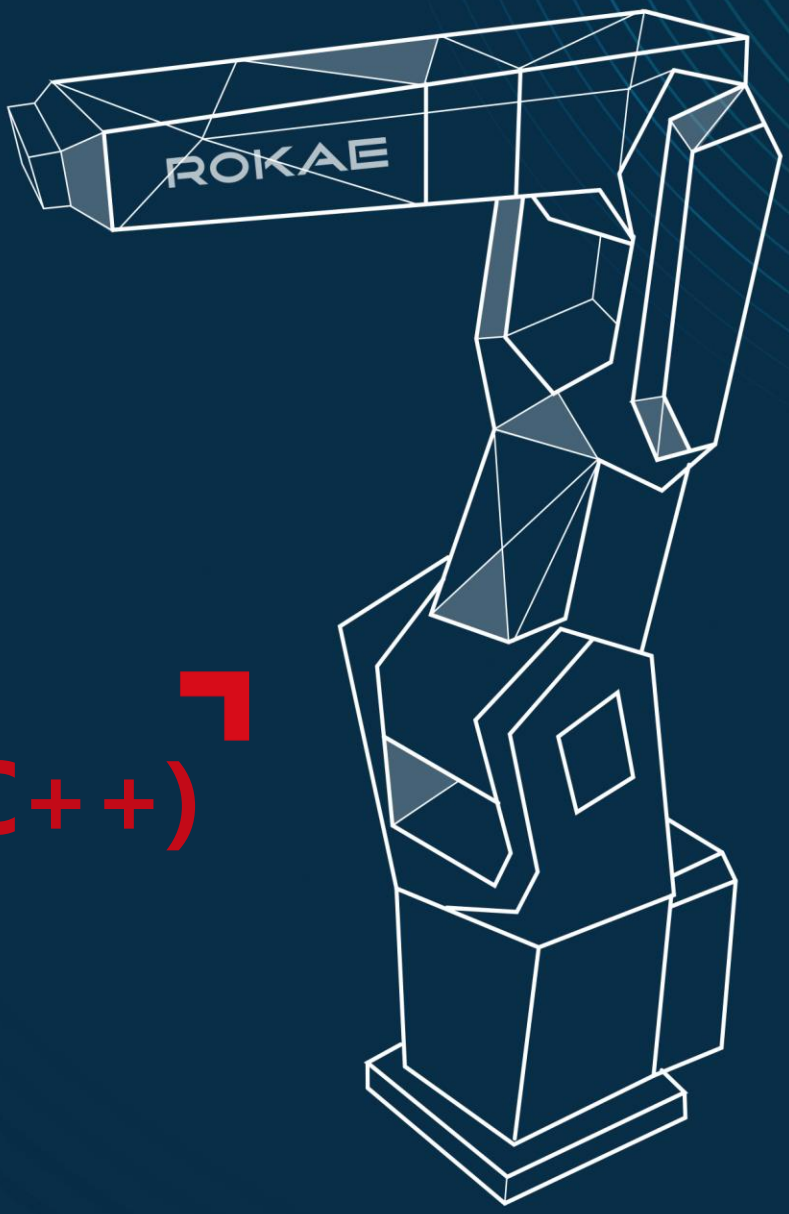
ROKAE 珞石

轻 型 机 器 人 专 家



xCore SDK(C++)

使用手册

让智造更高效

xCore SDK(C++)使用手册

[类别]

[备注]

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 SDK(C++)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基础的机器人操作、C++编程等所需的知识，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	初始版本；
V1.6	2022.09	Rokae SDK 初版，适配 xCore 版本 v1.6.2；
V1.7	2023.02	Rokae SDK 正式版本，适配 xCore 版本 v1.7；
V1.8	2023.06	新增接口，优化和修复一些问题，适配 xCore 版本 v2.0;

2 概述

xCore SDK 编程接口库是珞石机器人提供给客户用于二次开发的软件产品，通过编程接口库，客户可以对配套了 xCore 系统的机器人进行一系列控制和操作，包括实时和非实时的运动控制，机器人通信相关的读写操作，查询及运行 RL 工程，等等。该使用说明书主要介绍编程接口库的使用方法，以及各接口函数的功能。用户可编写自己的应用程序，集成到外部软硬件模块中。

2.1 兼容性

2.1.1 控制器版本和机器人型号

- 控制器版本：xCore v2.0 及以后。
- 机器人型号：支持控制所有机型，根据协作和工业机器人支持的功能不同，可调用的接口有所差别。

2.1.2 编译平台及语言

操作系统	编译器	平台	语言
Ubuntu 18.04/20.04/22.04	build-essential	X86_64	C++, Python
Windows 10	MSVC 14.1+	X86_64	C++, Python, C#
Android		armeabi-v7a, arm64-v8a, x86，x86_64	Java

2.2 非实时控制

xCore SDK 提供对机器人的非实时控制，主要通过给机器人发送运动指令，使用控制器内部的轨迹规划，完成路径规划和运动执行。非实时模式提供的操作有：

- 轴空间运动（MoveAbsJ, MoveJ）
- 笛卡尔空间运动（MoveL, MoveC, MoveCF）
- 机器人通信: 数字量和模拟量 I/O，寄存器读写
- RL 工程的查询与执行
- 拖动与路径回放（只针对 xMate 协作机器人）
- 其他操作：设置碰撞检测，清除报警，查询控制器日志等等

2.3 实时控制

xCore SDK 的实时控制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。

xMate 协作机器人支持 5 种控制模式：

- 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

6 轴工业机器人支持 2 种位置控制模式：

- 轴空间位置控制
- 笛卡尔空间位置控制

工业 3、4 轴机型暂不支持。

3 使用指南

本章介绍如何配置并运行一个 xCore SDK C++程序。
其他语言版本（Python、Java）请参考分手册。

3.1.1 硬件设置

关于机器人本体和控制柜等硬件的设置，请参考《xCore 机器人控制系统使用手册 V2.0》。除网络配置外，使用 xCore SDK 无需其他额外的硬件设置。

3.1.2 网络配置

xCore SDK 通过以太网（TCP/IP）连接机器人。通过有线或无线连接皆可，使用户 PC 和机器人连接同一局域网。如果只使用非实时控制，对于网络性能要求不高，可以通过无线连接。
使用实时控制的话推荐通过有线直连到机器人。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。

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。

3.1.3 机器人功能设置

无需通过 RobotAssist 进行任何设置，用户可直接用 xCore SDK 控制机器人。
切换到实时模式后，机器人重启后会保持打开状态，并自动切换成自动模式。

3.1.4 xCore SDK 工程包说明

```
librokae
├── doc: 文档和使用手册
├── example: 示例程序
├── external:  Eigen
├── include: 头文件
└── lib: 各操作系统和架构的库文件
```

3.1.5 C++编译

xCore SDK 版本使用 CMake 构建工程，CMake 版本不低于 3.12。

3.1.5.1 Linux 平台编译

以编译示例程序 sdk_example 为例，设置安装路径为根目录下 out:

```
cd librokae
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=../out
cmake --build . --target sdk_example
cmake --build . --target install
```

3.1.5.2 Windows 平台编译

1. 下载并安装 Microsoft Visual Studio 2017 或以后版本，选择安装“使用 C++的桌面应用”。
2. 打开 CMake 工程，选择根目录下的 CMakeLists.txt
3. 选择编译 Release 或 Debug，编译示例程序

3.1.5.3 QT 平台编译

1. 下载并安装 Qt 5.15.2 或以后版本，并将编译器勾选为 MSVC2019 编译器；
2. 将 SDK 工程包保存到本地（无中文路径）；
3. 创建新工程文件，并将编译器选用为 MSVC2019；
4. 进入配置文件（.pro 文件），输入以下配置语句：

```
LIBS += -L+工程包 lib 文件所在路径 -lRokaeSDK

INCLUDEPATH += 工程包 include 文件所在路径
DEPENDPATH += 工程包 include 文件所在路径
```

