置 USB 波特率为 460800 int baudrate = 460800; ret = Set_USB_Data(m_sockhand,baudrate);

3.15.5. 设置 RS485 配置 Set_RS485

int Set_RS485(SOCKHANDLE ArmSocket, int baudrate);	
函 数 功 能	该函数用于控制器设置 RS485 配置。 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，断电重启后仍会使用该波特率对外通信。
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。
示例	//配置 RS485 波特率为 460800 int baudrate = 460800; ret = Set_RS485(m_sockhand,baudrate);

3.15.6. 设置机械臂电源 Set_Arm_Power

int Set_Arm_Power (SOCKHANDLE ArmSocket, bool cmd, bool block);	
函 数 功 能	该函数用于设置机械臂电源。
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。
示例	//控制机械臂上电 ret = Set_Arm_Power (m_sockhand,true, RM_BLOCK);

3.15.7. 获取机械臂电源 Get_Arm_Power_State

int Get_Arm_Power_State (SOCKHANDLE ArmSocket, int* power);	
函数功能	该函数用于获取机械臂电源
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。
示例	//读取机械臂电源状态 int power; ret = Get_Arm_Power_State (m_sockhand,&power);

3.15.8. 读取机械臂软件版本 Get_Arm_Software_Version

int Get_Arm_Software_Version(SOCKHANDLE ArmSocket, char* plan_version, char* ctrl_version, char *kernal1, char *kernal2,char *product_version);	
函数功能	该函数用于读取机械臂软件版本
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。
示例	//读取机械臂软件版本 char plan_version[128];

	char ctrl_version[128]; char kernal1[128]; char kernal2[128]; char product_version[128]; ret=Get_Arm_Software_Version(m_sockhand,plan_version,ctrl_version,kernal1,kernal2,product_version);

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

Int Get_System_Runtime (SOCKHANDLE ArmSocket, int *day, int *hour, int *min, int *sec);	
函数功能	读取控制器的累计运行时间。
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。
示例	//获取控制器的累计运行时间 char state = 0; int day; int hour; int min; int sec; ret = Get_System_Runtime (m_sockhand, &day, &hour, &min, &sec);

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

int Clear_System_Runtime(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于清空控制器累计运行时间。
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。
示例	//清空控制器累计运行时间 ret = Clear_System_Runtime(m_sockhand,RM_BLOCK);

3.15.11. 获取关节累计转动角度 Get_Joint_Odom

int Get_Joint_Odom(SOCKHANDLE ArmSocket, float* odom);	
函 数 功 能	该函数用于读取关节的累计转动角度。
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。
示例	//获取关节累计转动角度 float odom[7]; ret = Get_Joint_Odom(m_sockhand,odom);

3.15.12. 清除关节累计转动角度 Clear_Joint_Odom

int Clear_Joint_Odom(SOCKHANDLE ArmSocket, bool block);	
函 数 功 能	该函数用于清空关节累计转动角度
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。
示例	//清除关节累计转动角度 ret = Clear_Joint_Odom(m_sockhand,RM_BLOCK);

3.15.13. 配置高速网口 Set_High_Speed_Eth

int Set_High_Speed_Eth (SOCKHANDLE ArmSocket, byte num, bool block);	
函数功能	该函数用于配置高速网口。
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。
示例	//关闭高速网口 byte num = 0; ret = Set_High_Speed_Eth (m_sockhand,num, RM_BLOCK);

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

int Set_High_Ethernet(SOCKHANDLE ArmSocket, const char * ip, const char * mask, const char * gateway);	
函数功能	该函数用于设置高速网口网络配置[配置通讯内容]。
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。
示例	//设置高速网口网络配置 ret=Set_High_Ethernet(m_sockhand,(char*)"192.168.1.27",(char *)"255.255.255.0",(char *)"192.168.1.1");

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

int Get_High_Ethernet(SOCKHANDLE ArmSocket, char * ip, char * mask, char * gateway, char * mac);	
函数	该函数用于获取高速网口网络配置[配置通讯内容]

功能	
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。
示例	//获取高速网口网络配置 char after_ip[128]; char after_mask[128]; char after_gateway[128]; char mac[128]; ret=Get_High_Ethernet(m_sockhand,after_ip,after_mask,after_gateway,mac);

3.15.16. 保存参数 Save_Device_Info_All--基础系列

int Save_Device_Info_All(SOCKHANDLE ArmSocket);	
函数功能	该函数用于保存所有参数
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。
示例	//保存所有参数 int ret = -1; ret = Save_Device_Info_All(m_sockhand);

3.15.17. 配置有线网卡 IP 地址 Set_NetIP--I 系列

int Set_NetIP(SOCKHANDLE ArmSocket, const char * ip);	
函数功能	该函数用于配置有线网卡 IP 地址[-I]
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。
示例	//配置有线网卡 IP 地址 ret = Set_NetIP(m_sockhand, (char*)"192.168.1.19");

3.15.18. 查询有线网卡网络信息 Get_Wired_Net--l 系列

int Get_Wired_Net(SOCKHANDLE ArmSocket, char * ip, char * mask, char * mac);	
函数功能	该函数用于查询有线网卡网络信息[-l]
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。
示例	//查询有线网卡网络信息 char ip[128]; char mask[128]; char mac[128]; ret = Get_Wired_Net(m_sockhand,ip,mask,mac);

3.15.19. 查询无线网卡网络信息 Get_Wifi_Net--l 系列

int Get_Wifi_Net(SOCKHANDLE ArmSocket, WiFi_Info* network);	
函数功能	该函数用于查询无线网卡网络信息[-l]
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。
示例	//查询无线网卡网络信息 WiFi_Info wifi; ret = Get_Wifi_Net(handle, &wifi);

3.15.20. 恢复网络出厂设置 Set_Net_Default--l 系列

int Set_Net_Default(SOCKHANDLE ArmSocket);	
函数功能	该函数用于恢复网络出厂设置[-l]
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。
示例	//恢复网络出厂设置 int ret = -1; ret = Set_Net_Default(m_sockhand);

3.15.21. 清除系统错误代码 Clear_System_Err

int Clear_System_Err(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于清除系统错误
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。
示例	//清除系统错误代码 ret = Clear_System_Runtime(m_sockhand,RM_BLOCK);

3.15.22. 读取机械臂软件信息 Get_Arm_Software_Info

int Get_Arm_Software_Info(SOCKHANDLE ArmSocket, ArmSoftwareInfo* software_info);	
函数功能	该函数用于读取机械臂软件信息
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。
示例	//读取机械臂软件信息 ArmSoftwareInfo info; ret = Get_Arm_Software_Info(handle, &info);

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

int Set_Arm_Run_Mode(SOCKHANDLE ArmSocket, int mode);

函数功能	该函数用于设置机械臂模式(仿真/真实)
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。
示例	//设置机械臂模式为真实 ret = Set_Arm_Run_Mode(handle, 1);

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

int Get_Arm_Run_Mode(SOCKHANDLE ArmSocket, int* mode);	
函数功能	该函数用于获取机械臂模式(仿真/真实)
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。
示例	//获取机械臂模式 int mode; ret = Get_Arm_Run_Mode(handle, &mode);

3.16. 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

3.16.1. 设置 IO 状态 Set_IO_State

int Set_IO_State(SOCKHANDLE ArmSocket, int IO,byte num, bool state, bool block);	
函数功能	该函数用于配置指定 IO 输出状态。
参数	（1）ArmSocket Socket 句柄 （2）IO 指定设置数字 IO 为 0，指定模拟 IO 为 1 （3）num 指定 IO 通道号，范围 1~4

	（4）state 输入参数 true-高， false-低 （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。
示例	//设置数字 IO 1 号通道输出高电平 int IO = 0; byte num = 1; bool state = true; ret = Set_IO_State (m_sockhand,IO,num,state,RM_BLOCK);

3.16.2. 设置数字 IO 模式 Set_IO_Mode--I 系列

int Set_IO_Mode(SOCKHANDLE ArmSocket, byte io_num, byte io_mode);	
函 数 功 能	该函数用于设置数字 IO 模式[-I]。
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（六维力版本可配置），8-输入进入力只动姿态拖动模式（六维力版本可配置），9-输入进入力位姿结合拖动复用模式（六维力版本可配置），10-输入外部轴最大软限位复用模式（外部轴模式可配置），11-输入外部轴最小软限位复用模式（外部轴模式可配置）、12-输入初始位姿功能复用模式、13-输出碰撞功能复用模式。
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//设置数字 IO 1 号通道输入开始功能复用模式 byte num = 1; byte mode = 2; ret = Set_IO_Mode (m_sockhand,num,mode,RM_BLOCK);

3.16.3. 查询指定 IO 状态 Get_IO_State

int Get_IO_State(SOCKHANDLE ArmSocket, byte num, byte *state,byte *mode);	
函数功能	该函数用于查询指定 IO 状态。
参数	（1）ArmSocket Socket 句柄 （2）num

