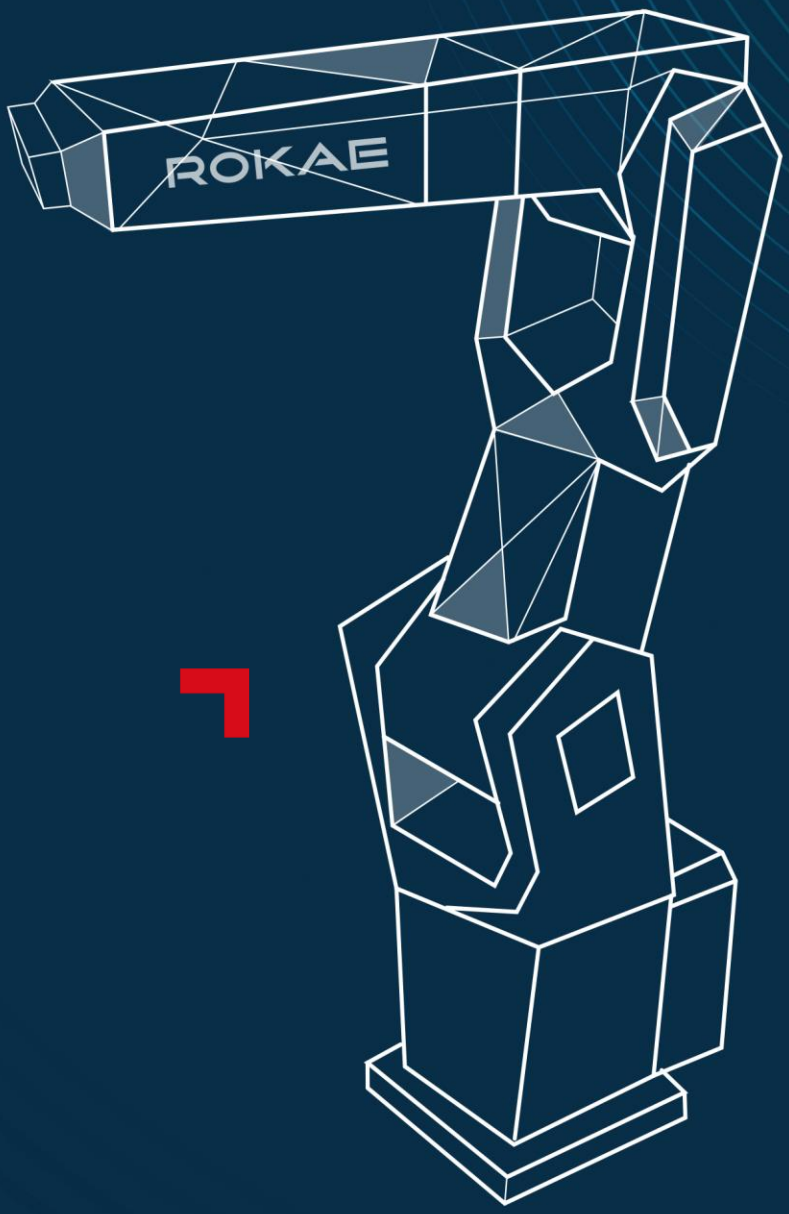
ROKAE 珞石

轻 型 机 器 人 专 家



xCore RCI

使用手册

让智造更高效

xCore RCI 使用手册

产品手册

控制系统版本：V2.0

文档版本：A

本手册中记载的内容如有变更，恕不事先通告。本公司对手册中可能出现的错误均不承担任何责任。
本公司对因使用本手册及其中所述产品而引起的意外或间接伤害均不承担任何责任，敬请谅解。
本公司不可能预见所有的危险和后果，因此本手册不能警告用户所有可能的危险。

如您发现本手册的内容有误或需要改进抑或补充之处，请不吝指正。
本手册的原始语言为中文，所有其他语言版本均翻译自中文版本。

珞石（北京）智能科技有限公司
中国.北京

目录

1 手册概述

1.1 关于本手册

感谢您购买本公司的机器人系统。

本手册记载了正确安装使用机器人的以下说明：

- 机器人二次开发接口 RCI 的使用。

安装使用该机器人系统前，请仔细阅读本手册与其他相关手册。

阅读之后，请妥善保管，以便随时取阅。

1.2 手册对象

本手册面向：

- 机器人应用开发工程师

请务必保证以上人员具备电气安装、编程和操作所需的知识，并已接受本公司的相关培训。

1.3 如何阅读产品手册

本手册包含单独的安全章节，必须在阅读安全章节后，才能进行安装或维护作业。

1.4 版本记录

版本编号	日期	说明
V1.5	2022.6	
V1.6	2022.09	
V1.7	2023.02	
V2.0	2023.06	

2 概述

2.1 简介

RCI 是 ROKAE Control Interface 的首字母缩写，使用 C++语言编写而成，包含了一系列底层控制接口，科研或二次开发用户可以使用该软件包实现最高达 1KHz 的实时控制，用于算法验证以及新应用的开发。

下文适用版本：RCI V1.4.1

2.2 使用环境配置

2.2.1 PC 操作系统

RCI 目前运行在 Linux 和 Windows 平台，为保障实时性，推荐使用 Linux 搭配实时内核使用。
Linux with PREEMPT_RT patched kernel：推荐使用 Ubuntu16.04 with 4.14.12-rt10 kernel。