3.1.6 Linux 实时环境配置（可选）

xCore 控制器实时模式接收运动命令的周期为 1ms，客户端需保证至少 1kHZ 的发送周期。如果计算量较大，推荐安装实时内核。

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 kernel；

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
```

4. 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。

4 接口说明

本章列出各版本 xCore SDK 所支持的接口和功能简述。不同开发语言的版本对接口的功能定义基本一致，但是参数、返回值和调用方法会有区别。

4.1 API 支持

下表是各语言版本接口支持情况概览。

模块	API 功能	C++	Python & C#	Android
rokae::Robot	基本操作	全部支持	全部支持	全部支持
	非实时运动	全部支持	全部支持	全部支持
	Jog 机器人	全部支持	支持	不支持
	通信	全部支持	部分支持	部分支持
	RL 工程	全部支持	不支持	不支持
	协作相关	全部支持	部分支持	全部支持
rokae::Model	运动学计算	全部支持	部分支持	不支持
rokae::RtMotionControl	实时模式	全部支持	不支持	不支持
rokae::Planner	上位机路径规划	全部支持	不支持	不支持
rokae::xMateModel	运动学和动力学计算	全部支持	不支持	不支持

4.2 C++：实例化 rokae::Robot 类

根据机器人构型和轴数不同，C++版本的 SDK 提供了下列几个可供实例化的 Robot 类，初始化时会检查所选构型和轴数是否和连接的机器人匹配：

类名	适用机型
xMateRobot	协作 6 轴
xMateErProRobot	协作 7 轴
StandardRobot	工业 6 轴
PCB4Robot	工业 4 轴
PCB3Robot	工业 3 轴

4.3 机器人基本操作及信息查询

简述	接口	参数	返回
连接机器人	connectToRobot()		
断开连接	disconnectFromRobot()		
查询机器人基本信息	robotInfo()		控制器版本，机型，轴数
查询上电状态	powerState()		on/off/Estop/Gstop
机器人上下电	setPowerState(state)	state - on/off	
查询当前操作模式	operateMode()		auto/manual
切换手自动模式	setOperateMode(mode)	mode - auto/manual	
查询机器人运行状态	operationState()		idle/jog/RLprogram/moving 等状态
获取当前末端/法兰位姿	posture()		[X, Y, Z, Rx, Ry, Rz]
获取当前末端/法兰位姿	cartPosture()		[X, Y, Z, Rx, Ry, Rz] 及轴配置参数
获取当前关节角度	jointPos()		各轴角度 rad
获取当前关节速度	jointVel()		各轴速率 rad/s

获取关节力矩	jointTorque()		各轴力矩 Nm
查询基坐标系	baseFrame()		[X, Y, Z, Rx, Ry, Rz]
查询当前工具工件组	toolset()		末端坐标系，参考坐标系，负载信息
设置工具工件组	setToolset(toolset) setToolset(toolName, wobjName)	toolset – 工具工件组信息 toolName – 工具名字 wobjName – 工件名字	
计算逆解	calcIk(posture)	posture – 末端相对于外部参考坐标系位姿	关节角度
计算正解	calcFk(joints)	joints – 关节角度	末端相对于外部参考坐标系位姿
清除伺服报警	clearServoAlarm()		
查询控制器日志	queryControllerLog(count, level)	count - 查询个数 level - 日志等级	
置碰撞检测相关参数，打开碰撞检测功能	enableCollisionDetection (sensitivity, behaviour, fallback)	sensitivity – 灵敏度 behaviour – 碰撞后行为 fallback – 回退距离	
关闭碰撞检测功能	disableCollisionDetection()		
查询 SDK 版本号	sdkVersion()		版本号

4.4 运动控制

非实时模式运动控制相关接口。

简述	接口	参数	返回
设置运动控制模式	setMotionControlMode(mode)	mode - NRT/RT/RL 工程	
开始/继续运动	moveStart()		
重置运动缓存	moveReset()		
停止机器人运动	stop()		
添加运动指令	moveAppend(command, id)	command - 一条或多条 MoveL/MoveJ/MoveAbsJ/ MoveC/MoveCF 指令 id – 指令 ID，用于执行信息反馈	
设置默认运动速度	setDefaultSpeed(speed)	speed - 末端最大线速度	
设置默认转弯区	setDefaultZone(zone)	zone - 转弯区半径	
开始 Jog 机器人	startJog(space, rate, step, index, direction)	space - 参考坐标系 rate - 速率 step - 步长 index - XYZABC/J1-7 direction - 方向	
动态调整机器人运动速率	adjustSpeedOnline(scale)	scale – 速率	
设置接收事件的回调函数	setEventWatcher(eventType, callback)	eventType – 事件类型，目前只用于运动指令执行结果 callback – 处理事件的回调函数	
查询事件信息	queryEventInfo(eventType)	eventType – 事件类型	
执行运动指令	executeCommand(command)	command - 一条或多条 MoveL/MoveJ/MoveAbsJ/	

		MoveC 指令	
(已弃用) 运动指令执行错误	lastErrorCode()		

4.5 实时运动控制

简述	接口	参数	返回
重新连接实时控制服务器	reconnectNetwork()		
断开连接	disconnectNetwork()		
设置周期调度	setControlLoop(callback, priority, useStateDataInLoop)	callback - 回调函数 priority - 线程优先级 useStateDataInLoop – 是否在回调中读取实时状态数据	
开始执行调度任务	startLoop(blocking)	blocking – 是否阻塞	
停止调度任务	stopLoop()		
开始运动	startMove(mode)	mode – 控制模式	
停止运动	stopMove()		
开始接收实时状态数据	startReceiveRobotState(timeout, fields)	timeout – 超时等待时间 fields – 接收的数据列表	
停止接收实时状态数据	stopReceiveRobotState()		
更新机器人状态数据到当前最新	updateRobotState(timeout)	timeout – 超时等待时间	
获取机器人状态数据	getStateData(name, data)	name - 数据名 data – 数据值	
PTP 轴空间规划运动	MoveJ(speed, start, target)	speed – 速度系数 start – 起始关节角度 target – 目标关节角度	
PTP 笛卡尔空间直线规划运动	MoveL(speed, start, target)	speed – 速度系数 start – 起始位姿 target – 目标位姿	
3 点圆弧规划运动	MoveC(speed, start, aux, target)	speed – 速度系数 start – 起始位姿 aux – 辅助点位姿 target – 目标位姿	
设置限幅滤波参数	setFilterLimit(limit, frequency)	limit – 是否开启限幅 frequency - 截止频率	
设置笛卡尔空间运动区域	setCartesianLimit(length, frame)	length - 区域长宽高 frame - 区域中心坐标系	
设置关节阻抗系数	setJointImpedance(factor)	factor – 各关节阻抗系数	
设置笛卡尔空间阻抗控制系数	setCartesianImpedance(factor)	factor - 系数	
设置碰撞检测阈值	setCollisionBehaviour(threshold)	threshold – 各关节阈值	
设置末端执行器位姿	setEndEffectorFrame(frame)	frame - 末端相对于法兰的位姿	
设置负载	setLoad(load)	load - 负载信息	
设置滤波截止频率	setFilterFrequency(joint, cart, torque)	joint - 关节位置截止频率 cart - 笛卡尔空间位置截止频率	

		torque - 关节力矩截止频率	
设置笛卡尔空间阻抗控制末端期望力	setCartesianImpedanceDesiredTorque(torque)	torque - 末端期望力	
设置滤波参数	setTorqueFilterCutoffFrequency(frequency)	frequency - 频率	
设置力控坐标系	setFcCoor(frame, type)	frame – 坐标系 type – 力控任务坐标系类别	
发生错误后自动恢复机器人	automaticErrorRecovery()		
设置网络延迟阈值	setNetworkTolerance(percent)	percent – 阈值半分比	
切换使用 RCI 客户端	useGen1RciClient(use)	use – 是否切换	

路径规划相关

简述	类
S 速度规划的笛卡尔空间运动	CartMotionGenerator
S 速度规划的轴空间运动	JointMotionGenerator
点位跟随，点位可以是笛卡尔位姿或轴角度	FollowPosition

4.5.1 参数设置

设置开始运动前的参数。所有参数都只用于实时模式运动，与非实时运动、通过 RobotAssist 操作的运动均不互相影响。

4.5.2 发送运动指令

通过 setControlLoop()设置回调函数，函数内计算出每周期的运动指令，作为函数返回值返回。SDK 将返回的指令滤波后发送给控制器。指令的数据类型（关节角度/笛卡尔位姿/关节力矩）应与控制模式匹配。控制器的控制周期是 1ms。控制器的监测窗口是 2 秒，根据网络延迟阈值的高低，如果在监测窗口内没有收到足够的指令，将返回“网络不稳定”错误，然后做停止运动并下电处理。

4.5.3 获取实时数据

在开始运动之前，通过 startReceiveRobotState()设置要接收的数据和控制器发送的时间间隔。根据是否在上述的回调函数中读取状态数据，更新的方式也不同。如果需要在回调函数中读取状态数据，为保证 1ms 的控制周期，xCore-SDK 会在每次执行回调前更新状态数据，在回调函数内部直接调用 getStateDate()读取。

如果不需要在回调函数里读取状态数据，则需要用户程序按照设置的发送周期调用 updateStateData()更新状态数据，然后再通过 getStateDate()接口读取。

4.5.4 运动学和动力学计算

SDK 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，目前支持 xMateER 系列所有机型, xMateCR7/12, xMateSR3/4；兼容性上支持 Linux x86_64、Windows 64bit Release 编译类型。主要接口功能包括：运动学正解 getCartPose()，运动学逆解 getJointPos()，雅可比矩阵计算 jacobian()，动力学正解 getTorque()。

详见附录 A - C++ API。

4.5.4.1 编译及使用

使用运动学与动力学计算库需要在编译时加上下列选项：