	指定数字 IO 通道号，范围 1~4 （3）state 输出参数 true-高， false-低 （4）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（六维力版本可配置），8-输入进入力只动姿态拖动模式（六维力版本可配置），9-输入进入力位姿结合拖动复用模式（六维力版本可配置），10-输入外部轴最大软限位复用模式（外部轴模式可配置），11-输入外部轴最小软限位复用模式（外部轴模式可配置）、12-输入初始位姿功能复用模式、13-输出碰撞功能复用模式。
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//查询数字 IO 输出 1 号通道状态 byte num = 1; byte state; byte mode; ret = Get_IO_State(m_sockhand,num,&state,&mode);

3.16.4. 查询所有 IO 输入状态 Get_IO_Input

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

函 数 功 能	该函数用于查询所有 IO 输入状态。
参数	（1）ArmSocket Socket 句柄 （2）DI_state 数字 IO 输入通道 1~3 状态数组地址，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。
示例	//查询所有 IO 输入状态 int DI_state[4] = {0}; float AI_voltage[4] = {0}; ret = Get_IO_Input(m_sockhand,DI_state,AI_voltage);

3.16.5. 查询所有 IO 的输出状态 Get_IO_Output

int Get_IO_Output(SOCKHANDLE ArmSocket, int *DO_state, float *AO_voltage);	
函 数 功 能	该函数用于查询所有 IO 输出状态。
参数	（1）ArmSocket Socket 句柄 （2）DO_state

	数字 IO 输出通道 1~4 状态数组地址，1-高，0-低 （3）AO_voltage 模拟 IO 输出通道 1~4 输处电压数组
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//查询所有输出状态 int DO_state[4] = {0}; float AO_voltage[4] = {0}; ret = Get_IO_Output(m_sockhand,DO_state,AO_voltage);

3.16.6. 设置电源输出 Set_Voltage--I 系列

int Set_Voltage(SOCKHANDLE ArmSocket, byte voltage_type, bool start_enable);	
函数功能	该函数用于设置控制器端电源输出。
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。
示例	//设置控制器端电源开机输出输出 24V byte type = 3;

	ret = Set_Voltage(m_sockhand, type, true);

3.16.7. 获取电源输出 Get_Voltage--I 系列

int Get_Voltage(SOCKHANDLE ArmSocket, byte * voltage_type);	
函数功能	该函数用于获取控制器端电源输出。
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。
示例	//查询电源输出状态 byte voltage_type; ret = Get_Voltage(m_sockhand,&voltage_type);

3.17. 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
数字输出：DO	2 路，参考电平与电源输出一致
数字输入：DI	2 路，参考电平与电源输出一致
模拟输出：AO	1 路，输出电压 0~10V

模拟输入：AI	1 路，输入电压 0~10V
通讯接口	1 路，可配置为 RS485/RS232/CAN

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

int Set_Tool_DO_State(SOCKHANDLE ArmSocket, byte num, bool state, bool block);	
函数功能	该函数用于配置工具端指定数字 IO 输出状态。
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。
示例	//设置工具端 1 号通道输出高电平 byte num = 1; bool state = true; ret = Set_Tool_DO_State(m_sockhand,num,state,RM_BLOCK);

3.17.2. 设置工具端数字 IO 模式 Set_Tool_IO_Mode

int Set_Tool_IO_Mode(SOCKHANDLE ArmSocket, byte num, bool state,bool block);	
函数功能	该函数用于设置数字输入输出 IO 模式。
参数	（1）ArmSocket Socket 句柄 （2）num 指定数字输入通道号，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。
示例	//设置工具端 IO2 号数字通道为输出模式 byte num = 2; bool state = 1; ret = Set_Tool_IO_Mode(m_sockhand,num,state,RM_BLOCK);

3.17.3. 查询工具端数字 IO 状态 Get_Tool_IO_State

int Get_Tool_IO_State(SOCKHANDLE ArmSocket, float* IO_Mode, float *IO_state);	
函数功能	该函数用于查询工具端数字 IO 状态。
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。
示例	//查询工具端数字 IO 状态 float IO_Mode[2]; float IO_state[2]; ret = Get_Tool_IO_State(IO_Mode,IO_state)

3.17.4. 设置工具端电源输出 Set_Tool_Voltage

int Set_Tool_Voltage(SOCKHANDLE ArmSocket, byte type, bool block);	
函数功	该函数用于设置工具端电源输出。

能	
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。
示例	//设置工具端电源输出类型 5V byte type = 1; ret = Set_Tool_Voltage(m_sockhand, type, RM_BLOCK);

3.17.5. 获取工具端电源输出 Get_Tool_Voltage

int Get_Tool_Voltage(SOCKHANDLE ArmSocket, byte *voltage);	
函 数 功能	该函数用于获取工具端电源输出。
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。

示例	//获取工具端电源输出 byte voltage; ret = Get_Tool_Voltage(m_sockhand,&voltage);

3.18. 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。

3.18.1. 配置手爪的开口度 Set_Gripper_Route

int Set_Gripper_Route(SOCKHANDLE ArmSocket, int min_limit, int max_limit, bool block);	
函数功能	该函数用于配置手爪的开口度。
参数	（1）ArmSocket Socket 句柄 （2）min_limit 手爪开口最小值，范围：0~1000，无单位量纲 （3）Max_limit 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。
示例	//设置手爪开口最小值 70，最大值 500 int min_limit = 70; int max_limit = 500; ret=Set_Gripper_Route(m_sockhand,min_limit,max_limit,RM_BLOCK);

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

int Set_Gripper_Release (SOCKHANDLE ArmSocket, int speed, bool block, int timeout);	
函数功能	该函数用于控制手爪以指定速度张开到最大开口处
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，不接收夹爪到位指令；RM_BLOCK-阻塞，等待控制器返回夹爪到位指令。 （4）timeout 非阻塞模式：0-发送后立即返回；其他值-接收设置成功指令后返回；

	阻塞模式：等待夹爪到位指令超时时间，单位：秒；
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//手爪以 500 的速度松开，10s 无返回则超时 int speed =500; ret = Set_Gripper_Release (m_sockhand, speed, RM_BLOCK，10);

3.18.3. 设置夹爪夹取 Set_Gripper_Pick

int Set_Gripper_Pick(SOCKHANDLE ArmSocket, int speed, int force, bool block, int timeout);	
函数功能	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。
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，不接收夹爪到位指令；RM_BLOCK-阻塞，等待控制器返回夹爪到位指令。 （5）timeout

	非阻塞模式：0-发送后立即返回；其他值-接收设置成功指令后返回； 阻塞模式：等待夹爪到位指令超时时间，单位：秒；
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//设置夹取速度为 500，力矩阈值 200,10s 无返回则超时 int speed = 500; int force = 200; ret = Set_Gripper_Pick(m_sockhand,speed,force,RM_BLOCK,10);

3.18.4. 设置夹爪持续夹取 Set_Gripper_Pick_On

int Set_Gripper_Pick_On(SOCKHANDLE ArmSocket, int speed, int force, bool block, int timeout);	
函数功能	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。
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，不接收夹爪到位指令；RM_BLOCK-阻塞，等待控制器返回夹爪到位指令。 （5）timeout 非阻塞模式：0-发送后立即返回；其他值-接收设置成功指令后返回； 阻塞模式：等待夹爪到位指令超时时间，单位：秒；
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//设置夹取速度 500，夹取力矩阈值 200，10s 无返回则超时。 int speed = 500; int force = 200; ret = Set_Gripper_Pick_On(m_sockhand,speed,force,RM_BLOCK,10);

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

int Set_Gripper_Position (SOCKHANDLE ArmSocket, int position, bool block, int timeout);	
函数功能	该函数用于控制手爪到达指定开口度位置。
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，不接收夹爪到位指令；RM_BLOCK-阻塞，等待控制器返回夹爪到位指令。 （4）timeout 非阻塞模式：0-发送后立即返回；其他值-接收设置成功指令后返回； 阻塞模式：等待夹爪到位指令超时时间，单位：秒；
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//控制手爪到达 500 开口度，10s 无返回则超时 int position = 500; ret = Set_Gripper_Position(m_sockhand,position,RM_BLOCK,10);

3.18.6. 获取夹爪状态 Get_Gripper_State

int Get_Gripper_State(SOCKHANDLE ArmSocket, GripperState* gripper_state);	
函数功能	该函数用于获取夹爪状态。
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。
示例	//获取夹爪状态

	GripperState state; ret = Get_Gripper_State(handle, &state);
备注	此接口需升级夹爪最新固件方可使用！！！

3.19. 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。

3.19.1. 进入拖动示教模式 Start_Drag_Teach

int Start_Drag_Teach (SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于控制机械臂进入拖动示教模式
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。
示例	//控制机械臂进入拖动示教模式 ret = Start_Drag_Teach(m_sockhand,RM_BLOCK);

3.19.2. 退出拖动示教模式 Stop_Drag_Teach

int Stop_Drag_Teach (SOCKHANDLE ArmSocket, bool block);

函数功能	该函数用于控制机械臂退出拖动示教模式
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。
示例	//退出拖动示教模式 ret = Stop_Drag_Teach(m_sockhand,RM_BLOCK);

3.19.3. 拖动示教轨迹复现 Run_Drag_Trajectory

int Run_Drag_Trajectory (SOCKHANDLE ArmSocket, bool block);	
函数功能	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，请先调用运动到轨迹起点函数，否则会返回报错信息。
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。