2.2.2 Linux 实时环境配置

RCI 需运行在实时环境中，实时环境配置有以下步骤：
(1) 安装依赖

- apt-get install build-essential bc curl ca-certificates fakeroot gnupg2 libssl-dev lsb-release libelf-dev bison flex cmake libeigen3-dev

(2)下载实时内核补丁
通过 uname -r 命令可以知道本机正在使用的内核；
通过网站 https://www.kernel.org/pub/linux/kernel/projects/rt/查找离现在内核版本最接近的 kernal；
下载文件：

- curl -SLO https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.14.12.tar.xz
- curl -SLO https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.14.12.tar.sign
- curl -SLO https://www.kernel.org/pub/linux/kernel/projects/rt/4.14/older/patch-4.14.12-rt10.patch.xz
- curl –SLO https://www.kernel.org/pub/linux/kernel/projects/rt/4.14/older/patch-4.14.12-rt10.patch.sign

如果国内网速很慢可以直接手动从网站上下载或者找其他镜像源；
解压：

- xz -d linux-4.14.12.tar.xz
- xz -d patch-4.14.12-rt10.patch.xz

检查 sign 文件完整性

- gpg2 --verify linux-4.14.12.tar.sign

会得到类似于如下的信息：

$ gpg2 --verify linux-4.14.12.tar.sign gpg: assuming signed data in 'linux-4.14.12.tar
gpg: Signature made Fr 05 Jan 2018 06:49:11 PST using RSA key ID 6092693E
gpg: Can't check signature: No public key

记下 ID 6092693E 执行：

- gpg2 --keyserver hkp://keys.gnupg.net --recv-keys 0x6092693E

同理对于 patch 文件，执行相同的操作。

下载完成 server key 后再次验证，若得到如下信息就说明是正确的。
$ gpg2 --verify linux-4.14.12.tar.sign
gpg: assuming signed data in 'linux-4.14.12.tar'
gpg: Signature made Fr 05 Jan 2018 06:49:11 PST using RSA key ID 6092693E
gpg: Good signature from "Greg Kroah-Hartman <gregkh@linuxfoundation.org>"[unknown]
gpg: aka "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
gpg: aka "Greg Kroah-Hartman (Linux kernel stable release signing key) <greg@kroah.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner. Primary key fingerprint: 647F 2865 4894 E3BD 4571 99BE 38DB BDC8 6092 693E

同理验证一下 patch 文件。

(3)编译内核
解压：

- tar xf linux-4.14.12.tar
- cd linux-4.14.12
- patch -p1 < ../patch-4.14.12-rt10.patch

配置内核：

- make oldconfig

出现以下信息：
Preemption Model
1. No Forced Preemption (Server) (PREEMPT_NONE)
2. Voluntary Kernel Preemption (Desktop) (PREEMPT_VOLUNTARY)
3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT__LL) (NEW)
4. Preemptible Kernel (Basic RT) (PREEMPT_RTB) (NEW)
> 5. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)

选 5 然后一直 enter。
开始编译：

- fakeroot make -j4 deb-pkg

dpkg 安装：

- sudo dpkg -i ../linux-headers-4.14.12-rt10_*.deb ../linux-image-4.14.12-rt10_*.deb

(3)验证是否安装成功

重启一下，进 ubuntu 高级选项，可以看到你安装的内核。选择新安装的内核进入后，通过 uname -r 查看对应内核版本，如果版本正确，/sys/kernel/realtime 里内容是 1。

2.2.3 Linux 环境下 RCI 编译过程

- 解压 SDK 包；
- 执行以下指令：

```
cd rci_client/build/
rm -rf *
cmake ..
make
```

生成的可执行文件在 build/examples 目录下。执行程序时需要添加 root 权限。

2.2.4 Windows 环境下 RCI 编译过程

- 解压 SDK 包；
- 打开 example 目录下的 vs 工程，编译运行

2.2.5 网络配置

使用 RCI 之前需要对网络进行配置，将用户 PC 与机器人连接起来。

机器人配置有 2 个网口，一个是外网口，一个是直连网口。直连网口默认静态 IP 地址是 192.168.0.160。连接机器人有两种方式。

- 连接方式 1：机器人与用户 PC 采用网线直连的方式连接。如果用户工控机与机器人不处于同一个网段，需要配置用户 PC 的 IP 使其与机器人静态 IP 地址处于同一个网段，例如 192.168.0.22。
- 连接方式 2：机器人外网口连接路由器或者交换机，用户 PC 也连接路由器或者交换机，两者处于同一局域网。

用户打开 HMI，在 RCI 界面将 IP 设置成用户 PC 的 IP，端口号设置成 1337，按下使能开关即可连接机器人。

注：推荐使用方式 1 进行连接，连接方式 2 网络通信质量差时可能会造成机器人运动不稳定现象。

2.2.6 RCI 自启动

开启 RCI 自启动功能后，可以脱离 HMI 对机器人进行控制。开启 RCI 自启动只需要将 HMI 上的 RCI 使能开关打开即可，机器人重启后会保持 RCI 打开状态，并自动切换成自动模式。