```
cmake .. -DXCORE_USE_XMATE_MODEL=ON
```

然后通过 robot.model()接口获取。

4.5.5 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(q_start);  //指定初始关节角度 q_start，并做同步处理;
joint_s.calculateDesiredValues(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 SDK 中 planner.h 文件。

4.5.6 错误异常

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 SDK，用户可以通过以下 20 个错误位判断异常类型。

错误位	错误名称	错误原因	解决方法
0	kActualJointPositionLimitsViolation	实际轴角度超限	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。
1	kActualCartesianPositionLimitsViolation	实际末端位姿超限	
2	kActualCartesianMotionGeneratorElbowLimitViolation	实际臂角超限	
3	kActualJointVelocityLimitsViolation	实际轴速度超限	
4	kActualCartesianVelocityLimitsViolation	实际末端速度超限	
5	kActualJointAccelerationLimitsViolation	实际轴加速度超限	
6	kActualCartesianAccelerationLimitsViolation	实际末端加速度超限	
7	kCommandJointPositionLimitsViolation	指令轴角度超限	
8	kCommandCartesianPositionLimitsViolation	指令末端位姿超限	
9	kCommandCartesianMotionGeneratorElbowLimitViolation	指令臂角超限	
10	kCommandJointVelocityLimitsViolation	指令轴速度超限	
11	kCommandCartesianVelocityLimitsViolation	指令末端速度超限	
12	kCommandJointAccelerationLimitsViolation	指令轴加速度超限	
13	kCommandCartesianAccelerationLimitsViolation	指令末端加速度超限	
14	kCommandJointAccelerationDiscontinuity	指令轴加速度不连续	
17	kCommandTorqueDiscontinuity	指令力矩不连续	
18	kCommandTorqueRangeViolation	指令力矩超限	
15	kCollision	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
16	kCartesianPositionMotionGeneratorInvalidFrame	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
19	kInstabilityDetection	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;

4.6 通信相关

简述	接口	参数	返回值
查询 DI 信号值	getDI(board, port)	board - IO 板序号 port - 信号端口号	on | off
设置 DI 信号值	setDI(board, port, state)	board - IO 板序号 port - 信号端口号 state - 信号值	
查询 DO 信号值	getDO(board, port)	board - IO 板序号 port - 信号端口号	on | off
设置 DO 信号值	setDO(board, port, state)	board - IO 板序号 port - 信号端口号 state - 信号值	
查询 AI 信号值	getAI(board, port)	board - IO 板序号 port - 信号端口号	信号值
设置 AO 信号	setAO(board, port, value)	board - IO 板序号 port - 信号端口号 value - 信号值	
设置输入仿真模式	setSimulationMode(state)	state – 打开/关闭	
读取寄存器值	readRegister(name, index, value)	name - 寄存器名称 index - 寄存器数组索引 value - 读取的数值	
写入寄存器值	writeRegister(name, index, value)	name - 寄存器名称 index - 寄存器数组索引 value - 写入的数值	
设置 xPanel 对外供电模式	setxPanelVout(opt)	opt – 模式	

4.7 RL 工程

控制器中需要有已创建好的 RL 工程，支持查询工程信息和运行。

简述	接口	参数	返回值
查询 RL 工程列表	projectInfo()		工程名称和任务名
加载工程	loadProject(name, tasks)	name - 工程名称 tasks – 任务列表	
pp-to-main	ppToMain()		
开始运行工程	runProject()		
暂停运行工程	pauseProject()		
设置运行速率和循环模式	setProjectRunningOpt(rate, loop)	rate – 运行速率 loop – 循环/单次	
查询工具信息	toolsInfo()		工具名称，位姿，负载等信息
查询工件信息	wobjsInfo()		工件名称，位姿，负载等信息

4.8 协作相关

包括拖动示教和路径录制相关功能。

简述	接口	参数	返回值
打开拖动	enableDrag(space, type)	space – 拖动空间	

		type – 拖动类型	
关闭拖动	disableDrag()		
开始录制路径	startRecordPath(duration)	duration – 录制时长	
停止录制路径	stopRecordPaht()		
取消录制	cancelRecordPath()		
保存路径	saveRecordPath(name, saveAs)	name – 路径名称 saveAs – 重命名为	
路径回放	replayPath(name, rate)	name – 路径名称 rate – 回放速率	
删除保存的路径	removePath(name, all)	name – 路径名称 all – 是否删除所有路径	
查询路径列表	queryPathLists()		路径名称列表

4.9 错误码和异常

非实时接口的调用结果通过错误码反馈，每个接口都会传入一个错误码 ec，可通过 ec.message()来获取错误码对应的信息。

实时模式下发送运动指令、周期调度、读取状态数据等接口在调用过程中会抛出异常，异常类型见 rokae/exception.h。

5 注意事项与问题排查

5.1 与 RobotAssist 同时使用

目前如果使用 xCore SDK 控制机器人，并不会限制通过 RobotAssist 的控制。机器人的一些状态，通过 xCore SDK 更改后也会体现在 Robot Assist 界面上；一些工程运行，运动控制则是分离的。大致总结如下：

会同步更新的组件	● 底部状态栏：手自动，机器人状态，上下电； ● 状态监控窗口； ● 日志上报；
双方可控，即通过 RobotAssist 修改会生效的组件	● RL 工程运行速率和循环/单次模式； ● 非实时模式运动的停止；
不会同步更新的组件，包括但不限于	● 下发的运动指令无法通过点击开始按钮让机器人开始执行； ● 加载的 RL 工程，以及前瞻指针、运动指针等，不会同步显示； ● 所有机器人设置界面显示的功能打开状态和设定值等；

建议的控制方式是单一控制源，避免混淆。对于运动指令，推荐在每次使用 SDK 下发指令之前调用 moveReset() 接口来重置运动缓存。

5.2 兼容 RCI 客户端

对于原 RCI 客户端的用户，可以在控制器升级到 v2.0（v1.7 也同样可以）之后直接使用 RokaeSDK。

5.2.1 首次使用

1. 确保 RobotAssist – 通信 – RCI 设置是关闭的状态，也可通过状态栏中间的机器人状态确认；
2. 调用 setMotionControlMode(RtCommand) 接口来打开 RCI 功能，若打开成功，机器人状态变为 RCI，状态栏的图标也显示 RCI：

3. 之后就可通过上述接口或 RobotAssist 打开关闭 RCI 功能。需要注意的是，如果进行了抹除配置操作，那么同样视为首次使用，需要再通过 SDK 的接口打开 RCI。

5.2.2 切换到使用 RCI 客户端

在用过 SDK 后，RCI 客户端就无法同时使用了。如果想切换回去，需调用 useRciClient(true)，调用前按照函数说明，确保 RCI 是关闭的状态。然后再通过 RobotAssist 打开关闭 RCI 功能。

5.3 实时指令

用户下发的运动指令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。

5.3.1 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：

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

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:

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

运动结束时:

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

5.3.2 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:

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

5.3.3 力矩直接控制

1. 必要条件

力矩指令不超限且连续:

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

5.3.4 运动限制条件

xMateER3 Pro、xMateER7 Pro、xMateER3、xMateER7 笛卡尔空间运动限制条件:

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

xMateER3 Pro 和 xMateER7 Pro 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 和 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

xMateER3 Pro 和 xMateER7 Pro 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 和 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

5.3.5 DH 参数

xMateER3 Pro 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

xMateER7 Pro 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

5.4 问题排查

5.4.1 网络连接问题

实时运动指令和状态信息都通过 UDP 协议单播发送。如果出现“超时前未收到机器人状态反馈“的实时状态异常，Windows 系统请检查防火墙设置，UDP 是否是允许的状态。

Linux 系统(Debian 发行版): 打开终端，输入命令‘‘nc –vul –p 1337’’，然后再任意运行一个实时模式控制的示例程序，查看是否有来自机器人地址的端口连接过来，并且收到了数据。

5.4.2 实时模式直接力矩控制下坠问题

力控和机器本身硬件状态有关系。如果出现开启直接力矩控制之后没有发任何指令，但是机器人下坠或者发飘等问题，可能是由于力控模型不准的原因。首先需要确认各项力控参数是否准确；其次可以根据实机情况调整发送的运动指令。

6 使用示例

本章展示一些 C++示例程序，更多示例请见软件包中 examples。

6.1 非实时接口

6.1.1 示例一：信息查询，Jog，拖动等