示例	//复现拖动示教轨迹 ret = Run_Drag_Trajectory (m_sockhand,1);

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

int Pause_Drag_Trajectory (SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于控制机械臂在轨迹复现过程中的暂停。
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。
示例	//轨迹复现暂停 ret = Pause_Drag_Trajectory (m_sockhand,RM_BLOCK);

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

int Continue_Drag_Trajectory (SOCKHANDLE ArmSocket, bool block);	
函数功能	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。
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。
示例	//轨迹复现继续 ret = Continue_Drag_Trajectory (m_sockhand,RM_BLOCK);

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

int Stop_Drag_Trajectory (SOCKHANDLE ArmSocket, bool block);	
函数功能	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。
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。
示例	//轨迹复现停止 ret = Stop_Drag_Trajectory (m_sockhand,RM_BLOCK);

3.19.7. 运动到轨迹起点 Drag_Trajectory_Origin

int Drag_Trajectory_Origin (SOCKHANDLE ArmSocket, bool block);	
函数功能	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。
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。
示例	//运动到轨迹起点 ret =Drag_Trajectory_Origin (m_sockhand,RM_BLOCK);

3.19.8. 复合模式拖动示教 Start_Multi_Drag_Teach

int Start_Multi_Drag_Teach(SOCKHANDLE ArmSocket, int mode, int singular_wall, bool block);	
函数功能	该函数用于复合模式拖动示。
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。
示例	//使用末端六维力，只动位置，不开启拖动奇异墙 int mode = 1; ret = Start_Multi_Drag_Teach(m_sockhand,mode,0,RM_BLOCK);

3.19.9. 复合模式拖动示教-新参数 Start_Multi_Drag_Teach_New

int Start_Multi_Drag_Teach_New(SOCKHANDLE ArmSocket, MultiDragTeach teach_state);	
函数功能	该函数用于复合模式拖动示教。
参数	（1）ArmSocket Socket 句柄 （2）teach_state 复合拖动示教参数

返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	MultiDragTeach param = { {0,0,0,0,0,0},// 所有轴都不可拖动 0,// 使用工作坐标系 0// 不开启拖动奇异墙 }; param.free_axes[0] = 1; // 参考坐标系 X 轴方向可拖动 ret = Start_Multi_Drag_Teach_New(handle, param);

3.19.10. 设置电流环拖动示教灵敏度 Set_Drag_Teach_Sensitivity

int Set_Drag_Teach_Sensitivity(SOCKHANDLE ArmSocket, int grade);	
函数功能	该函数用于设置电流环拖动示教灵敏度。
参数	（1）ArmSocket Socket 句柄 （2）grade 等级，0 到 100，表示 0~100%，当设置为 100 时保持初始状态
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	ret = Set_Drag_Teach_Sensitivity(handle, 100);

3.19.11. 获取电流环拖动示教灵敏度 Get_Drag_Teach_Sensitivity

int Get_Drag_Teach_Sensitivity(SOCKHANDLE ArmSocket, int *grade);

函数功能	该函数用于获取电流环拖动示教灵敏度。
参数	（1）ArmSocket Socket 句柄 （2）grade 等级，0 到 100，表示 0~100%，当设置为 100 时保持初始状态
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	int grade; ret = Get_Drag_Teach_Sensitivity(handle, &grade);

3.19.12. 保存拖动示教轨迹 Save_Trajectory

int Save_Trajectory(SOCKHANDLE ArmSocket, char * filename, int* num);	
函数功能	该函数用于保存拖动示教轨迹。
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
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//保存拖动示教轨迹 int num;

	ret = Save_Trajectory(m_sockhand, (char *)"H:/123.txt", &num);

3.19.13. 设置力位混合控制 Set_Force_Postion

int Set_Force_Postion(SOCKHANDLE ArmSocket, int sensor, int mode, int direction, int N, bool block);	
函数功能	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。
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。
示例	//设置六维力基坐标系 Z 轴力控，1N 力大小 ret = Set_Force_Postion (m_sockhand, 1, 0, 2, 10, RM_BLOCK);
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。

3.19.14. 设置力位混合控制-新参数 Set_Force_Postion_New

int Set_Force_Postion_New(SOCKHANDLE ArmSocket, ForcePosition param);	
函数功能	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。
参数	（1）ArmSocket socket 句柄 （2）param 力位混合控制参数
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	ForcePosition force = { 1, 0, {0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0} };

	Ret = Set_Force_Postion_New(handle, force);
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。

3.19.15. 结束力位混合控制 Stop_Force_Postion

int Stop_Force_Postion (SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于结束力位混合控制.
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。
示例	//结束力位混合控制 ret = Stop_Force_Postion_Move(m_sockhand,RM_BLOCK);

3.20. 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。

3.20.1. 获取六维力数据 Get_Force_Data

int Get_Force_Data(SOCKHANDLE ArmSocket, float *Force,float *zero_force);	
函 数 功 能	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。
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
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//获取六维力数据 float Force[6]; float zero_force[6]; ret = Get_Force_Data(m_sockhand,Force,zero_force);

3.20.2. 清空六维力数据 Clear_Force_Data

int Clear_Force_Data(SOCKHANDLE ArmSocket, bool block);	
函 数 功	该函数用具清空六维力数据，即后续获得的所有数据都是基于当前数

能	据的偏移量。。
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。
示例	//清空六维力数据 ret = Clear_Force_Data(m_sockhand,RM_BLOCK);

3.20.3. 设置六维力重心参数 Set_Force_Sensor

int Set_Force_Sensor (SOCKHANDLE ArmSocket);	
函数功能	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。
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。
示例	//设置六维力重心参数 ret = Set_Force_Sensor (m_sockhand);

3.20.4. 手动标定六维力数据 Manual_Set_Force

int Manual_Set_Force (SOCKHANDLE ArmSocket, int type,float *joint);	
函 数 功能	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
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
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//手动标定六维力数据,即第一个位置关节 1~7 的目标角度依次为 0°，1°，2°，3°，4°，5°，6° float joint1[6] = {0,1,2,3,4,5}; ret = Manual_Set_Force (m_sockhand,1,joint1);

3.20.5. 退出标定流程 Stop_Set_Force_Sensor

int Stop_Set_Force_Sensor (SOCKHANDLE ArmSocket, bool block);	
函 数 功	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

能	动，退出标定流程。
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。
示例	//退出六维力标定流程 ret = Stop_Set_Force_Sensor (m_sockhand,RM_BLOCK);

3.21. 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。

3.21.1. 设置灵巧手手势序号 Set_Hand_Posture

int Set_Hand_Posture (SOCKHANDLE ArmSocket, int posture_num, bool block);	
函数功能	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。
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。
示例	//设置灵巧手执行 1 号手势 int posture_num = 1; ret = Set_Hand_Posture(m_sockhand,posture_num,RM_BLOCK);

3.21.2. 设置灵巧手动作序列序号 Set_Hand_Seq

int Set_Hand_Seq (SOCKHANDLE ArmSocket, int seq_num, bool block);	
函数功能	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。
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。
示例	//设置灵巧手执行 1 号动作序列

	int seq_num = 1; ret = Set_Hand_Seq(m_sockhand,seq_num,RM_BLOCK);

3.21.3. 设置灵巧手角度 Set_Hand_Angle

int Set_Hand_Angle(SOCKHANDLE ArmSocket, const int *angle, bool block);	
函数功能	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。
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。
示例	//设置灵巧手各手指动作 const int angle[6]= {-1,100,200,300,400,500}; ret = Set_Hand_Angle(m_sockhand,angle,RM_BLOCK);

3.21.4. 设置灵巧手各关节速度 Set_Hand_Speed

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

函数功能	设置灵巧手各关节速度
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。
示例	//设置灵巧手各手指速度 int speed = 500; ret = Set_Hand_Speed (m_sockhand,speed, RM_BLOCK);

3.21.5. 设置灵巧手各关节力阈值 Set_Hand_Force

int Set_Hand_Force (SOCKHANDLE ArmSocket, int force, bool block);	
函数功能	设置灵巧手各关节力阈值。
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。
示例	//设置灵巧手力阈值 500 int force = 500; ret = Set_Hand_Force(m_sockhand,force,RM_BLOCK);

3.22. 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。

3.22.1. 查询一维力数据 Get_Fz

int Get_Fz(SOCKHANDLE ArmSocket, float *Fz,float *zero_Fz, float *work_zero, float *tool_zero);	
函数功能	该函数用于查询末端一维力数据。
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。
示例	//查询末端一维力数据 float force; ret = Get_Fz(m_sockhand,&force);
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。

3.22.2. 清空一维力数据 Clear_Fz

int Clear_Fz(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于清零末端一维力数据。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。
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。
示例	//清空末端一维力数据

	ret = Clear_Fz(m_sockhand,RM_BLOCK);

3.22.3. 自动标定末端一维力数据 Auto_Set_Fz

int Auto_Set_Fz(SOCKHANDLE ArmSocket);	
函数功能	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
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。
示例	//自动标定末端一维力 ret = Auto_Set_Fz(m_sockhand);

3.22.4. 手动标定末端一维力数据 Manual_Set_Fz

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

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。
示例	//手动标定末端一维力数据 float joint[7]={0,0,0,0,0,0,0}; float joint1[7]={0,0,90,0,0,90,0}; ret = Manual_Set_Fz(m_sockhand,joint,joint1);