2.3 软件接口功能

通常将使用 RCI 的用户工控机视为客户端，机器人控制器作为服务端。

RCI 提供了使用简单的网络通信与控制功能接口：

- 处理非实时命令
- 处理实时命令

- 获取机器人状态和回调机制
- 提供计算机器人运动学与动力学的 Model 库
- S 轨迹规划接口

2.3.1 非实时命令

非实时命令也叫 TCP 命令，主要功能是：设置运动开始前的运动参数。TCP 命令有：

2.3.1.1 startMove

void startMove(ControllerMode，MotionGeneratorMode)	
函数说明	运动开始前指定控制器运动和控制模式，调用此接口机器人不会立即开始运动，在每段回调（运动）执行前需要先调用此接口设置。若设置了与回调函数定义不一致的类型，会出现机器人不响应回调指令的现象。回调函数用法见 获取机器人状态和回调机制。
参数说明	● ControllerMode，控制器模式：轴空间位置控制/笛卡尔空间位置控制/轴空间阻抗控制/笛卡尔空间阻抗控制/直接力矩控制 ● MotionGeneratorMode，运动模式：轴空间运动/笛卡尔空间运动

2.3.1.2 stopMove

void stopMove()	
函数说明	停止运动，停止收发 UDP 数据。在运动过程中可以调用此接口停止运动或通过回调函数的返回值停止。
参数说明	

2.3.1.3 setCollisionBehavior

void setCollisionBehavior(upper_torque_thresholds_nominal)	
函数说明	设置碰撞检测阈值，碰撞检测只在位置控制时生效，力控时不生效。若检测到碰撞，控制器会下发下电指令，电机抱闸吸合下使能。
参数说明	● upper_torque_thresholds_nominal,碰撞检测阈值，各轴最大值：（75，75，60，45，30，30，20），若为六轴机型，那么各轴最大值是：（75，75，45，30，30，20，0）；

2.3.1.4 setCartesianLimit

void setCartesianLimit(object_world_size，object_frame，object_activation)	
函数说明	设置安全区域，非力控虚拟墙。若机器人末端或 TCP 末端超过安全区域，电机同样会做下使能处理。
参数说明	● object_world_size,安全区域长宽高，对应 XYZ，单位：m ● object_frame,安全区域中心位姿 ● object_activation,bool 变量，是否开启安全区域

2.3.1.5 setJointImpedance

void setJointImpedance(K_theta)	
函数说明	设置轴空间阻抗系数，轴空间阻抗运动时生效。

参数说明	● K_theta,轴空间阻抗系数：max={{3000，3000，3000，3000，300，300，300}}，若为六轴机型，那么各轴最大值是：(3000，3000，3000，300，300，300，0)，数值对应单位是 Nm/rad

2.3.1.6 setCartsianImpedance

void setCartesianImpedance(K_x)	
函数说明	设置笛卡尔空间阻抗系数，笛卡尔阻抗运动时生效。
参数说明	● K_x,笛卡尔空间阻抗系数：max={{3000，3000，3000，300，300，300}}，数值对应单位是 N/m 或者 Nm/rad

2.3.1.7 setCoor

void setCoor(F_T_EE)	
函数说明	设置工具 TCP，设置 TCP 后控制器会保存配置，机器人重启后恢复默认设置。
参数说明	● F_T_EE,工具 TCP 相对于机器人末端法兰的位姿，用齐次矩阵表示，单位是 rad 或者 m。

2.3.1.8 setLoad

void setLoad(load_mass， load_center，load_inertia)	
函数说明	设置负载，设置负载后控制器会保存负载配置，机器人重启后恢复默认设置。
参数说明	● load_mass,负载质量，单位：kg ● load_center,负载质心，单位：m ● load_inertia,负载惯量，kg·m²

2.3.1.9 setFilters

void setFilters(joint_position_filter_frequency, cartesian_position_filter_frequency, torque_filter_frequency)	
函数说明	设置滤波截至频率，用来平滑指令。建议设置参数范围：10~100Hz
参数说明	● joint_position_filter_frequency，轴空间位置滤波截至频率，单位：Hz ● cartesian_position_filter_frequency，笛卡尔空间位姿滤波截至频率，单位：Hz ● torque_filter_frequency，关节力矩滤波截至频率，单位：Hz

2.3.1.10 setCartImpDesiredTau

void setCartImpDesiredTau(tau)	
函数说明	设置笛卡尔空间阻抗期望力，在笛卡尔空间阻抗运动时生效。
参数说明	● Tau，笛卡尔空间阻抗期望力，例如 tau={{0，0，10，0，0，0}}，单位：N，N·m。

2.3.1.11 setMotorPower

void setMotorPower(int motor_state)	
函数说明	设置电机抱闸的释放与闭合。
参数说明	● motor_state，机器人上电状态，1 代表使机器人上电，0 代表使机器人下电。

2.3.1.12 getMotorState

int getMotorState ( )	
函数说明	获取机器人上电状态，返回值：0 代表机器人处于下使能状态，1 代表电机处于上使能状态。
参数说明	

2.3.1.13 setDO

void setDO(DOSIGNAL DO_signal, bool state)	
函数说明	设置 IO OUTPUT 信号
参数说明	● DO_signal,DO 端口号，分别对应{DO0_0, DO0_1, DO0_2, DO0_3, DO1_0, DO1_1}。 ● State,1 代表高电平输出，0 代表低电平输出。

2.3.1.14 getDI

bool getDI(DISIGNAL DI_signal)	
函数说明	获取 IO INPUT 信号，返回值，true 代表高电平，false 代表低电平。
参数说明	● DI_signalS,DI 端口号，分别对应{DI0_0, DI0_1, DI0_2, DI0_3, DI1_0, DI1_1}。