```
int main() {
  try {
    // *** 1. 连接机器人 ***
    std::string ip = "192.168.0.160";
    std::error_code ec;
    xMateErProRobot robot(ip); // 此处连接的是 xMatePro 机型

    // 其它机型
    // xMateRobot robot(ip); // 连接 xMate6 轴机型
    // StandardRobot robot(ip); // 连接工业 6 轴机型

    // *** 2. 查询信息 ***
    auto robotinfo = robot.robotInfo(ec);
    print(os, "控制器版本号:", robotinfo.version, "机型:", robotinfo.type);
    print(os, "xCore-SDK 版本:", robot.sdkVersion());
    auto model = robot.model();

    // *** 3. 获取机器人当前位姿，轴角度，基坐标系等信息 ***
    auto joint_pos = robot.jointPos(ec);
    auto joint_vel = robot.jointVel(ec);
    auto joint_torque = robot.jointTorque(ec);
    auto flan = robot.posture(CoordinateType::flangeInBase, ec);
    auto tcp = robot.posture(CoordinateType::endInRef, ec);
    auto base = robot.baseFrame(ec);
    print(os, "法兰相对基坐标系位姿", flan, "\n 末端相对外部参考坐标系位姿", tcp);

    // *** 4. 计算逆解 ***
    auto ik = model.calcIk(tcp, ec);

    // *** 5. 查询 IO/寄存器 ***
    std::cout << "DO1_0 当前信号值为: " << robot.getDO(1,0,ec) << std::endl;
    robot.setSimulationMode(true, ec); // 只有在打开输入仿真模式下才可以设置 DI
    robot.setDI(0, 2, true, ec);
    print(os, "DI0_2 当前信号值:", robot.getDI(0, 2, ec));
    robot.setSimulationMode(false, ec); // 关闭仿真模式

    // 读取单个寄存器，类型为 float
    // 假设"register0"是个寄存器数组，长度是 10
```

```
    float val_f;
    std::vector<float> val_af;
    // 读第 1 个，即状态监控里的 register0[1]，读取结果赋值给 val_f
    robot.readRegister("register0", 0, val_f, ec);
    // 读第 10 个，即状态监控里的 register0[10]，读取结果赋值给 val_f
    robot.readRegister("register0", 9, val_f, ec);
    // 读整个数组，赋值给 val_af，val_af 的长度也变为 10。此时 index 参数是多少都无所谓
    robot.readRegister("register0", 9, val_af, ec);

    // 读取 int 类型寄存器数组
    std::vector<int> val_ai;
    robot.readRegister("register1", 1, val_ai, ec);
    // 写入 bool/bit 类型寄存器
    robot.writeRegister("register2", 0, true, ec);

    // *** 6. 断开连接再重连 ***
    robot.disconnectFromRobot(ec);
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    robot.connectToRobot(ec);

    // *** 7. 打开，关闭拖动 ***
    robot.setOperateMode(rokae::OperateMode::manual, ec);
    robot.setPowerState(false, ec); // 自动模式下上电
    robot.enableDrag(DragParameter::cartesianSpace, DragParameter::freely, ec);
    std::cout << "打开拖动结果: " << ec.message() << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(2)); //等待切换控制模式
    auto power = robot.powerState(ec);
    std::cout << "上下电状态: " << static_cast<int>(power)<< std::endl; // 打开拖动后应自动上电
    while(getchar() != '\n');
    robot.disableDrag(ec);
    std::this_thread::sleep_for(std::chrono::seconds(2)); //等待切换控制模式


    // *** 8. Jog 机器人 ***
    robot.setMotionControlMode(rokae::MotionControlMode::NrtCommand, ec);
    robot.setOperateMode(rokae::OperateMode::manual, ec); // 手动模式下 jog
    std::cout << "准备 Jog 机器人，需手动模式上电，请在 10 秒内按住使能开关\n";
    std::this_thread::sleep_for(std::chrono::seconds(10));

    std::cout << "-- 开始 Jog 机器人-- \n 世界坐标系下，沿 Z+方向运动 50mm，速率 50%，等待机器人停止运动后
按回车继续\n";
    robot.startJog(JogOpt::world, 0.5, 50, 2, true, ec);
    while(getchar() != '\n');
    std::cout << "轴空间，6 轴负向连续转动，速率 5%，按回车停止 Jog\n";
```

```
    robot.startJog(JogOpt::jointSpace, 0.05, 5000, 5, false, ec);
    while(getchar() != '\n'); // 按回车停止
    robot.stop(ec); // jog 结束必须调用 stop()停止

    // *** 9. 运动指令，设置运动控制模式为非实时运动指令 ***
    robot.setMotionControlMode(MotionControlMode::NrtCommand, ec);
    robot.setOperateMode(OperateMode::automatic, ec); // 切换到自动模式
    robot.setPowerState(true, ec); // 自动模式下上电
    robot.moveReset(ec); // 发运动指令前必须重置缓存

    // MoveL
    std::string id;
    rokae::MoveLCommand movel0({0.563,0.0,0.432,3.1415,0,3.14159},4000,100);
    rokae::MoveLCommand movel1({0.33467,-0.095,0.510,3.14159,-0,3.14159},4000,100);
    robot.moveAppend({movel0, movel1}, id, ec); // 发送指令
    robot.moveStart(ec); // 机器人开始运动
    WaitRobot(&robot);
    robot.stop(ec); // 停止运动

    // MoveAbsJ
    MoveAbsJCommand abs({-0.1,0.597,0,1.129,-1.32,1.57,0});
    robot.moveAppend({abs}, id, ec);
    // MoveC
    MoveCCommand cir({0.559,-0.0766,0.385,-2.877,-0.031,3.122},
                     {0.549,-0.2,0.49,-2.397,-0.08745,3.085});
    robot.moveAppend({cir}, id, ec);
    robot.moveStart(ec);

    // 客户端程序执行结束前会强制机器人停止运动。故程序需要等待机器人运动结束
    WaitRobot(&robot);

    // *** 10. 结束控制，断开连接 ***
    robot.setPowerState(false, ec); // 机器人下电
    robot.disconnectFromRobot(ec);

  } catch (const rokae::Exception &e) {
    std::cout << e.what();
  }

  return 0;
}
```

6.1.2 示例二：机器人运动圆形轨迹

```
int main(){
  using namespace rokae;
  try{
    XMateErProRobot robot("192.168.0.160"); // 连接到xMateEr Pro机器人
    error_code ec;

    // *** 开始运动前的设置 ***
    robot.setOperateMode(OperateMode::automatic, ec);  // 切换到自动模式
    robot.setMotionControlMode(MotionControlMode::NrtCommand, ec); //设置非实时模式执行运动指令
    robot.setPowerState(true, ec); //上电
    robot.moveReset(ec); //重置运动缓存
    robot.setDefaultSpeed(1000, ec); //设定默认运动速度v1000
    robot.setDefaultZone(5, ec); // 设定默认转弯区5
    std::string id;


    // *** 运动到起始点 ***
    MoveAbsJCommand drag({0, M_PI/6, 0, M_PI/3, 0, M_PI/2, 0}); // 选择拖拽位姿作为中心点,
MoveAbsJ运动至中心点
    robot.moveAppend({drag}, id, ec);

    // ***  第一个圆 ***
    MoveLCommand p1({0.69, 0, 0.507, 3.14, 0, 3.14}); // 以中心点为向x轴正方向移动0.06米为半径,
开始画圆
    robot.moveAppend ({p1}, id, ec);
    // 前半圆和后半圆
    MoveCCommand firstHalf({0.57125, 0, 0.507, 3.14, 0, 3.141}, {0.63, -0.06, 0.507, 3.14,
0, 3.14}),
    secondHalf({0.69, 0, 0.507, 3.14, 0, 3.14}, {0.63, 0.06, 0.507, 3.14, 0, 3.14});
    robot.moveAppend({firstHalf, secondHalf}, id, ec);

    robot.moveStart(ec);// 开始执行

    while(getchar() != '\n'); // 按回车等待运行结束，亦可通过检查机器人状态来判断是否运行结束

    robot.setPowerState(false, ec);  // 机器人下电
    robot.setOperateMode(OperateMode::manual, ec); // 切换到手动模式
  }catch (Exception &e){
    cout << e.what();
  }
  return 0;
}
```

6.2 实时运动控制

6.2.1 示例一：笛卡尔空间阻抗控制