3.23. Modbus RTU 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。

3.23.1. 设置通讯端口 Modbus RTU 模式 Set_Modbus_Mode

int Set_Modbus_Mode (SOCKHANDLE ArmSocket, int port,int baudrate,int timeout,bool block);	
函数功能	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。
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 四种常见波特率 （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。
示例	//配置通讯端口为末端接口板 RS485 端口，波特率配置 115200，超时时间 2 百毫秒 int port = 1; int baudrate = 115200; int timeout = 2; ret=Set_Modbus_Mode(m_sockhand,port,baudrate,timeout,RM_BLOCK);

3.23.2. 关闭通讯端口 Modbus RTU 模式 Close_Modbus_Mode

int Close_Modbus_Mode (SOCKHANDLE ArmSocket, int port , bool block);	
函数功能	该函数用于关闭通讯端口 Modbus RTU 模式。
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。
示例	//关闭控制器 RS485 通讯端口

	int port = 0; ret = Close_Modbus_Mode (m_sockhand,port,RM_BLOCK);

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

int Set_Modbustcp_Mode(SOCKHANDLE ArmSocket, const char* ip, int port, int timeout);	
函数功能	该函数用于配置连接 ModbusTCP 从站。
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。
示例	//配置 ret = Set_Modbustcp_Mode(m_sockhand, (char*)"192.168.1.88", 502, 2000);

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

int Close_Modbustcp_Mode(SOCKHANDLE ArmSocket);

函数功能	该函数用于配置关闭 ModbusTCP 从站。
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。
示例	//关闭 ModbusTCP 从站 ret = Close_Modbustcp_Mode (m_sockhand);

3.23.5. 读线圈 Get_Read_Coils

int Get_Read_Coils (SOCKHANDLE ArmSocket, int port, int address, int num, int device, int* coils_data);	
函数功能	该函数用于读线圈。
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。
示例	//通讯端口为末端接口板 RS485 端口，读线圈 int port = 1; int address = 10; int num = 1; int device = 2; int coils_data; ret = Get_Read_Coils (m_sockhand,port, address, num, device, &coils_data);

3.23.6. 读离散输入量 Get_Read_Input_Status

int Get_Read_Input_Status(SOCKHANDLE ArmSocket, int port,int address,int num,int device,int* coils_data);	
函数功能	读离散输入量。
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。
示例	//通讯端口为末端接口板 RS485 端口，读离散输入量 int port = 1; int address = 10; int num = 2; int device = 2; int coils_data; ret = Get_Read_Input_Status (m_sockhand, port, address, num, device, &coils_data);

3.23.7. 读保持寄存器 Get_Read_Holding_Registers

int Get_Read_Holding_Registers(SOCKHANDLE ArmSocket, int port,int address,int device,int* coils_data);	
函数功能	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。
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。
示例	//通讯端口为末端接口板 RS485 端口，读保持寄存器 int port = 1; int address = 10;

	int device = 2; int coils_data; ret = Get_Read_Holding_Registers (m_sockhand, port, address, device, &coils_data);

3.23.8. 读输入寄存器 Get_Read_Input_Registers

int Get_Read_Input_Registers(SOCKHANDLE ArmSocket, int port,int address,int device,int* coils_data);	
函数功能	该函数用于读输入寄存器。
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。
示例	//通讯端口为末端接口板 RS485 端口，读输入寄存器

	int port = 1; int address = 10; int device = 2; int coils_data; ret = Get_Read_Input_Registers (m_sockhand, port, address, device, &coils_data);

3.23.9. 写单圈数据 Write_Single_Coil

int Write_Single_Coil(SOCKHANDLE ArmSocket, int port,int address,int data,int device,bool block);	
函数功能	该函数用于写单圈数据。
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 data 要写入线圈的数据 （4）device 外设设备地址

	（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。
示例	//通讯端口为末端接口板 RS485 端口，写单圈数据 int port = 1; int address = 10; int data = 1000; int device = 2; ret = Write_Single_Coil (m_sockhand, port, address, data, device, RM_BLOCK);

3.23.10. 写单个寄存器 Write_Single_Register

int Write_Single_Register(SOCKHANDLE ArmSocket, int port,int address,int data,int device,bool block);	
函数功能	该函数用于写单个寄存器。
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。
示例	//通讯端口为末端接口板 RS485 端口，写单个寄存器 int port = 1; int address = 10; int data = 1000; int device = 2; ret = Write_Single_Register (m_sockhand, port, address, data, device, RM_BLOCK);

3.23.11. 写多个寄存器 Write_Registers

int Write_Registers(SOCKHANDLE ArmSocket, int port,int address,int num,byte *single_data, int device, bool block);	
函数功能	该函数用于写多个寄存器。
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。
示例	//通讯端口为末端接口板 RS485 端口，写单个寄存器 int port = 1; int address = 10; int num = 2; int device = 2; byte single_data[] = {15, 20, 25, 30};

	ret=Write_Registers(m_sockhand,port,address,num,single_data,device,RM_BLOCK);

3.23.12. 写多圈数据 Write_Coils

int Write_Coils(SOCKHANDLE ArmSocket, int port,int address,int num, byte * coils_data,int device,bool block);	
函数功能	该函数用于写多圈数据。
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。
示例	//通讯端口为末端接口板 RS485 端口，写单圈数据 int port = 1; int address = 10; int num = 16; int device = 2; byte coils_data[] = {15,20}; ret=Write_Coils(m_sockhand,port,address,num,coils_data,device, RM_BLOCK);

3.23.13. 读多圈数据 Get_Read_Multiple_Coils

int Get_Read_Multiple_Coils(SOCKHANDLE ArmSocket, int port,int address,int num,int device,int *coils_data);	
函数功能	该函数用于读多圈数据。
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。
示例	//通讯端口为末端接口板 RS485 端口，读多个线圈 int port = 1; int address = 10; int num = 16; int device = 2; int coils_data; ret=Get_Read_Multiple_Coils(m_sockhand,port,address,num,device,&after_coils_data);

3.23.14. 读多个保持寄存器 Read_Multiple_Holding_Registers

int Read_Multiple_Holding_Registers(SOCKHANDLE ArmSocket, byte port, int

address,byte num, int device, int8_t *coils_data);	
函数功能	该函数用于读多个保持寄存器。
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 返回线圈状态
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//通讯端口为末端接口板 RS485 端口，读多个保持寄存器 int port = 1; int address = 10; int num = 2; int device = 2;

	int coils_data; ret=Read_Multiple_Holding_Registers(m_sockhand,port,address,num,device,&coils_data);

3.23.15. 读多个输入寄存器 Read_Multiple_Input_Registers

int Read_Multiple_Input_Registers(SOCKHANDLE ArmSocket, byte port, int address,byte num, int device, int8_t *coils_data);	
函数功能	该函数用于读多个保持寄存器。
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 返回线圈状态

返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	//通讯端口为末端接口板 RS485 端口，读多个输入寄存器 int port = 1; int address = 10; int num = 2; int device = 2; int coils_data[4]; ret=Read_Multiple_Input_Registers(m_sockhand,port,address,num,device,coils_data);

3.24. 升降机构

睿尔曼机械臂可集成自主研发升降机构。

3.24.1. 升降机构速度开环控制 Set_Lift_Speed

Set_Lift_Speed (SOCKHANDLE ArmSocket, int speed);	
函数功能	设置升降机构速度开环控制。
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。
示例	//设置移动平台移动速度 50%，向下运动 int speed = -50; ret = Set_Lift_Speed(m_sockhand,speed);

3.24.2. 设置升降机构高度 Set_Lift_Height

int Set_Lift_Height(SOCKHANDLE ArmSocket, int height,int speed,bool block);	
函数功能	该函数用于设置升降机构高度。
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。
示例	//设置目标高度 100mm，升降速度 50% int height = 100; int speed = 50;

	ret = Set_Lift_Height(m_sockhand,height,speed,RM_BLOCK);

3.24.3. 获取升降机构状态 Get_Lift_State

int Get_Lift_State(SOCKHANDLE ArmSocket, int* height,int* current,int* err_flag, int *mode);	
函数功能	该函数用于获取升降机构状态。
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。
示例	//获取升降机构状态 int height; int current;

	int err_flag; Int mode; ret = Get_Lift_State(m_sockhand,&height,&current,&err_flag.&mode);

3.25. 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。

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

int Start_Force_Position_Move (SOCKHANDLE ArmSocket, bool block);	
函数功能	开启透传力位混合控制补偿模式。
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。
示例	//开启透传力位混合控制补偿模式

	ret = Start_Force_Position_Move(m_sockhand,RM_BLOCK);

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

int Force_Position_Move_Joint(SOCKHANDLE ArmSocket, const float *joint,byte sensor,byte mode,int dir,float force, bool follow);	
函数功能	该函数用于力位混合控制补偿透传模式（关节角度）。
参数	（1）ArmSocket Socket 句柄 （2）joint 目标关节角度 （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。