2.3.1.15 setJointLimit

void setJointLimit(upper_joint_limit, lower_joint_limit, auto_limit)	
函数说明	设置机器人软限位
参数说明	● upper_joint_limit, XMATE3_PRO 和 XMATE3_PRO 机型软限位最大值 max = {{170,120,170,120,170,120,360}}。其他机型的软限位范围请参见 xCore 机器人控制系统使用手册。 ● lower_joint_limit, XMATE3_PRO 和 XMATE3_PRO 机型软限位最小值 min = {{-170,-120,-170,-120,-170,-120,-360}}。其他机型的软限位范围请参见 xCore 机器人控制系统使用手册。 ● auto_limit,置为 true 时，轴空间运动时，机器人在即将到达限位附近时会对关节角度进行限幅处理，机器人不至于下电。

2.3.1.16 setFcCoor

void setFcCoor(fc_coor, fc_type)	
函数说明	设置力控坐标系。
参数说明	● fc_coor,力控坐标系相对于法兰坐标系的变换矩阵。

	● fc_type ,力控坐标系类型：1 世界坐标系；2 工具坐标系；3 路径坐标系；

2.3.1.17 startDrag

void startDrag(drag_space, drag_type)	
函数说明	开启拖动，此接口非阻塞接口，若开启拖动后程序立即结束，会立即自动关闭拖动，拖动状态下不能执行运动指令。
参数说明	● drag_space，拖动空间：1 轴空间拖动； 2 笛卡尔空间拖动。 ● drag_type，拖动类型：1 仅平移；2 仅旋转；3 自由拖动。

2.3.1.18 stopDrag

void stopDrag()	
函数说明	关闭拖动。
参数说明	● DI_signalS，DI 端口号，分别对应{DI0_0, DI0_1, DI0_2, DI0_3, DI1_0, DI1_1}。

2.3.1.19 rebootRobot

void rebootRobot ()	
函数说明	● 重启机器人控制器。重启后客户端会断开，机器人控制器重启之后自动切换到 RCI 状态。
参数说明	

2.3.1.20 getSafetyStopState

int getSafetyStopState()	
函数说明	● 获取机器人急停状态，返回值为急停类型，1 无急停； 2 ESTOP； 3 GSTOP。
参数说明	

2.3.1.21 MOVEJ

void MOVEJ(speed_ratio, q_start, q_target, robot)	
函数说明	PTP 轴空间规划运动，S 轨迹规划方法，此接口是阻塞接口。
参数说明	● speed_ratio，速度缩比系数； ● q_start，初始关节角度； ● q_target，目标关节角度； ● robot，实例化机器人对象；

2.3.1.22 MOVEL

void MOVEL(speed_ratio, pose_start, pose_target, robot)	
函数说明	PTP 笛卡尔空间直线规划运动，S 轨迹规划方法，此接口是阻塞接口。

参数说明	speed_ratio，速度缩比系数； pose_start，初始位姿；如果设置了 TCP，那么应该为 TCP 相对基坐标系的位姿； pose_target，目标位姿；如果设置了 TCP，那么应该为 TCP 相对基坐标系的位姿； Robot，实例化机器人对象；

2.3.1.23 MOVEC

void MOVEC(speed，pose_start， pose_mid，pose_target， robot)	
函数说明	3 点圆弧规划运动，S 轨迹规划方法。
参数说明	● speed_ratio，速度缩比系数； ● pose_start，初始位姿；如果设置了 TCP，那么应该为 TCP 相对基坐标系的位姿； ● pose_mid ，中间位姿；如果设置了 TCP，那么应该为 TCP 相对基坐标系的位姿； ● pose_target，目标位姿；如果设置了 TCP，那么应该为 TCP 相对基坐标系的位姿； ● robot，实例化机器人对象；

2.3.2 实时数据

实时数据通过 UDP 协议来发送，主要有 RCI 发送给机器人的数据 RobotCommand 和机器人给 RCI 发送的数据 RobotState。

2.3.2.1 RobotCommand

说明	RCI 发送给机器人数据结构
定义	

```
struct MotionCommand {
    std::array<double, 7> q_c ; //关节角度
    std::array<double, 16> toolTobase_pos_c ; //末端位姿，行优先排列
    bool psi_valid ; //是否有臂角
    double psi_c ; //臂角
    bool motion_generation_finished ; //运动是否结束
};

struct TorqueCommand {
    std::array<double, 7> tau_c ; //关节力矩
};

struct RobotCommand{
    uint64_t message_id ;
    MotionCommand motion ;
    TorqueCommand torque ;
} ;
```

2.3.2.2 RobotState

说明	机器人发送给 RCI 的数据结构
定义	

```
struct RobotState {
    uint64_t message_id;
    std::array<double, 7> q{};//关节角度
    std::array<double, 7> q_c{};//指令关节角度
    std::array<double, 7> dq_m{};//关节速度
    std::array<double, 7> dq_c{};//指令关节速度
    std::array<double, 7> ddq_c{};//指令关节加速度
    std::array<double, 16> toolTobase_pos_m{};//机器人位姿
    std::array<double, 16> toolTobase_pos_c{};//指令机器人位姿
    std::array<double, 6> toolTobase_vel_c{};//指令机器人末端速度
    std::array<double, 6> toolTobase_acc_c{};//指令机器人末端加速度
    double psi_m;//臂角
    double psi_c;//指令臂角
    double psi_vel_c;//指令臂角速度
    double psi_acc_c;//指令臂角加速度
    std::array<double, 7> tau_m;//关节力矩
    std::array<double,7> tau_m_filtered;
    std::array<double, 7> tau_c;//指令关节力矩
    std::array<double, 7> tau_vel_c;//指令力矩微分
    std::array<double, 6> tau_ext_in_base;//基坐标系中外部力矩
    std::array<double, 6> tau_ext_in_stiff;//力控坐标系中外部力矩
    std::array<double, 7> theta_m;//电机位置
    std::array<double, 7> theta_vel_m;//电机位置微分
    std::array<double, 7> motor_tau;//电机转矩
    std::array<double,7> motor_tau_filtered;
    std::array<bool, 20> errors;//错误位
    double control_command_success_rate;//指令接收成功率
    MotionGeneratorMode motion_generator_mode;//运动发生模式
    ControllerMode controller_mode;//控制模式
    RobotMode robot_mode;//机器人状态
};
```