```
int main() {
  using namespace std;
  try {
    std::string ip = "192.168.0.160";
    std::error_code ec;
    rokae::xMateErProRobot robot(ip, "192.168.0.100"); // 本机地址 192.168.0.100

    robot.setOperateMode(rokae::OperateMode::automatic,ec);
    robot.setMotionControlMode(MotionControlMode::RtCommand, ec);
    robot.setPowerState(true, ec);

    auto rtCon = robot.getRtMotionController().lock();

    // 设置要接收数据。其中 jointPos_m 是本示例程序会用到的
    rtCon->startReceiveRobotState({RtSupportedFields::jointPos_m,
RtSupportedFields::tauVel_c,
                           RtSupportedFields::tcpPose_m, RtSupportedFields::elbow_m});

    // 读取实时状态数据：力矩微分
    rtCon->updateRobotState();
    std::array<double,7> array7 {};
    rtCon->getStateData(RtSupportedFields::tauVel_c, array7);

    std::array<double, 16> init_position {};
    static bool init = true;
    double time = 0;

    rtCon->getStateData(RtSupportedFields::jointPos_m, array7);
    std::array<double,7> q_drag_xm7p = {0, M_PI/6, 0, M_PI/3, 0, M_PI/2, 0};

    // 获取机器人当前轴角度(需要设置"jointPos_m"为要接收的状态数据)
    // 从当前位置 MoveJ 运动到拖拽位姿
    rtCon->MoveJ(0.5, array7, q_drag_xm7p);

    // 设置笛卡尔空间阻抗系数，开始笛卡尔空间阻抗运动
    rtCon->setCartesianImpedance({1000, 1000, 1000, 100, 100, 100}, ec);
    rtCon->startMove(RtControllerMode::cartesianImpedance);
    std::atomic<bool> stopManually { true };

    std::function<CartesianPosition()> callback = [&, rtCon] {
        time += 0.001;
        if(init) {
          rtCon->getStateData(RtSupportedFields::tcpPose_m, init_position);
```

```
        init = false;
      }
      constexpr double kRadius = 0.2;
      double angle = M_PI / 4 * (1 - std::cos(M_PI / 2 * time));
      double delta_z = kRadius * (std::cos(angle) - 1);

      CartesianPosition output{};
      output.pos = init_position;
      output.pos[7] += delta_z;

      if(time > 20) {
        output.setFinished(); // 20 秒后结束
        stopManually.store(false);  // loop 为非阻塞，和主线程同步停止状态
      }
      return output;
    };

    rtCon->setControlLoop(callback);
    // 非阻塞 loop
    rtCon->startLoop(false);
    while(stopManually.load());
    } catch (const std::exception &e) {
      std::cout << e.what();
  }
  return 0;
}
```

6.2.2 示例二：上位机轨迹规划指令组合

```
int main() {
  using namespace std;
  try {
    std::string ip = "192.168.0.160";
    std::error_code ec;
    rokae::xMateErProRobot robot(ip, "192.168.0.180"); // ****  XMate 7-axis
    robot.setOperateMode(rokae::OperateMode::automatic,ec);
    robot.setRtNetworkTolerance(20, ec);
    robot.setMotionControlMode(MotionControlMode::RtCommand, ec);
    robot.setPowerState(true, ec);

    auto rtCon = robot.getRtMotionController().lock();
    print(os, "Start receive");
    robot.startReceiveRobotState(std::chrono::milliseconds(1),
{RtSupportedFields::jointPos_m, RtSupportedFields::elbow_m, RtSupportedFields::tcpPose_m});
```

```
    // 示例程序使用机型：xMateER7 Pro
    // 1. 从当前位置 MoveJ 运动到拖拽位置
    std::array<double,7> q_drag_xm7p = {0, M_PI/6, 0, M_PI/3, 0, M_PI/2, 0};
    robot.updateRobotState(std::chrono::milliseconds(1));
    rtCon->MoveJ(0.4,  robot.jointPos(ec), q_drag_xm7p);

    // 2. 圆弧运动 (X-Y 平面上)
    CartesianPosition start, aux, target;
    robot.updateRobotState(std::chrono::milliseconds(1));
    Utils::postureToTransArray(robot.posture(rokae::CoordinateType::endInRef, ec),
start.pos);
    Eigen::Matrix3d rot_start;
    Eigen::Vector3d trans_start, trans_aux, trans_end;
    Utils::arrayToTransMatrix(start.pos, rot_start, trans_start);
    trans_end = trans_start; trans_aux = trans_start;
    trans_aux[0] -= 0.28;
    trans_aux[1] -= 0.05;
    trans_end[1] -= 0.15;

    Utils::transMatrixToArray(rot_start, trans_aux, aux.pos);
    Utils::transMatrixToArray(rot_start, trans_end, target.pos);
    rtCon->MoveC(0.2, start, aux, target);

    // 3. 直线运动
    Utils::postureToTransArray(robot.posture(rokae::CoordinateType::endInRef, ec),
start.pos);
    Utils::arrayToTransMatrix(start.pos, rot_start, trans_start);

    trans_end = trans_start;
    // 沿 x-0.1m, y-0.3m, z-0.25
    trans_end[0] -= 0.1;
    trans_end[1] -= 0.3;
    trans_end[2] -= 0.25;
    Utils::transMatrixToArray(rot_start, trans_end, target.pos);

    print(os, "MoveL start position:", start.pos, "Target:", target.pos);
    rtCon->MoveL(0.3, start, target);

    // 4. 关闭实时模式
    robot.setMotionControlMode(rokae::MotionControlMode::NrtCommand, ec);
    robot.setOperateMode(rokae::OperateMode::manual, ec);