示例	//透传力位混合补偿--关节角度 const float joint[6] = {1,2,3,4,5,6}; byte sensor = 0; byte mode = 0; int dir = 2; float force = 15; bool follow = true; ret=Force_Position_Move_Joint(m_sockhand,joint,sensor,mode,dir,force,follow);
备注	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

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

int Force_Position_Move_Pose(SOCKHANDLE ArmSocket, Pose pose,byte sensor,byte mode,int dir,float force, bool follow);	
函数功能	该函数用于力位混合控制补偿透传模式（位姿）。
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。
示例	//透传力位混合补偿--位姿 byte sensor = 0; byte mode = 0; int dir = 2; float force = 15; bool follow = true; ret=Force_Position_Move_Pose(m_sockhand,pose,sensor,mode,d

	ir,force,follow);
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

3.25.4. 力位混合控制补偿透传模式-新参数 Force_Position_Move

int Force_Position_Move(SOCKHANDLE ArmSocket, ForcePositionMove param);	
函数功能	该函数用于力位混合控制补偿透传模式（位姿）。
参数	（1）ArmSocket Socket 句柄 （2）param 透传力位混合补偿参数
返回值	成功返回：0；失败返回：错误码， rm_define.h 查询。
示例	ForcePositionMove move = { 1,{0,0,0},{0,0,0,0,0,0},1,1,true,{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0} };

	ret = Force_Position_Move(handle, move);
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

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

int Stop_Force_Position_Move(SOCKHANDLE ArmSocket, bool block);	
函数功能	该函数用于关闭透传力位混合控制补偿模式。
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。
示例	//关闭透传力位混合控制补偿模式 ret = Stop_Force_Position_Move(m_sockhand,RM_BLOCK);

3.26. 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。

3.26.1. 初始化算法依赖数据 Algo_Init_Sys_Data

void Algo_Init_Sys_Data(RobotType dMode, SensorType bType);	
函数功能	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)。
参数	（1）dMode 机械臂型号 （2）bType 传感器型号
示例	//初始化算法依赖数据 RobotType dmode = RM65; SensorType rbt_type = _B; Algo_Init_Sys_Data(dmode,rbt_type);

3.26.2. 设置算法的安装角度 Algo_Set_Angle

void Algo_Set_Angle(float x, float y, float z);	
函数功能	设置算法的安装角度参数。
参数	（1）x X 轴安装角度，单位度。 （2）y Y 轴安装角度，单位度。 （3）z Z 轴安装角度，单位度。
示例	//设置算法的安装角度 float x = 0; float y = 0; float z = 0; Algo_Set_Angle(x, y, z);

3.26.3. 获取算法的安装角度 Algo_Get_Angle

Algo_Get_Angle(float* x, float* y, float* z);	
函数功能	设置算法的安装角度参数。
参数	（1）x X 轴安装角度，单位度。 （2）y

	Y 轴安装角度，单位度。 （3）z Z 轴安装角度，单位度。
示例	//获取算法的安装角度 float x,y,z; Algo_Get_Angle(&x,&y,&z);

3.26.4. 设置算法工作坐标系 Set_Algo_WorkFrame_Params

Algo_Set_WorkFrame(const FRAME* const coord_work);	
函数功能	设置算法工作坐标系。
参数	（1）coord_work 工作坐标系数据
示例	FRAME coord_work; strcpy(coord_work.frame_name.name,"123"); coord_work.pose.position.x = 1; coord_work.pose.position.y = 12; coord_work.pose.position.z = 123; coord_work.pose.euler.rx = 0.5; coord_work.pose.euler.ry = 1; coord_work.pose.euler.rz = 1.5; Algo_Set_WorkFrame(&coord_work);

3.26.5. 获取当前工作坐标系

Algo_Get_Curr_WorkFrame(FRAME* coord_work);	
函数功能	设置算法工作坐标系。
参数	（1）coord_work 工作坐标系数据

示例	FRAME current_work; Algo_Get_Curr_WorkFrame(&current_work);

1、

3.26.1. 设置算法工具坐标系和负载 Set_Algo_ToolFrame_Params

Algo_Set_ToolFrame(const FRAME* const coord_tool);	
函数功能	设置算法工具坐标系和负载。
参数	（1）coord_tool 工具坐标系数据
示例	FRAME coord_tool; strcpy(coord_tool.frame_name.name,"123"); coord_tool.pose.position.x = 1; coord_tool.pose.position.y = 12; coord_tool.pose.position.z = 123; coord_tool.pose.euler.rx = 0.5; coord_tool.pose.euler.ry = 1; coord_tool.pose.euler.rz = 1.5; coord_tool.payload = 5; coord_tool.x = 1; coord_tool.y = 1; coord_tool.z = 1; Algo_Set_ToolFrame(&coord_tool);

3.26.6. 获取算法当前工具坐标系

Algo_Get_Curr_ToolFrame(FRAME* coord_tool);	
函数功能	设置算法工具坐标系和负载。
参数	（1）coord_tool 坐标系数据
示例	FRAME current_tool; Algo_Get_Curr_ToolFrame(&current_tool);

3.26.7. 正解 Forward_Kinematics

Pose Algo_Forward_Kinematics(const float* const joint);	
函数功能	用于睿尔曼机械臂正解计算。
参数	（1）joint 关节 1 到关节 7 角度，单位度。
返回值	正解结果
示例	//机械臂正解计算 float joint[6] = {0,0,90,0,90,0}; pose = Algo_Forward_Kinematics(joint);

3.26.8. 设置逆解求解模式 Algo_Set_Redundant_Parameter_Traversal_Mode

void Algo_Set_Redundant_Parameter_Traversal_Mode(bool mode);	
函数功能	用于设置逆解求解模式。
参数	（1）mode true：遍历模式，冗余参数遍历的求解策略。适于当前位姿跟要求解的位姿差别特别大的应用场景，如 MOVJ_P、位姿编辑等，耗时较长 false：单步模式，自动调整冗余参数的求解策略。适于当前位姿跟要求解的位姿差别特别小、连续周期控制的场景，如笛卡尔空间规划的位姿求解等，耗时短
返回值	SYS_NORMAL：计算正常，CALCULATION_FAILED：计算失败；

示例	Algo_Set_Redundant_Parameter_Traversal_Mode(true);

3.26.9. 逆解 Algo_Inverse_Kinematics

int Algo_Inverse_Kinematics(const float* const q_in, const Pose* const q_pose,float* q_out, uint8_t flag);	
函数功能	用于睿尔曼机械臂逆解计算，默认单步模式，可使用 Algo_Set_Redundant_Parameter_Traversal_Mode 接口设置逆解求解模式。
参数	（1）q_in 上一时刻关节角，单位度。 （2）q_pose 目标位姿。 （3）q_out 输出的关节角度，单位度。 （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；
示例	//逆解计算 float joint[6]={20.68,17.788,73.086,30.004,88.904,120.152}; float joint1[6]; Pose pose; float joint2[6] = {20,20,70,30,90,120}; pose = Algo_Forward_Kinematics(joint2); int ret; ret = Algo_Inverse_Kinematics(joint,&pose,joint1,1);

3.26.10. 逆解 Algo_Inverse_Kinematics_Wrap

int Algo_Inverse_Kinematics_Wrap(const IK_Params* params);	
函数功能	用于睿尔曼机械臂逆解计算。该接口与 Algo_Inverse_Kinematics 算法接口作用相同，不同的是将 Algo_Inverse_Kinematics 的参数封装成结构体，以使它更易用。
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；
示例	//逆解计算 `const float q_in_data[7] = {20,30,70,30,90,120,0};` `uint8_t flag = 1;` `Pose q_pose_data;` `q_pose_data.position.x = -0.250113f;` `q_pose_data.position.y = -0.182819f;` `q_pose_data.position.z = 0.331672f;` `q_pose_data.euler.rx = -2.86056f;` `q_pose_data.euler.ry = -0.447832f;` `q_pose_data.euler.rz = -1.80973f;` `IK_Params params;` `params.q_in = q_in_data;` `params.q_pose = &q_pose_data;` `params.flag = flag;` `int result = Algo_Inverse_Kinematics_Wrap(&params);`

3.26.11. 计算环绕运动位姿 Algo_RotateMove

Pose Algo_RotateMove(const float* const curr_joint, int rotate_axis, float rotate_angle, Pose choose_axis);	
函数功能	用于计算环绕运动位姿。
参数	（1）curr_joint 当前关节角度，单位度。 （2）rotate_axis 旋转轴：1：x 轴， 2：y 轴，3：z 轴 （3）rotate_angle 旋转角度：旋转角度，单位（度） （4）choose_axis 指定计算时使用的坐标系
返回值	计算位姿结果
示例	// 计算在 frame 坐标系下环绕 X 轴旋转 10 度后的位姿 float joint[6] = {0,0,90,0,90,0}; Pose frame; frame.position.x=0; frame.position.y=0; frame.position.z=0; frame.euler.rx = 0; frame.euler.ry = 0;

	frame.euler.rz = 0; Pose pose; float rotateAngle = 10; pose = Algo_RotateMove(joint,1,rotateAngle ,frame); printf("POSE: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positio n.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