2.3.3 获取机器人状态和回调机制

RCI 通过回调函数的形式实现一个控制周期内机器人状态 RobotState 的获取与运动指令 RobotCommand 的下发,控制周期是 1ms。

调用 Control(callbcak)函数来注册和开启回调，control()函数是阻塞接口，客户端每隔 1ms 接收一次 RobotStates 数据，同时下发此时的 RobotCommand 命令。

示例程序：

```cpp
int main(int argc, char *argv[]) {
    std::string ipaddr = "192.168.0.160";
    uint16_t port = 1337;

    // RCI 连接机器人
    xmate::Robot robot(ipaddr, port);

    //防止网络还未连接就进行数据交互
    sleep(2);

    //电机上使能，RCI 使用前提是机器人处于自动模式，并且电机是上使能状态
    int power_state=1;

    robot.setMotorPower(power_state);
    std::array<double,7> q_init;
    std::array<double,7> q_drag = {{0,PI/6,0,PI/3,0,PI/2,0}};

    //接收一次 udp 数据，即接收一次机器人状态
    q_init = robot.receiveRobotState().q;
    //调用 MOVEJ 移动到目标位姿，此接口阻塞，规划运动完成后自动退出。
    MOVEJ(0.2,q_init,q_drag,robot);

    //用户规划回调程序
    std::array<double, 7> q_end = {{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}};

    //创建一个 S 规划器，0.2 是速度缩放系数，数值越大代表速度越快。
    JointMotionGenerator joint_s(0.2, q_end);

    //开始回调（运动）前先设置控制模式和运动模式
    robot.startMove(RCI::robot::StartMoveRequest::ControllerMode::kJointPosition,
                    RCI::robot::StartMoveRequest::MotionGeneratorMode::kJointPosition);
    std::array<double, 7> init_position;
    std::array<double, 7> joint_delta;
    JointPositions output{};
    static bool init = true;
    double time = 0;

    //用户回调函数，规划或者控制算法写在这里
    JointControl joint_position_callback;
    joint_position_callback = [&](RCI::robot::RobotState robot_state) -> JointPositions {
        time += 0.001;
```

```
        if(init==true){
        //保存机器人初始角度
            init_position = robot_state.q;
            init=false;
        }
        //同步各轴角度，使得各轴同时到达目标角度
        joint_s.calculateSynchronizedValues_joint(init_position);
        // calculateDesiredValues_joint 返回 true 代表规划结束，根据 time 值计算出相对于初始关节角度的角度增量 joint_delta
        if (joint_s.calculateDesiredValues_joint(time, joint_delta) == false) {
            output.q = {{init_position[0] + joint_delta[0],
                         init_position[1] + joint_delta[1],
                         init_position[2] + joint_delta[2],
                         init_position[3] + joint_delta[3],
                         init_position[4] + joint_delta[4],
                         init_position[5] + joint_delta[5],
                         init_position[6] + joint_delta[6]}};
        } else {
            std::cout<<"运动结束："<<std::endl;
            return MotionFinished(output);
        }
        // 返回关节角度指令
        return output;
    };

    //调用 control 函数开启回调，运动结束前处于阻塞状态，每次回调 RCI 以 UDP 的形式发送指令到机器人控制器。
    robot.Control(joint_position_callback);
    return 0;
}
```

2.3.4 运动学和动力学计算

RCI 为用户提供了 xMate 运动学与动力学计算的库，方便计算机器人正逆解和雅克比矩阵等。主要接口功能包括：

- 运动学正解：pose = model.GetCartPose(q)，由关节角度 q 正解得到笛卡尔空间位姿 pose。
- 运动学逆解：int res = model.GetJointPos(Cartpose，psi，q_init，q_out)，由目标笛卡尔空间位姿 pose，目标臂角 psi 和机器人初始关节角度 q_init 逆解得到目标关节角度 q_out，一个位姿可能对应多个关节角度，q_out 的选取原则是选取一个与 q_init 最近的解。Res 返回值：
  - -1，-2，-3 代表无解，原因是 Cartpose 超出机器人工作空间;
  - -4，-5 代表 q_out 与 q_init 相差较大，一般认为 q_init 代表机器人当前位置，

q_out 与 q_init 只差可以等效为电机转速。若超过机器人轴额定转速，则返回-4或-5;

- -6，-7 代表 q_out 超过软限位;
- -8 代表机器人奇异;

- 雅可比矩阵计算：jac = model. Jacobian(q)，由关节角度 q 计算得到雅可比矩阵 jac。
- 动力学正解：model. GetTauNoFriction(q,dq,ddq,trq_full,trq_inertial,trq_coriolis,trq_gravity)，由关节角度 q，关节角速度 dq，关节角加速度 ddq 正解得到总关节力矩 trq_full，惯性力 trq_inertial，trq_coriolis 科氏力，trq_gravity 重力矩。