  } catch (const std::exception &e) {
```

```
    std::cout << e.what();
  }
  return 0;
}
```

7 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。

8 附录 A – C++ API

8.1 枚举类型

8.1.1 机器人工作状态 rokae::OperationState

idle	机器人静止
jog	Jog 机器人中
rtControlling	实时模式控制中
drag	拖动已开启
rlProgram	RL 工程运行中
demo	Demo 演示中
dynamicIdentify	动力学辨识中
frictionIdentify	摩擦力辨识中
loadIdentify	负载辨识中
moving	机器人运动中
unknown	未知

8.1.2 机型类别 rokae::WorkType

industrial	工业机器人
collaborative	协作机器人

8.1.3 机器人操作模式 rokae::OperateMode

manual	手动
automatic	自动
unknown	未知(发生异常)

8.1.4 机器人上下电及急停状态 rokae::PowerState

on	上电
off	下电
estop	急停被按下
gstop	安全门打开
unknown	未知(发生异常)

8.1.5 位姿坐标系类型 rokae:: CoordinateType

flangeInBase	法兰相对于基坐标系
endInRef	末端相对于外部坐标系

8.1.6 运动控制模式 rokae::MotionControlMode

manual	手动
automatic	自动
unknown	未知(发生异常)

8.1.7 控制器实时控制模式 rokae::RtControllerMode

jointPosition	实时轴空间位置控制
cartesianPosition	实时笛卡尔空间位置控制
jointImpedance	实时轴空间阻抗控制
cartesianImpedance	实时笛卡尔空间阻抗控制
torque	实时力矩控制

8.1.8 机器人停止运动等级 rokae::StopLevel

stop0	快速停止机器人运动后断电
stop1	规划停止机器人运动后断电，停在原始路径上
stop2	规划停止机器人运动后不断电，停在原始路径上

8.1.9 机器人拖动模式参数 rokae::DragParameter

Space::jointSpace	轴空间拖动
Space::cartesianSpace	笛卡尔空间拖动
Type::translationOnly	仅平移
Type::rotationOnly	仅旋转
Type::freely	自由拖拽

8.1.10 Jog 选项 - 坐标系 rokae::JogOpt::Space

world	世界坐标系
flange	法兰坐标系
baseFrame	基坐标系
toolFrame	工具坐标系
wobjFrame	工件坐标系
jointSpace	轴空间

8.1.11 xPanel 配置: 对外供电模式 rokae::xPanelOpt::Vout

off	不输出
reserve	保留
supply12v	输出 12V
supply24v	输出 24V

8.1.12 力控任务坐标系 rokae::ForceControlFrameType

world	世界坐标系
tool	工具坐标系
path	路径坐标系，力控任务坐标系需要跟踪轨迹变化的过程

8.1.13 力矩类型 rokae::TorqueType

full	关节力矩，由动力学模型计算得到
inertia	惯性力
coriolis	科氏力
friction	摩擦力
gravity	重力

8.2 数据结构

8.2.1 机器人基本信息 rokae::Info

std::string id	机器人 uid，可用于区分连接的机器人
std::string version	控制器版本
std::string type	机器人机型名称
int joint_num	轴数

8.2.2 坐标系 rokae::Frame

std::array< double, 3 > trans	平移量, [X, Y, Z]，单位:米
std::array< double, 3 > rpy	XYZ 欧拉角, [A, B, C]，单位:弧度
std::array< double, 16 > pos	行优先变换矩阵

8.2.3 笛卡尔点位 rokae::CartesianPosition

std::array< double, 3 > trans	平移量, [x, y, z], 单位:米
std::array< double, 3 > rpy	旋转量, [r, p, y], 单位:弧度
double elbow	臂角, 适用于 7 轴机器人, 单位: 弧度
bool hasElbow	是否有臂角
std::vector< int > confData	轴配置数据，元素个数应和机器人轴数一致
std::vector< double > external	外部关节角度, 单位:弧度

8.2.4 笛卡尔点位偏移量 rokae::CartesianPosition::Offset

Type type	类型: Offs/RelTool
Frame frame	偏移坐标

8.2.5 关节点位 rokae::JointPosition

std::vector< double > joints	关节角度值, 单位:弧度
std::vector< double > external	外部关节角度值, 单位:弧度

8.2.6 关节扭矩, 不包含重力和摩擦力 rokae::Torque

std::vector< double > tau	期望关节扭矩，单位: Nm

8.2.7 负载信息 rokae::Load

double mass	负载质量, 单位:千克
std::array< double, 3 > cog	质心 [x, y, z], 单位:米
std::array< double, 3 > inertia	惯量 [ix, iy, iz, 单位:千克·平方米

8.2.8 工具工件组信息 rokae::Toolset

根据一对工具工件的坐标、负载、机器人手持设置计算得出。

Load load	机器人末端手持负载
Frame end	机器人末端坐标系相对法兰坐标系转换
Frame ref	机器人参考坐标系相对世界坐标系转换

8.2.9 RL 工程信息 rokae::RLProjectInfo

std::string name	工程名称
std::vector< std::string > taskList	任务名称列表

8.2.10 工具/工件信息 rokae::WorkToolInfo

std::string name	名称
std::string alias	别名, 暂未使用
bool robotHeld	是否机器人手持
Frame pos	位姿。工件的坐标系已相对其用户坐标系变换
Load load	负载

8.2.11 运动指令 MoveAbsJ rokae::MoveAbsJCommand

JointPosition target	目标关节点位
int speed	速率
int zone	转弯区大小

8.2.12 运动指令 MoveJ rokae::MoveJCommand

CartesianPosition target	目标笛卡尔点位
int speed	速率
int zone	转弯区大小

CartesianPosition::Offset offset	偏移量

8.2.13 运动指令 MoveL rokae::MoveLCommand

CartesianPosition target	目标笛卡尔点位
int speed	速率
int zone	转弯区大小
CartesianPosition ::Offset offset	偏移量

8.2.14 运动指令 MoveC rokae::MoveCCommand

CartesianPosition target	目标笛卡尔点位
CartesianPosition aux	辅助点位
int speed	速率
int zone	转弯区大小
CartesianPosition ::Offset targetOffset	目标点偏移量
CartesianPosition ::Offset auxOffset	辅助点偏移量

8.2.15 运动指令 MoveCF rokae::MoveCFCommand

CartesianPosition target	目标笛卡尔点位
CartesianPosition aux	辅助点位
int speed	速率
int zone	转弯区大小
double angle	执行角度
CartesianPosition ::Offset targetOffset	目标点偏移量
CartesianPosition ::Offset auxOffset	辅助点偏移量

8.2.16 控制器日志信息 rokae::LogInfo

const int	id	日志 ID 号
const std::string	timestamp	日期及时间
const std::string	content	日志内容
const std::string	repair	修复办法

8.3 方法

8.3.1 机器人基本操作及信息查询

connectToRobot()
void rokae::BaseRobot::connectToRobot (error_code &ec)
建立与机器人的连接。机器人地址为创建 Robot 实例时传入的
参数 [out] ec 错误码

disconnectFromRobot()
void rokae::BaseRobot::disconnectFromRobot (error_code & ec)
断开与机器人连接。断开前会停止机器人运动，请注意安全
参数 [out] ec 错误码

robotInfo()
Info rokae::BaseRobot::robotInfo (error_code &ec) const
查询机器人基本信息
参数 [out] ec 错误码
返回 机器人基本信息：控制器版本，机型，轴数

powerState()
PowerState rokae::BaseRobot::powerState (error_code & ec) const 机器人上下电以及急停状态
参数 [out] ec 错误码 **返回** on-上电 | off-下电 | estop-急停 | gstop-安全门打开

setPowerState()
void rokae::BaseRobot::setPowerState (bool on, error_code & ec) 机器人上下电。注: 只有无外接使能开关或示教器的机器人才能手动模式上电。
参数 [in] ontrue-上电 | false-下电 [out] ec 错误码

operateMode()
OperateMode rokae::BaseRobot::operateMode(error_code & ec) const 查询机器人当前操作模式
参数 [out] ec 错误码 **返回** 手动 | 自动

setOperateMode()
void rokae::BaseRobot::setOperateMode (OperateMode mode, error_code & ec) 切换手自动模式
参数 [in] mode 手动/自动 [out] ec 错误码

operationState()
OperationState rokae::BaseRobot::operationState (error_code & ec) const 查询机器人当前运行状态 (空闲,运动中, 拖动开启等)
参数 [out] ec 错误码 **返回** 运行状态枚举类

posture()
std::array< double, 6 > rokae::BaseRobot::posture(CoordinateType ct, error_code & ec) 获取机器人法兰或末端的当前位姿
参数 [in] ct 坐标系类型 1) flangeInBase: 法兰相对于基坐标系; 2) endInRef: 末端相对于外部参考坐标系。例如,当设置了手持工具及外部工件后，该坐标系类型返回的是工具相对于工件坐标系的坐标。 [out] ec 错误码 **返回** double 数组, [X, Y, Z, Rx, Ry, Rz]，其中平移量单位为米旋转量单位为弧度

cartPosture ()
CartesianPosition rokae::BaseRobot::cartPosture(CoordinateType ct, error_code & ec) 获取机器人法兰或末端的当前位姿
参数 [in] ct 坐标系类型 [out] ec 错误码

返回	当前笛卡尔位置

jointPos()
template<WorkType Wt, unsigned short DoF> std::array< double, DoF > rokae::Robot_T< Wt, DoF >::jointPos (error_code & ec) 机器人当前轴角度，单位：弧度
参数 [out] ec 错误码 **返回** 轴角度值

jointVel()