3.26.12. 末端位姿转成工具位姿 end2tool

Pose Algo_End2Tool(Pose eu_end);	
函数功能	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
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	工具位姿
示例	Pose pose; Pose eu_end; eu_end.position.x = -0.259256f; eu_end.position.y = -0.170727f; eu_end.position.z = 0.35621f; eu_end.euler.rx = -2.85993f; eu_end.euler.ry = -0.447394f; eu_end.euler.rz = -1.81038f;

	pose = Algo_End2Tool(eu_end); printf("Pose: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positi on.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

3.26.13. 工具位姿转末端位姿 tool2end

Pose Algo_Tool2End(Pose eu_tool);	
函数功能	末端位姿转成工具位姿。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
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	末端位姿
示例	Pose pose; Pose eu_tool; eu_tool.position.x = -0.17391f; eu_tool.position.y = 0.437109f; eu_tool.position.z = -0.21619f; eu_tool.euler.rx = 2.741f; eu_tool.euler.ry = -0.244002f; eu_tool.euler.rz = 2.938f; pose = Algo_Tool2End(eu_tool); printf("POSE: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positio n.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

3.26.14. 四元数转欧拉角 Algo_Quaternion2Euler

Euler Algo_Quaternion2Euler(Quat qua);	
函数功能	四元数转欧拉角 。
参数	（1） qua 四元数
返回值	欧拉角
示例	Quat quat; Euler eu; quat.w = -0.0882345f; quat.x = -0.62068f; quat.y = 0.740123f; quat.z = -0.243289f; eu = Algo_Quaternion2Euler(quat); qDebug()<<"欧拉角： "<<eu .rx<<eu .ry<<eu .rz;

3.26.15. 欧拉角转四元数 euler2quaternion

Quat Algo_Euler2Quaternion(Euler eu);	
函数功能	欧拉角转四元数。
参数	（1）eu 欧拉角

返回值	四元数
示例	Quat quat; Euler eu; eu.rx = -2.85993f; eu.ry = -0.447394f; eu.rz = -1.81038f; quat = Algo_Euler2Quaternion(eu); qDebug()<<"四元数："<<quat .x<<quat .y<<quat .z<<quat .w;

3.26.16. 欧拉角转旋转矩阵 Algo_Euler2Matrix

Matrix Algo_Euler2Matrix(Euler _rot3);	
函数功能	欧拉角 rx,ry,rz 转换成旋转矩阵（3*3）
参数	（1）_rot3 输入欧拉角
返回值	旋转矩阵
示例	//欧拉角转旋转矩阵 Euler _rot3; Matrix matrix; _rot3.rx = -2.85993f; _rot3.ry = -0.447394f; _rot3.rz = -1.81038f;

	matrix = Algo_Euler2Matrix(_rot3);

3.26.17. 位姿转旋转矩阵 Algo_Pos2Matrix

Matrix Algo_Pos2Matrix(Pose _point);	
函数功能	位姿转旋转矩阵。
参数	（1）_point 位姿（x、y、z、rx、ry、rz)
返回值	旋转矩阵(4*4)
示例	//位姿转旋转矩阵 Pose _point; Matrix matrix; _point.position.x = -0.177347f; _point.position.y = 0.438112f; _point.position.z = -0.215102f; _point.euler.rx = 2.09078f; _point.euler.ry = 0.942362f; _point.euler.rz = 2.39144f; matrix = Algo_Pos2Matrix(_point);

3.26.18. 旋转矩阵转位姿 Algo_Matrix2Pos

Pose Algo_Matrix2Pos(Matrix _matPos);	
函数功能	旋转矩阵转位姿。

参数	（1）_matPos 位姿的齐次变换矩阵
返回值	位姿
示例	Matrix _matPos; _matPos.irow = 4; _matPos.iline = 4; float point[4][4] = {{1.0, 0.0, 0.0, 10.0},{0.0, 1.0, 0.0, 20.0},{0.0, 0.0, 1.0, 30.0},{0.0, 0.0, 0.0, 1.0}}; for(int i = 0; i < 4; i++){ for(int j = 0; j < 4; j++){ _matPos.data[i][j] = point[i][j]; } }; Pose pose = Algo_Matrix2Pos(_matPos);

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

Pose Algo_Base2WorkFrame(Matrix matWork2Base, Pose poseBase);	
函数功能	基坐标系位姿转工作坐标系位姿。
参数	（1）matWork2Base 工作坐标系矩阵 （2）poseBase 基坐标系下的位姿

返回值	工作坐标系下的位姿
示例	Pose state; Pose pose; Pose pose1; pose1.position.x = -0.259256f; pose1.position.y = -0.170727f; pose1.position.z = 0.35621f; pose1.euler.rx = -2.85993f; pose1.euler.ry = -0.447394f; pose1.euler.rz = -1.81038f; state.position.x = 0.1f; state.position.y = 0.2f; state.position.z = 0.3f; state.euler.rx = 1; state.euler.ry = 2; state.euler.rz = 3; Matrix matrix; matrix = Algo_Pos2Matrix(state); pose = Algo_Base2WorkFrame(matrix,pose1); printf("POSE: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positio n.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

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

Pose Algo_WorkFrame2Base(Matrix matrix, Pose state);	
函数功能	工作坐标系位姿转基坐标系位姿
参数	（1）matWork2Base 工作坐标系矩阵 （2）poseBase 工作坐标系下的位姿
返回值	基坐标系下的位姿
示例	Pose state; Pose pose; Pose pose1; pose1.position.x = -0.177347f; pose1.position.y = 0.438112f; pose1.position.z = -0.215102f; pose1.euler.rx = 2.09078f; pose1.euler.ry = 0.942362f; pose1.euler.rz = 2.39144f; state.position.x = 0.1f; state.position.y = 0.2f; state.position.z = 0.3f; state.euler.rx = 1;

	state.euler.ry = 2; state.euler.rz = 3; Matrix matrix; matrix = Algo_Pos2Matrix(state); pose = Algo_WorkFrame2Base(matrix,pose1); printf("POSE: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positio n.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

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

Pose Algo_Cartesian_Tool(const float* const curr_joint, float move_lengthx,float move_lengthy, float move_lengthz);	
函数功能	计算沿工具坐标系运动位姿。
参数	（1）curr_joint 当前关节角度，单位度 （2）move_lengthx 沿 X 轴移动长度，米为单位 （3）move_lengthy 沿 Y 轴移动长度，米为单位 （4）move_lengthz 沿 Z 轴移动长度，米为单位
返回值	基坐标系下的位姿
示例	float joint[6] = {20,20,70,30,90,120};

	Pose pose; pose = Algo_Cartesian_Tool(joint,0.01f,0.01f,0.01f); printf("POSE: %f, %f, %f, %f, %f, %f\n",pose.position.x,pose.positio n.y,pose.position.z,pose.euler.rx ,pose.euler.ry ,pose.euler.rz);

3.26.22. 计算平移、旋转运动位姿 Algo_PoseMove

Pose Algo_PoseMove(Pose poseCurrent, const float *deltaPosAndRot, int frameMode);	
函数功能	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
参数	（1）poseCurrent 当前时刻位姿（欧拉角形式） （2）deltaPosAndRot 移动及旋转数组，位置移动（单位：m），旋转（单位：度） （3）frameMode 坐标系模式选择 0：Work，1：Tool（work 即可任意设置坐标系）
返回值	经平移、旋转后的位姿
示例	float q_in_init[7] = {0,-30,90,30,90,0,0}; Pose poseCurrent; poseCurrent = Algo_Forward_Kinematics(q_in_init); float deltaPosAndRot[6] = {0.01,0.01,0.01,20,20,20};

	Pose afterPose = Algo_PoseMove(poseCurrent, deltaPosAndRot, 1);

3.26.23. 设置算法关节最大限位 Set_Algo_Joint_Max_Limit

Algo_Set_Joint_Max_Limit(const float* const joint_limit);	
函数功能	设置算法关节最大限位。
参数	（1）Joint_limit 关节的最大限位数组
示例	float joint_limit[6] = {150, 100, 90, 120, 120, 300}; Algo_Set_Joint_Max_Limit(joint_limit);

3.26.24. 获取算法关节最大限位 Get_Algo_Joint_Max_Limit

Algo_Get_Joint_Max_Limit(float* joint_limit);	
函数功能	设置算法关节最大限位。
参数	（1）Joint_limit 关节的最大限位数组
示例	float after_joint_limit[6]; Algo_Get_Joint_Max_Limit(after_joint_limit);

3.26.25. 设置算法关节最小限位 Set_Algo_Joint_Min_Limit

Algo_Set_Joint_Min_Limit(const float* const joint_limit);	
函数功能	设置算法关节最大限位。
参数	（1）Joint_limit

	关节的最小限位数组
示例	float joint_limit[6] = {-150,-100,-90,-120,-120,-300}; Algo_Set_Joint_Min_Limit(joint_limit);

3.26.26. 获取算法关节最小限位

Algo_Get_Joint_Min_Limit(float* joint_limit);	
函数功能	设置算法关节最大限位。
参数	（1）Joint_limit 关节的最小限位数组
示例	float after_joint_limit[6]; Algo_Get_Joint_Min_Limit(after_joint_limit);