具体接口和使用方法可以参考 RCI_SDK 中 model.h 文件和 torques_control.cpp 文件。

2.3.5 S 轨迹规划接口

S 规划指规划轨迹速度是 S 曲线，能保证速度加速度的连续可导，使得运动高效平稳。用户可根据需求进行轴空间或者笛卡尔空间的 S 规划，是一种离线的规划方法，应该明确的是此方法不涉及到路径规划，路径应由用户定义与生成。

以轴空间的 s 规划为例：

- JointMotionGenerator joint_s(0.2, q_end)，创建一个目标关节角度为 q_end 的规划器，0.2 为速度系数。
- joint_s.calculateSynchronizedValues_joint(q_start)，指定初始关节角度 q_start，并做同步处理。
- joint_s.calculateDesiredValues_joint(t, q_delta)，计算在时间 t 时的关节角度相对于 q_start 的增量 q_delta。

具体使用方法参见 RCI_SDK 中 joint_motion.h， cart_motion.h， joint_s.cpp 文件。

2.4 注意事项

用户下发的 RobotCommand 需要满足建议条件和必要条件。建议条件是为了让机器人运动更加平稳，性能最优。必要条件则是必须满足的，若不满足，机器人将会停止。

2.4.1 轴空间运动

1) 必要条件

轴空间轨迹平滑，至少保证速度是连续可导的：

$$
\begin{aligned}
q_{min} < q_c < q_{max} \\
-\dot{q}_{max} < \dot{q}_c < \dot{q}_{max} \\
-\ddot{q}_{max} < \ddot{q}_c < \ddot{q}_{max} \\
-\dddot{q}_{max} < \dddot{q}_c < \dddot{q}_{max}
\end{aligned}
$$

2) 建议条件

力矩指令不超限:

$$
\begin{aligned}
-\tau_{jmax} < \tau_{jc} < \tau_{jmax} \\
-\dot{\tau}_{jmax} < \dot{\tau}_{jc} < \dot{\tau}_{jmax}
\end{aligned}
$$

运动开始时：

$$
\begin{aligned}
q = q_c \\
\dot{q}_c = 0 \\
\ddot{q}_c = 0
\end{aligned}
$$

运动结束时：

$$\dot{q}_c = 0$$
$$\ddot{q}_c = 0$$

2.4.2 笛卡尔空间运动

1) 必要条件

不处于奇异位且在工作空间内:

$$-\dot{p}_{max} < \dot{p}_c < \dot{p}_{max}$$
$$-\ddot{p}_{max} < \ddot{p}_c < \ddot{p}_{max}$$
$$-\dddot{p}_{max} < \dddot{p}_c < \dddot{p}_{max}$$

2) 建议条件

力矩指令不超限:

$$-\tau_{jmax} < \tau_{jc} < \tau_{jmax}$$
$$-\dot{\tau}_{jmax} < \dot{\tau}_{jc} < \dot{\tau}_{jmax}$$

运动开始时:

$$T = T_c$$
$$\dot{p}_c = 0$$
$$\ddot{p}_c = 0$$

运动结束时:

$$\dot{p}_c = 0$$
$$\ddot{p}_c = 0$$

2.4.3 力矩直接控制

1) 必要条件

力矩指令不超限且连续:

$$-\dot{\tau}_{jmax} < \dot{\tau}_{jc} < \dot{\tau}_{jmax}$$
$$-\tau_{jmax} < \tau_{jc} < \tau_{jmax}$$

2.4.4 运动限制条件

xMatePro3、xMatePro7、xMate3、xMate7 笛卡尔空间运动限制条件:

Name	Translation	Rotation
$\dot{p}_c$	$1.0\ m/s$	$2.5\ rad/s$
$\ddot{p}_c$	$10.0\ m/s^2$	$10.0\ rad/s^2$
$\dddot{p}_c$	$5000.0\ m/s^3$	$5000.0\ rad/s^3$

xMatePro3、xMatePro7 轴空间运动限制条件:

Name	Joint1	Joint2	Joint3	Joint4	Joint5	Joint6	Joint7	Unit
q_{max}	170	120	170	120	170	120	360	°
q_{min}	-170	-120	-170	-120	-170	-120	-360	°
$\dot{q}_{max}$	2.175	2.175	2.175	2.175	2.610	2.610	2.610	rad/s

$\ddot{q}_{\max}$	15	7.5	10	10	15	15	20	rad/s^2
$\dddot{q}_{\max}$	5000	3500	5000	5000	7500	7500	7500	rad/s^3

xMate3、xMate7 轴空间运动限制条件:

Name	Joint1	Joint2	Joint4	Joint5	Joint6	Joint7	Unit
$q_{\max}$	170	120	120	170	120	360	°
$q_{\min}$	-170	-120	-120	-170	-120	-360	°
$\dot{q}_{\max}$	2.175	2.175	2.175	2.610	2.610	2.610	rad/s
$\ddot{q}_{\max}$	15	7.5	10	15	15	20	rad/s^2
$\dddot{q}_{\max}$	5000	3500	5000	7500	7500	7500	rad/s^3

xMatePro3、xMatePro7 直接力矩控制限制条件

Name	Joint1	Joint2	Joint3	Joint4	Joint5	Joint6	Joint7	Unit
$\tau_{\max}$	85	85	85	85	36	36	36	Nm
$\dot{\tau}_{\max}$	1500	1500	1500	1500	1000	1000	1000	Nm/s

xMate3、xMate7 直接力矩控制限制条件

Name	Joint1	Joint2	Joint4	Joint5	Joint6	Joint7	Unit
$\tau_{\max}$	85	85	85	36	36	36	Nm
$\dot{\tau}_{\max}$	1500	1500	1500	1000	1000	1000	Nm/s