template<WorkType Wt, unsigned short DoF> std::array< double, DoF > rokae::Robot_T< Wt, DoF >::jointVel (error_code & ec) 机器人当前关节速度，单位：弧度/秒
参数 [out] ec 错误码 **返回** 关节速度

jointTorque()
template<WorkType Wt, unsigned short DoF> std::array< double, DoF > rokae::Robot_T< Wt, DoF >::jointTorque (error_code &ec) 关节力传感器数值，单位: Nm
参数 [out] ec 错误码 **返回** 力矩值

baseFrame()
std::array< double, 6 > rokae::BaseRobot::baseFrame (error_code &ec) const 用户定义的基坐标系，相对于世界坐标系
参数 [out] ec 错误码 **返回** double 数组, [X, Y, Z, Rx, Ry, Rz]，其中平移量单位为米旋转量单位为弧度

toolset()
Toolset rokae:: BaseRobot::toolset (std::error_code & ec) const 查询当前工具工件组信息
注解 此工具工件组仅为 SDK 运动控制使用，不与 RL 工程相关. **参数** [out] ec 错误码 **返回** 见 Toolset 数据结构

setToolset()
void rokae:: BaseRobot::setToolset (const Toolset & toolset, error_code & ec) 设置工具工件组信息
注解 此工具工件组仅为 SDK 运动控制使用，不与 RL 工程相关. 除此接口外，如果通过 RobotAssist 更改默认工具工件(右上角的选项)，该工具工件组也会相应更改. **参数** [in] toolset 工具工件组信息 [out] ec 错误码

setToolset()
void rokae:: BaseRobot::setToolset (const std::string &toolName, const std::string &wobjName,

error_code &ec)
使用已创建的工具和工件，设置工具工件组信息
注解 设置前提：已加载一个 RL 工程，且创建了工具和工件。否则，只能设置为默认的工具工件，即"tool0"和"wobj0"。一组工具工件无法同时为手持或外部；如果有冲突，以工具的位置为准，例如工具工件同时为手持，不会返回错误，但是工件的坐标系变成了外部
参数 [in] toolName 工具名称 [in] wobjName 工件名称 [out] ec 错误码

calcIk()
template<unsigned short DoF> JointArray rokae::Model_T< DoF >::calcIk(CartesianPosition posture, error_code &ec)
根据位姿计算逆解
参数 [in] posture 机器人末端位姿，相对于外部参考坐标系 [out] ec 错误码
返回 轴角度，单位:弧度

calcFk()
template<unsigned short DoF> CartesianPosition rokae::Model_T< DoF >::calcFk (const JointArray & joints, error_code & ec)
根据轴角度计算正解
参数 [in] joints轴角度，单位: 弧度 [out] ec 错误码
返回 机器人末端位姿，相对于外部参考坐标系

clearServoAlarm()
void rokae::BaseRobot::clearServoAlarm (error_code & ec)
清除伺服报警
参数 [out] ec 错误码，当有伺服报警且清除失败的情况下错误码置为-1

enableCollisionDetection()
template<unsigned short DoF> void Cobot<DoF>::enableCollisionDetection(const std::array<double, DoF> sensitivity, StopLevel behaviour, double fallback, error_code &ec)
设置碰撞检测相关参数，打开碰撞检测功能
参数 [in] sensitivity 碰撞检测灵敏度，范围 0.01-2.0 [in] behaviour 碰撞后机器人行为，支持 stop1(安全停止，stop0 和 stop1 处理方式相同)和 stop2(触发暂停) [in] fallback 碰撞后回退距离，单位: 米 [out] ec 错误码

disableCollisionDetection ()
template<unsigned short DoF>

void Cobot<DoF>::disableCollisionDetection(error_code &ec) 关闭碰撞检测功能	
参数	[out] ec 错误码

queryControllerLog()	
std::vector< LogInfo > rokae::BaseRobot::queryControllerLog (unsigned count, const std::set< LogInfo::Level > & level, error_code & ec) 查询控制器最新的日志	
参数	[in] count 查询个数，上限是 10 条 [in] level 指定日志等级，空集合代表不指定 [out] ec 错误码
返回	日志信息

sdkVersion()	
static std::string rokae::BaseRobot::sdkVersion ( ) 查询 xCore-SDK 版本	
返回	版本号

8.3.2 运动控制（非实时模式）

setMotionControlMode()	
void rokae::BaseRobot::setMotionControlMode (MotionControlMode mode, error_code & ec) 设置运动控制模式	
注解	在调用各运动控制接口之前，必须设置对应的控制模式。
参数	[in] mode 模式 [out] ec 错误码

moveReset()	
void rokae::BaseRobot::moveReset (error_code &ec) 重置运动缓存，清空已发送的运动指令，清除执行信息	
注解	清空已发送的运动指令，每次程序开始运行并第一次执行运动指令之前，需调用该函数来重置运动缓存，否则控制器可能会报错
参数	[out] ec 错误码

stop()	
void rokae::BaseRobot::stop (error_code & ec) 暂停机器人运动；暂停后可调用 moveStart()继续运动。若需要完全停止，不再执行已添加的指令，可调用 moveReset()	
注解	目前支持 stop2 停止类型，规划停止不断电，参见 StopLevel。 调用此接口后，暂停后可调用 moveStart()继续运动。若需要完全停止，不再执行已添加的指令，可调用 moveReset()
参数	[out] ec 错误码

moveStart()
void rokae::BaseRobot::moveStart(error_code &ec)
开始/继续运动

注解	目前支持 stop2 停止类型，规划停止不断电，参见 StopLevel。 调用此接口后，暂停后可调用 moveStart()继续运动。若需要完全停止，不再执行已添加的指令，可调用 moveReset()
参数	[out] ec 错误码

moveAppend() [1/2]
template<class Command > void rokae::BaseRobot::moveAppend (const std::vector<Command> &cmds, std::string &cmdID, error_code &ec) 添加单条或多条运动指令，添加后调用 moveStart()开始运动
模板参数 Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand | MoveCFCommand **参数** [in] cmds 指令列表，允许的个数为 1-100，须为同类型的指令 [out] cmdID 本条指令的 ID，可用于查询指令执行信息 [out] ec 错误码，仅反馈指令发送前的错误，包括: 1) 网络连接问题; 2) 指令个数不符;

moveAppend() [2/2]
template<class Command > void rokae::BaseRobot::moveAppend (std::initializer_list< Command > cmds, std::string &cmdID, error_code &ec) 添加单条或多条运动指令，添加后调用 moveStart()开始运动
模板参数 Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand | MoveCFCommand **参数** [in] cmds 指令列表，允许的个数为 1-100，须为同类型的指令 [out] cmdID 本条指令的 ID，可用于查询指令执行信息 [out] ec 错误码，仅反馈指令发送前的错误，包括: 1) 网络连接问题; 2) 指令个数不符;

executeCommand() [1/2]
template<class Command > void rokae::BaseRobot::executeCommand (std::initializer_list< Command > cmds, error_code & ec) 执行单条或多条运动指令，调用后机器人立刻开始运动
注解 须为同类型的指令; 运动指令异步执行，如发生执行错误，需通过 lastErrorCode()获取错误码。
模板参数 Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand **参数** [in] cmds指令列表，允许的个数为 1-1000 [out] ec 错误码，仅反馈执行前的错误，包括: 1) 网络连接问题; 2) 指令个数不符; 3) 机器人当前状态下无法运动，例如没有上电

executeCommand() [2/2]
template<class Command > void rokae::BaseRobot::executeCommand (std::vector< Command > cmds, error_code & ec) 执行单条或多条运动指令，调用后机器人立刻开始运动
注解 须为同类型的指令; 运动指令异步执行，如发生执行错误，需通过 lastErrorCode()获取错误码。 **模板参数** Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand **参数** [in] cmds指令列表，允许的个数为 1-1000 [out] ec 错误码，仅反馈执行前的错误，包括: 1) 网络连接问题; 2) 指令个数不符; 3) 机器人当前状态下无法运动，例如没有上电

setDefaultSpeed()	
void rokae::BaseRobot::setDefaultSpeed (int speed, error_code &ec)	
设定默认运动速度，初始值为 100	
注解	该数值表示末端最大线速度(单位 mm/s)，自动计算对应末端旋转速度及轴速度
参数	[in] speed 该接口不对参数进行范围限制。末端线速度的实际有效范围分别是 5-4000(协作)，5-7000(工业)。 关节速度百分比划分为 5 个的范围: < 100 : 10% ； 100 ~ 200 : 30% ； 200 ~ 500 : 50% ； 500 ~ 800 : 80% ； > 800 : 100% [out] ec 错误码

setDefaultZone()	
void rokae::BaseRobot::setDefaultZone (int zone, error_code & ec)	
设定默认转弯区	
注解	该数值表示运动最大转弯区半径(单位:mm)，自动计算转弯百分比. 若不设置，则为 0 (fine，无转弯区)
参数	[in] zone 该接口不对参数进行范围限制。转弯区半径大小实际有效范围是 0-200。 转弯百分比划分 4 个范围: < 1 : 0 (fine) ； 1 ~ 20 : 10% ； 20 ~ 60 : 30% ； > 60 : 100% [out] ec 错误码

setEventWatcher()	
void rokae::BaseRobot::setEventWatcher(Event eventType, const EventCallback &callback, error_code &ec)	
设置接收事件的回调函数	
参数	[in] eventType 事件类型，目前仅有非实时运动指令的执行信息 [in] callback 处理事件的回调函数 [out] ec 错误码

queryEventInfo()	