2、

3.26.27. 设置算法关节最大速度 Set_Algo_Joint_Max_Speed

Algo_Set_Joint_Max_Speed(const float* const joint_slim_max);	
函数功能	设置算法关节最大速度。
参数	（1）Joint_slim_max 关节的最大速度数组
示例	float joint_slimit_max[6] = {20,20,20,20,20,20}; Algo_Set_Joint_Max_Speed(joint_slimit_max);

3.26.28. 获取算法关节最大速度

Algo_Get_Joint_Max_Speed(float* joint_slim_max);	
函数功能	设置算法关节最大速度。

参数	（1）Joint_slim_max 关节的最大速度数组
示例	float after_joint_slimit_max[6]; Algo_Get_Joint_Max_Speed(after_joint_slimit_max);

3.26.29. 设置算法关节最大加速度 Set_Algo_Joint_Max_Acc

Algo_Set_Joint_Max_Acc(const float* const joint_alim_max);	
函数功能	设置算法关节最大加速度。
参数	（1）Joint_alim_max 关节的最大加速度数组
示例	float joint_alimit[6] = {20, 20, 20, 20, 20, 20}; Algo_Set_Joint_Max_Acc(joint_alimit);

3、

3.26.30. 获取算法关节最大加速度 Get_Algo_Joint_Max_Acc

Algo_Get_Joint_Max_Acc(float* joint_alim_max);	
函数功能	设置算法关节最大加速度。
参数	（1）Joint_alim_max 返回的关节的最大加速度数组
示例	float after_joint_alimit[6]; Algo_Get_Joint_Max_Acc(after_joint_alimit);

3.27. 在线编程

3.27.1. 文件下发 Send_TrajectoryFile

int Send_TrajectoryFile(SOCKHANDLE ArmSocket, Send_Project_Params params, int * err_line);	
函数功能	在线编程文件下发。
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
示例	int ret = -1; Send_Project_Params project; strcpy(project.project_path, "H:/Desktop/test.txt"); project.plan_speed = 20; project.only_save = 0; project.save_id = 3; project.project_path_len = strlen(project.project_path); int err_line; ret=Send_TrajectoryFile(m_sockhand,, project,&err_line);

3.27.2. 轨迹规划中改变速度比例系数 Set_Plan_Speed

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

函数功能	该函数用于轨迹规划中改变速度比例系数。
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
示例	`int ret;` `int speed = 20;` `ret = Set_Plan_Speed(m_sockhand, speed, RM_BLOCK);`

3.27.3. 文件树弹窗提醒 Popup

int Popup(SOCKHANDLE ArmSocket, int content, bool block);	
函数功能	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。
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
示例	int ret; int content = 1; ret = Popup(m_sockhand,content,RM_BLOCK);

3.28. 机械臂状态主动上报

3.28.1. 设置主动上报配置 Set_Realtime_Push

int Set_Realtime_Push(SOCKHANDLE ArmSocket, Realtime_Push_Config config);	
函数功能	该函数用于设置主动上报接口配置。以下参数可分开设置，均为可选字段。
参数	（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
示例	// 设置广播周期 500ms，端口号 8099，使能主动上报 Realtime_Push_Config config; config.cycle = 100; config.enable = true; config.force_coordinate = -1; strcpy(config.ip, "192.168.1.104"); config.port = 8099; config.custom.expand_state = 0; config.custom.lift_state = 0; config.custom.joint_speed = 1; ret = Set_Realtime_Push(handle, config);

	`` if (ret != SYS_NORMAL) { qDebug() << "Set_Realtime_Push err:" << ret; RM_API_UnInit(); return -1; } ```

3.28.2. 获取主动上报配置 Get_Realtime_Push

int Get_Realtime_Push(SOCKHANDLE ArmSocket, Realtime_Push_Config *config);	
函数功能	该函数用于获取主动上报接口配置。
参数	（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
示例	``` Realtime_Push_Config afterconfig; ret = Get_Realtime_Push(handle, &afterconfig); if (ret != SYS_NORMAL) { qDebug() << "_Get_Realtime_Push err:" << ret; service.Service_RM_API_UnInit(); return -1; } ```

3.28.3. 机械臂状态主动上报 Realtime_Arm_Joint_State

void Realtime_Arm_Joint_State(int port, RobotStatusListener RobotStatuscallback);	
函数功能	该函数该函数使用 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。
参数	（1）RobotStatuscallback 用于接收机械臂状态广播回调函数。
示例	Realtime_Arm_Joint_State(RobotStatuscallback);

3.29. 通用扩展关节

3.29.1. 关节速度环控制 Expand_Set_Speed

int Expand_Set_Speed(SOCKHANDLE ArmSocket, int speed, bool block);	
函数功能	扩展关节速度环控制。
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
示例	int ret = -1; int speed = 50; ret = Expand_Set_Speed(m_sockhand, speed, RM_NONBLOCK);

3.29.2. 关节位置环控制 Expand_Set_Pos

int Expand_Set_Pos(SOCKHANDLE ArmSocket, int pos, int speed, bool block);	
函数功能	该函数用于扩展关节位置环控制。
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
示例	// 以 20%的速度运行到 200mm 的位置 int ret; int target = 200; int speed = 20; ret = Expand_Set_Pos(m_sockhand, target, speed, RM_BLOCK);

3.29.3. 扩展关节状态获取 Expand_Get_State

int Expand_Get_State(SOCKHANDLE ArmSocket, int* pos, int* err_flag, int* current, int* mode);	
函数功能	该函数用于获取扩展关节状态。

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
示例	`int ret = -1;` `int pos = 0;` `int err_flag = 0;` `int current = 0;` `int mode = 0;` `ret = Expand_Get_State(m_sockhand, &pos, &err_flag, &current, &mode);`

3.30. 在线编程存储列表（I 系列）

3.30.1. 查询在线编程列表 Get_Program_Trajectory_List

```
int Get_Program_Trajectory_List(SOCKHANDLE ArmSocket, ProgramTrajectoryData*
programlist);
```

函数功能	查询在线编程程序列表。
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
示例	`// 全部查询` `ProgramTrajectoryData programData;` `programData.page_num = 0;` `programData.page_size = 0;` `programData.vague_search[0] = '\0';` `//  strcpy(programData.vague_search, "1");` `ret = Get_Program_Trajectory_List(handle, &programData);`

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

int Get_Program_Run_State(SOCKHANDLE ArmSocket, ProgramRunState* state);	
函数功能	该函数用于查询在线编程轨迹运行状态。
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

示例	ProgramRunState state; ret = Get_Program_Run_State(handle,&state);

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

int Set_Program_ID_Start(SOCKHANDLE ArmSocket, int id, int speed, bool block);	
函数功能	该函数用于开始运行指定编号轨迹。
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，按照存储的速度运行则传入 NULL （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
示例	// 以存储的速度运行第一条轨迹 int ret = -1; ret = Set_Program_ID_Start(handle, 1, NULL, RM_BLOCK);

3.30.4. 删除指定编号轨迹 Delete_Program_Trajectory

int Delete_Program_Trajectory(SOCKHANDLE ArmSocket, int id);	
函数功能	该函数用于删除指定编号轨迹。
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
示例	// 删除第二条轨迹 int ret = -1; ret = Delete_Program_Trajectory(handle, 2);

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

int Update_Program_Trajectory(SOCKHANDLE ArmSocket, int id, int plan_speed, const char *project_name);	
函数功能	该函数用于修改指定编号轨迹的信息。
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
示例	// 修改 id 为 1 的在线编程文件规划速度 20，文件名称“project_1” int ret = -1; ret = Update_Program_Trajectory(handle, 1, 20, "project_1");

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

int Set_Default_Run_Program(SOCKHANDLE ArmSocket, int id);	
函数功能	该函数用于设置 IO 默认运行的在线编程文件编号。
参数	（1）ArmSocket Socket 句柄 （2）id 指定在线编程轨迹编号
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	// 设置 IO 默认运行的在线编程文件编号为 1 int ret = -1; ret = Set_Default_Run_Program(handle, 1);

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

int Get_Default_Run_Program(SOCKHANDLE ArmSocket, int *id);	
函数功能	该函数用于获取 IO 默认运行的在线编程文件编号。
参数	（1）ArmSocket Socket 句柄 （2）id 指定在线编程轨迹编号
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	int ret = -1; int id; ret = Get_Default_Run_Program(handle, id);

3.31. 全局路点（I 系列）

3.31.1. 新增全局路点 Add_Global_Waypoint

int Add_Global_Waypoint(SOCKHANDLE ArmSocket, Waypoint waypoint);	
函数功能	该函数用于新增全局路点。
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
示例	// 新增全局路点 p3 Waypoint waypoint; strcpy(waypoint.point_name,"p3"); waypoint.joint[0] = 20; waypoint.joint[1] = 0.2; // 剩余关节角度均为 0 for (int i = 2; i < 6; ++i) { waypoint.joint[i] = 0.0; } // 设置位置姿态 waypoint.pose.position.x = 0.01; waypoint.pose.position.y = 0.02; waypoint.pose.position.z = 0.03; waypoint.pose.euler.rx = 0.1; waypoint.pose.euler.ry = 0.2; waypoint.pose.euler.rz = 0.3; strcpy(waypoint.work_frame, "World"); strcpy(waypoint.tool_frame, "Arm_Tip"); ret = Add_Global_Waypoint(handle, waypoint);