2.4.5 DH 参数

xMatePro3 DH 参数表:

Joint	A(mm)	Alpha(rad)	D(mm)	Theta(rad)
1	0	-pi/2	341.5	0
2	0	pi/2	0	0
3	0	-pi/2	394.0	0
4	0	pi/2	0	0
5	0	-pi/2	366	0
6	0	pi/2	0	0
7	0	0	250.3	0

xMatePro7 DH 参数表：

Joint	A(mm)	Alpha(rad)	D(mm)	Theta(rad)
1	0	-pi/2	404.0	0
2	0	pi/2	0	0
3	0	-pi/2	437.5	0
4	0	pi/2	0	0
5	0	-pi/2	412.5	0
6	0	pi/2	0	0
7	0	0	275.5	0

xMate3 DH 参数表：

Joint	A(mm)	Alpha(rad)	D(mm)	Theta(rad)
1	0	-pi/2	341.5	0
2	394	0	0	0
3	0	pi/2	0	0
4	0	-pi/2	366	0
5	0	pi/2	0	0
6	0	0	250.3	0

xMate7 DH 参数表：

Joint	A(mm)	Alpha(rad)	D(mm)	Theta(rad)
1	0	-pi/2	404	0
2	437.5	0	0	0
3	0	pi/2	0	0
4	0	-pi/2	412.5	0
5	0	pi/2	0	0
6	0	0	275.5	0

2.4.6 异常错误

RCI 运动过程中可能抛出以下几种异常：

异常类型	异常原因
NetworkException	TCP UDP 线程创建异常或网络连接异常
UdpException	udp 数据接收超时或者数据长度错误，通常是版本不匹配
TcpException	通常需要检查 IP 和端口设置
EventException	创建或添加回调 event 事件异常
ControlException	运动异常，通常是机器人控制器检测到 20 个 error 错误位异常
ProtocolException	接收到的 TCP response 未知错误
CommandException	接收到的 TCP response 返回设置失败
RealtimeException	实时环境配置错误
MotionException	MOVE 指令轨迹规划错误
std::invalid_argument	输入的运动指令为 NAN 或位姿矩阵不是齐次矩阵或数值误差问题

机器人在运动过程中会检测机器人状态，检测到异常会上报给 RCI，用户可以通过以下 20 个错误位

判断异常类型。

Errors	错误原因	解决方法
kActualJointPositionLimitsViolation：0 kActualCartesianPositionLimitsViolation：1 kActualCartesianMotionGeneratorElbowLimitViolation：2 kActualJointVelocityLimitsViolation：3 kActualCartesianVelocityLimitsViolation：4 kActualJointAccelerationLimitsViolation：5 kActualCartesianAccelerationLimitsViolation：6 kCommandJointPositionLimitsViolation：7 kCommandCartesianPositionLimitsViolation：8 kCommandCartesianMotionGeneratorElbowLimitViolation：9 kCommandJointVelocityLimitsViolation：10 kCommandCartesianVelocityLimitsViolation：11 kCommandJointAccelerationLimitsViolation：12 kCommandCartesianAccelerationLimitsViolation：13 kCommandJointAccelerationDiscontinuity：14 kCommandTorqueDiscontinuity：17 kCommandTorqueRangeViolation：18	实际轴角度超限 实际末端位姿超限 实际臂角超限 实际轴速度超限 实际末端速度超限 实际轴加速度超限 实际末端加速度超限 指令轴角度超限 指令末端位姿超限 指令臂角超限 指令轴速度超限 指令末端速度超限 指令轴加速度超限 指令末端加速度超限 指令轴加速度不连续 指令力矩不连续 指令力矩超限	检查规划轨迹是否连续可导，一般规划出来的轨迹需要做到速度和角速度连续可导，机器人运行不平稳或出现异响优先考虑轨迹是否平滑，必要时做额外的滤波处理。
kCollision：15	检测到碰撞	若频繁触发碰撞检测，应适当调高碰撞检测阈值，急停触发也有可能触发此报错
kCartesianPositionMotionGeneratorInvalidFrame：16	机器人奇异	笛卡尔空间运动时机器人不应该经过奇异位姿
kInstabilityDetection：19	检测到不稳定	1 检查丢包阈值是否过于低，一般设置为 10~20； 2 伺服报错，通常需要重启机器人； 3 按下了急停开关也会触发此错误；