3.31.2. 更新全局路点 Update_Global_Waypoint

int Update_Global_Waypoint(SOCKHANDLE ArmSocket, Waypoint waypoint);

函数功能	该函数用于更新全局路点
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
示例	// 更新全局路点 p3 Waypoint waypoint; strcpy(waypoint.point_name,"p3"); waypoint.joint[0] = 0.0; waypoint.joint[1] = 0.2; // 剩余关节角度均为 0 for (int i = 2; i < 6; ++i) { waypoint.joint[i] = 0.0; } // 设置位置姿态 waypoint.pose.position.x = 0.01; waypoint.pose.position.y = 0.02; waypoint.pose.position.z = 0.03; waypoint.pose.euler.rx = 0.1; waypoint.pose.euler.ry = 0.2; waypoint.pose.euler.rz = 0.3; strcpy(waypoint.work_frame, "World"); strcpy(waypoint.tool_frame, "Arm_Tip"); ret = Update_Global_Waypoint(handle, waypoint);

3.31.3. 删除全局路点 Delete_Global_Waypoint

int Delete_Global_Waypoint(SOCKHANDLE ArmSocket, const char* name);	
函数功能	该函数用于删除全局路点。
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
示例	// 删除全局路点 p3 Delete_Global_Waypoint(handle, "p3");

3.31.4. 查询多个全局路点 Get_Global_Point_List

int Get_Global_Point_List(SOCKHANDLE ArmSocket, WaypointsList* point_list);	
函数功能	该函数用于查询多个全局路点。
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
示例	// 获取多个全局路点列表 WaypointsList list ; ret = Get_Global_Point_List(handle, &list);

3.31.5. 查询指定全局路点 Given_Global_Waypoint

int Given_Global_Waypoint(SOCKHANDLE ArmSocket, const char *name, Waypoint* point);	
函数功能	该函数用于查询指定全局路点。
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
示例	// 获取全局路点 p1 参数 Waypoint point; ret = Get_Global_Point_List(handle, "p1", &point);

3.32. 电子围栏与虚拟墙（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。

3.32.1. 新增几何模型参数 Add_Electronic_Fence_Config

int Add_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);	
函数功能	该函数用于新增几何模型参数，最多支持 10 个几何模型。
参数	（1）ArmSocket Socket 句柄 （2）config 几何模型参数
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	// 新增几何模型“tadd” ElectronicFenceConfig config; config.form = 1; config.x_max_limit = 0.500; config.x_min_limit = -0.500; config.y_max_limit = 0.500; config.y_min_limit = -0.500; config.z_max_limit = 0.500; config.z_min_limit = -0.500; strcpy(config.name,"tadd"); ret = Add_Electronic_Fence_Config(handle, config);

3.32.2. 更新几何模型参数 Update_Electronic_Fence_Config

int Update_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);

函数功能	该函数用于更新几何模型参数，最多支持 10 个几何模型。
参数	（1）ArmSocket Socket 句柄 （2）config 几何模型参数
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	// 更新几何模型”tadd” ElectronicFenceConfig config; config.form = 1; config.x_max_limit = 0.500; config.x_min_limit = -0.500; config.y_max_limit = 0.500; config.y_min_limit = 0; config.z_max_limit = 0.100; config.z_min_limit = -0.500; strcpy(config.name,"tadd"); ret = Update_Electronic_Fence_Config(handle, config);

3.32.3. 删除几何模型参数 Delete_Electronic_Fence_Config

int Delete_Electronic_Fence_Config(SOCKHANDLE ArmSocket, const char* name);	
函数功能	该函数用于删除几何模型参数，最多支持 10 个几何模型。
参数	（1）ArmSocket Socket 句柄 （2）name 指定几何模型名称
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	// 删除几何模型 ret = Delete_Electronic_Fence_Config(handle, (char*)"tadd");

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

int Get_Electronic_Fence_List_Names(SOCKHANDLE ArmSocket, ElectronicFenceNames* names, int *len);	
函数功能	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。
参数	（1）ArmSocket Socket 句柄 （2）names 几何模型名称列表，长度为实际存在几何模型 （3）len 几何模型名称列表长度
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	`// 所有几何模型名称` `ElectronicFenceNames name[10];` `int len;` `ret = Get_Electronic_Fence_List_Names(handle, name, &len);`

3.32.5. 查询指定几何模型参数 Given_Electronic_Fence_Config

int Given_Electronic_Fence_Config(SOCKHANDLE ArmSocket, const char *name, ElectronicFenceConfig* config);	
函数功能	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。

参数	（1）ArmSocket Socket 句柄 （2）name 指定几何模型名称 （3）config 返回几何模型参数
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询
示例	// 查询指定几何模型“tadd” ElectronicFenceConfig config; ret = Given_Electronic_Fence_Config(handle, (char*)"tadd", &config);

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

int Get_Electronic_Fence_List_Info(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config, int *len);	
函数功能	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。
参数	（1）ArmSocket Socket 句柄 （2）config 几何模型信息列表，长度为实际存在几何模型 （3）len 几何模型信息列表长度
返回值	成功返回：0。失败返回：错误码，rm_define.h 查询

示例	// 所有几何模型信息 ElectronicFenceConfig list[10]; int len; ret = Get_Electronic_Fence_List_Info(handle, list, &len);

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

int Set_Electronic_Fence_Enable(SOCKHANDLE ArmSocket, bool enable_state, int in_out_side, int effective_region);	
函数功能	该函数用于设置电子围栏使能状态。
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
示例	// 设置电子围栏使能，在电子围栏内部，针对整臂区域生效 ret = Set_Electronic_Fence_Enable(handle, true, 0, 0);

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

int Get_Electronic_Fence_Enable(SOCKHANDLE ArmSocket, bool* enable_state, int* in_out_side, int* effective_region);	
函数功能	该函数用于获取电子围栏使能状态。

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
示例	`// 获取电子围栏状态` `bool state;` `int in_out;` `int region;` `ret = Get_Electronic_Fence_Enable(handle, &state, &in_out, &region);`

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

int Set_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);	
函数功能	该函数用于设置当前电子围栏参数。
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

示例	// 设置当前电子围栏参数为长方体 ElectronicFenceConfig config; config.form = 1; config.x_max_limit = 0.10; config.x_min_limit = 0.05; config.y_max_limit = 0.10; config.y_min_limit = 0.05; config.z_max_limit = 0.10; config.z_min_limit = 0.05; ret = Set_Electronic_Fence_Config(handle, config);

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

int Get_Electronic_Fence_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config);	
函数功能	该函数用于获取当前电子围栏参数。
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
示例	// 获取当前围栏参数 ElectronicFenceConfig aaconfig; ret = Get_Electronic_Fence_Config(handle, &aaconfig);

3.32.11. 设置虚拟墙使能状态 Set_Virtual_Wall_Enable

int Set_Virtual_Wall_Enable(SOCKHANDLE ArmSocket, bool enable_state, int in_out_side, int effective_region);	
函数功能	该函数用于设置虚拟墙使能状态。

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
示例	// 设置虚拟墙使能，在虚拟墙内部，针对整臂区域生效 ret = Set_Virtual_Wall_Enable(handle, true, 0, 0);

3.32.12. 获取虚拟墙使能状态 Get_Virtual_Wall_Enable

int Get_Virtual_Wall_Enable(SOCKHANDLE ArmSocket, bool* enable_state, int* in_out_side, int* effective_region);	
函数功能	该函数用于获取虚拟墙使能状态。
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

	0-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
示例	// 获取虚拟墙状态 bool state; int in_out; int region; ret = Get_Virtual_Wall_Enable(handle, &state, &in_out, &region);

3.32.13. 设置当前虚拟墙参数 Set_Virtual_Wall_Config

int Set_Virtual_Wall_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig config);	
函数功能	该函数用于设置当前虚拟墙参数。
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
示例	// 设置当前虚拟墙参数为长方体 ElectronicFenceConfig config; config.form = 1; config.x_max_limit = 0.10; config.x_min_limit = 0.05; config.y_max_limit = 0.10; config.y_min_limit = 0.05; config.z_max_limit = 0.10; config.z_min_limit = 0.05; ret = Set_Virtual_Wall_Config(handle, config);

3.32.14. 获取当前虚拟墙参数 Get_Virtual_Wall_Config

int Get_Virtual_Wall_Config(SOCKHANDLE ArmSocket, ElectronicFenceConfig* config);	
函数功能	该函数用于获取当前虚拟墙参数。
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
示例	`// 获取当前围栏参数` `ElectronicFenceConfig aaconfig;` `ret = Get_Virtual_Wall_Config(handle, &aaconfig);`

3.33. 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。

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

int Set_Self_Collision_Enable(SOCKHANDLE ArmSocket, bool enable_state);	
函数功能	该函数用于设置自碰撞安全检测使能状态。

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
示例	// 禁使能自碰撞安全检查 ret = Set_Self_Collision_Enable(handle, false);

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

int Get_Self_Collision_Enable(SOCKHANDLE ArmSocket, bool* enable_state);	
函数功能	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。
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
示例	// 查询自碰撞安全检测使能状态 bool state; ret = Get_Self_Collision_Enable(handle, &state);