2.5 使用示例

2.5.1 示例一

通过 RCI 实现机器人笛卡尔坐标系下的位置移动

```
1.  int main(int argc, char *argv[]) {
2.      SetConsoleOutputCP(CP_UTF8);
3.      std::string ipaddr = "192.168.66.128";
4.      uint16_t port = 1337;
5.      xmate::Robot robot(ipaddr, port, XmateType::XMATE7_PRO);
6.      try {
7.          //Sleep(10000);
8.          robot.connect();
9.          }
10.         catch (...) {
```

```
            do{
                std::cout << "reconnect" << std::endl;
            } while (robot.reconnect());
        }
        try {
            robot.setMotorPower(1);
        }
        catch (xmate::TcpException &e) {
            std::cout << "exception:" << e.what() << std::endl;
        }

    const double PI = 3.14159;
    std::array<double, 7> q_init;
    std::array<double, 7> q_drag = { { 0,PI / 6,0,PI / 3,0,PI / 2,0} };
    q_init = robot.receiveRobotState().q;
    MOVEJ(0.2, q_init, q_drag, robot);
    robot.setCoor({ {1.0,0.0,0.0,0.0,
        0.0,1.0,0.0,0.0,
        0.0,0.0,1.0,0.0,
        0.0,0.0,0.0,1.0} });
    cart_pos pos_start ,pos_end;
    pos_start.pos = robot.receiveRobotState().toolTobase_pos_m;
    pos_end = pos_start;
    pos_end.pos[11] -= 0.1;

    xmate::XmateModel model(&robot, XmateType::XMATE7_PRO);
    std::array<double, 7> q_out;
    int res = model.GetJointPos(pos_start.pos, 0.0, q_drag, q_out);
    try {
        MOVEL(0.1, pos_start, pos_end, robot, model);
    }
    catch (xmate::ControlException e ) {
        std::cout << "exception:" <<e.what()<< std::endl;
    }
    robot.startMove(RCI::robot::StartMoveRequest::ControllerMode::kCartesian
Position,
        RCI::robot::StartMoveRequest::MotionGeneratorMode::kCartesianPositio
n);

    uint64_t init_time;
    std::array<double, 16> init_position;
    static bool init = true;
    double init_psi;
    double time = 0;
```

```
53.
54.     CartesianControl cartesian_position_callback;
55.     cartesian_position_callback = [&](RCI::robot::RobotState robot_state) ->
    CartesianPose {
56.         time += 0.001;
57.         if (init == true) {
58.             init_position = robot_state.toolTobase_pos_m;
59.             init_psi = robot_state.psi_m;
60.             init = false;
61.         }
62.         constexpr double kRadius = 0.2;
63.         double angle = M_PI / 4 * (1 - std::cos(M_PI / 5 * time));
64.         double delta_x = kRadius * std::sin(angle);
65.         double delta_z = kRadius * (std::cos(angle) - 1);
66.
67.         CartesianPose output{};
68.         output.toolTobase_pos_c = init_position;
69.         output.toolTobase_pos_c[11] += delta_z;
70.         output.psi_c = init_psi;
71.
72.         return output;
73.     };
74.
75.     robot.Control(cartesian_position_callback);
76.
77.     return 0;
78. }
```

2.5.2 示例二

通过 RCI 实现机器人关节位置移动

```
1.  int main(int argc, char *argv[]) {
2.
3.      std::string ipaddr = "192.168.0.160";
4.      uint16_t port = 1337;
5.      std::string file = "../../xmate.ini";
6.      INIParser ini;
7.      if (ini.ReadINI(file)) {
8.          ipaddr = ini.GetString("network", "ip");
9.          port = static_cast<uint16_t>(ini.GetInt("network", "port"));
10.     }
11.     // RCI 连接机器人
12.     xmate::Robot robot(ipaddr, port, XmateType::XMATE7);
13.     //防止网络连接失败
```

```
    Sleep(2);
    int res = robot.getMotorState();
    std::cout << "机器人上电状态：" << res << std::endl;
    int power_state = 1;
    robot.setMotorPower(power_state);
    const double PI = 3.14159;
    std::array<double, 7> q_init;
    std::array<double, 7> q_drag = { { 0,PI / 6,PI / 3,0,PI / 2,0,0 } };
    q_init = robot.receiveRobotState().q;
    MOVEJ(0.2, q_init, q_drag, robot);
    std::array<double, 7> q_end = { { 0, 0, 0, 0, 0, 0,0 } };
    JointMotionGenerator joint_s(0.2, q_end);

    //开始运动前先设置控制模式和运动模式
    robot.startMove(RCI::robot::StartMoveRequest::ControllerMode::kJointPosition,
        RCI::robot::StartMoveRequest::MotionGeneratorMode::kJointPosition);

    std::array<double, 7> init_position;
    std::array<double, 7> joint_delta;
    JointPositions output{};
    static bool init = true;
    double time = 0;

    JointControl joint_position_callback;
    joint_position_callback = [&](RCI::robot::RobotState robot_state) -> JointPositions {
        time += 0.001;
        if (init == true) {
            init_position = robot_state.q;
            init = false;
        }

        joint_s.calculateSynchronizedValues_joint(init_position);
        //用户规划用的时间

        if (joint_s.calculateDesiredValues_joint(time, joint_delta) == false) {
            for (unsigned int i = 0; i<6; i++) {
                output.q[i] = init_position[i] + joint_delta[i];
            }
        }
        else {
            std::cout << "运动结束：" << std::endl;
```

```
54.             return MotionFinished(output);
55.         }
56.         return output;
57.     };
58.     robot.Control(joint_position_callback);
59.     return 0;
60. }
```

3 反馈与勘误

文档中若出现不准确的描述或者错误，恳请读者指正批评。如果您在阅读过程中发现任何问题或者有想提出的意见，可以发送邮件到 xuqiu@rokae.com,我们的同事会尽量一一回复。

ROKAE 珞石

轻 型 机 器 人 专 家

400-010-8700

北 京 总 部：北京市海淀区农科院西路6号海青大厦A座7层
山东分公司：济宁市邹城市中心店镇机电产业园恒丰路888号
苏州分公司：苏州工业园区星湖街328号创意产业园1-A1F
深圳分公司：深圳市宝安区中粮福安机器人智造产业园10栋1楼