void rokae::BaseRobot::queryEventInfo(Event eventType, error_code &ec)	
设置接收事件的回调函数	
参数	[in] eventType 事件类型，目前仅有非实时运动指令的执行信息 [out] ec 错误码

startJog()	
void rokae::BaseRobot::startJog (JogOpt::Space space, double rate, double step, unsigned index, bool direction, error_code & ec)	
开始 jog 机器人，需要切换到手动操作模式。	
注解	调用此接口并且机器人开始运动后，无论机器人是否已经自行停止，都必须调用 stop()来结束 jog 操作，否则机器人会一直处于 jog 的运行状态。
参数	[in] space jog 参考坐标系。工具/工件坐标系使用的是 RobotAssist 右上角设定的工具/工件。 [in] rate 速率，范围 0.01 - 1 [in] step 步长。单位: 笛卡尔空间-毫米 | 轴空间-度。步长大于 0 即可，不设置上限， 如果机器人机器人无法继续 jog 会自行停止运动。

	[in] index笛卡尔空间 - 0~5 分别对应 XYZABC | 轴空间 - 关节序号，从 0 开始计数 [in] direction 方向，true - 正向 | false - 负向 [out] ec 错误码

[已弃用]~~lastErrorCode()~~
std::error_code rokae::BaseRobot::lastErrorCode() 获取最新的错误码，目前为运动指令的执行结果
返回 错误码,可调用 message()获取详细信息

8.3.3 实时运动控制

getRtMotionController()
template <unsigned short DoF> std::weak_ptr<RtMotionControlCobot<DoF>> Cobot<DoF>::getRtMotionController ( ) std::weak_ptr<RtMotionControlIndustrial<6>> StandardRobot::getRtMotionController() 创建实时运动控制类实例，通过此实例指针进行实时模式相关的操作。
注解 除非重复调用此接口，客户端内部逻辑不会主动析构返回的对象， 包括但不限于断开和机器人连接 disconnectFromRobot()，切换到非实时运动控制模式等，但做上述操作之后再进行实时模式控制会产生异常。 **返回** 控制器对象 **异常** RealtimeControlException 创建 RtMotionControl 实例失败，由于网络问题 ExecutionException 没有切换到实时运动控制模式

reconnectNetwork()
void rokae::MotionControl< MotionControlMode::RtCommand >::reconnectNetwork (error_code & ec) 重新连接到实时控制服务器
参数 [out] ec 错误码

disconnectNetwork()
void rokae::MotionControl< MotionControlMode::RtCommand >::disconnectNetwork ( ) 断开与实时控制服务器的连接，关闭数据接收和命令发送端口。但不会断开和机器人的连接。若机器人在运动，断开后立即停止运动。

setControlLoop()
template<class Command> void rokae::MotionControl< MotionControlMode::RtCommand >::setControlLoop (const std::function<Command(void)>& callback, int priority = 0, bool useStateDataInLoop = false) 使用周期调度，设置回调函数。
注解 1) 回调函数应按照 1 毫秒为周期规划运动命令，规划结果为函数的返回值。SDK 对返回值进行滤波处理后发送给控制器。 2) JointPosition 的关节角度数组长度，和 Torque 的关节力矩值数组长度应和机器人轴数相同。若不同不会报错，但有可能造成不合理的命令 3) 一次运动循环结束时，可以通过返回的 Command.setFinish()的方式来标识，SDK 内部会负责停止运动以及停止调用回调函数 **模板参数** JointPosition | CartesianPosition | Torque **参数** [in] callback 回调函数。根据控制模式(RtControllerMode)不同，函数返回值有 3 种: 关节角度/

	笛卡尔位姿/力矩。其中笛卡尔位姿使用旋转矩阵表示旋转量，pos 为末端相对于基坐标系的位姿。 [in] priority 任务优先级, 0 为不指定。此参数仅当使用实时操作系统时生效，若无法设置会打印控制台错误信息。 [in] useStateDataInLoop 是否需要在周期内读取状态数据。当设置为 true 时： 1) xCore-SDK 会在回调函数之前更新实时状态数据(updateStateData()),在回调函数内直接 getStateData()即可; 2) 状态数据的发送周期应和控制周期一致，为 1ms: startReceiveRobotState(interval = milliseconds(1));

startLoop()	
void rokae::MotionControl< MotionControlMode::RtCommand >::startLoop (bool blocking = true)	
开始周期性执行调度任务。	
参数	[in] blocking 是否阻塞调用此函数的线程。若为非阻塞线程，需调用 stopLoop()停止调度任务，否则无法开始下一次循环周期。
异常	RealtimeControlException 命令发送网络异常；或命令类型与控制模式不匹配；或控制器执行已发送命令时发生错误

stopLoop()	
void rokae::MotionControl< MotionControlMode::RtCommand >::stopLoop ( )	
停止执行周期性调度任务。	
异常	RealtimeControlException 执行过程中发生异常

startMove()	
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::startMove (RtControllerMode rtMode)	
指定控制模式，机器人准备开始运动，在每段回调执行前需要先调用此接口。 调用此接口机器人不会立即开始运动，而是有运动命令发送后才会开始。	
注解	1) 在 startMove 之前应将参数依次设置好，例如滤波阻抗参数等等，设置完成后再调用 startMove()。 在调用 startMove 后执行其他指令可能会失败，例如下电等操作。正确停止方法是调用 stopMove; 2) 如果没有通过 startReceiveRobotState()设置要接收的状态数据，调用此函数时会自动设置
参数	[in] rtMode 控制模式
异常	RealtimeStateException 已经开始运动运动后重复调用 RealtimeParameterException 指定了不支持的控制模式 RealtimeControlException 控制器无法切换到该控制模式，多出现于切换到力控模式时

stopMove()	
void rokae::MotionControl< MotionControlMode::RtCommand >::stopMove ( )	
机器人停止运动，停止接收客户端发送的运动指令。	
注解	另外，JointPosition/CartesianPosition/Torque 指令可通过 setFinished()标识一次运动循环结束，标识后会机器人停止运动， 呈现的效果和调用 stopMove()一样。 此函数仅用于实时控制，不可以用于停止非实时运动指令。

startReceiveRobotState()
template<WorkType Wt, unsigned short DoF> void rokae::Robot_T::startReceiveRobotState(std::chrono::steady_clock::duration interval, const std::vector<std::string>& fields)
让机器人开始发送实时状态数据。阻塞等待收到第一帧消息，超时时间为 3 秒

参数	[in] interval 控制器发送状态数据的间隔，允许的时长：1ms/2ms/4ms/8ms/1s
	[in] fields接收的机器人状态数据，最大总长度为 1024 个字节
异常	RealtimeControlException 设置了不支持的状态数据；或机器人无法开始发送数据；或总长度超过 1024
	RealtimeStateException 已经开始发送数据；或超时后仍未收到第一帧数据

stopReceiveRobotState()

void rokae::BaseRobot::stopReceiveRobotState ( )

停止接收实时状态数据，同时控制器停止发送。可用于重新设置要接收的状态数据。

updateRobotState()

unsigned rokae::BaseRobot::updateRobotState(std::chrono::steady_clock::duration timeout)

接收一次机器人状态数据。在每周期读取数据前，需调用此函数；建议按照设定的发送频率来调用，以获取最新的数据

参数	[in] timeout 超时时间
返回	接收到的数据长度。如果超时前没有收到数据，那么返回 0。
异常	RealtimeControlException 无法收到数据；或收到的数据有错误导致无法解析

getStateData()

template<typename R >
int rokae::BaseRobot::::getStateData(const std::string &fieldName,
R & data)

读取机器人状态数据

注解	注意传入的 data 类型要和数据类型一致。
模板参数	R 数据类型
参数	[in] fieldName 数据名
	[out] data 数值
返回	若无该数据名；或未通过 startReceiveRobotState()设置为要接收的数据；或该数据类型和 R 不符，返回-1。 成功读取返回 0。
异常	RealtimeStateException 网络错误

MoveJ()

template <WorkType Wt, unsigned short DoF>
void RtMotionControl<Wt, DoF>::MoveJ (double speed,
const std::array< double, DoF > & start,
const std::array< double, DoF > & target)

MoveJ 指令，上位机规划路径，在到达 target 之前处于处于阻塞状态。如果运动中发生错误将停止阻塞状态并返回。

参数	[in] speed 速度比例系数
	[in] start 起始关节角度，需要是机器人当前关节角度，否则可能造成下电。
	[in] target 机器人目标关节角度
异常	RealtimeMotionException 机器人运动过程中发生错误

MoveL()

template <WorkType Wt, unsigned short DoF>
void RtMotionControl<Wt, DoF>::MoveL (double speed,
CartesianPosition & start,

CartesianPosition & target) MoveL 指令，上位机规划路径，在到达 target 之前处于处于阻塞状态。如果运动中发生错误将停止阻塞状态并返回。
参数 [in] speed 速度比例系数，范围 0 - 1 [in] start 起始位姿，需要是机器人当前位姿，否则可能造成下电。如果设置了 TCP，那么应该是工具相对于基坐标系的位姿。 [in] target 机器人目标位姿。同理如果设置了 TCP，应是 TCP 相对于基坐标系的位姿 **异常** RealtimeParameterException 起始或目标位姿参数错误 RealtimeMotionException 机器人运动过程中发生错误

MoveC()
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::MoveC(double speed, CartesianPosition & start, CartesianPosition & aux, CartesianPosition & target) MoveC 指令，在到达 target 之前处于阻塞状态。如果运动中发生错误将停止阻塞状态并返回。
参数 [in] speed 速度比例系数 [in] start 机器人起始位姿，需要是机器人当前位姿。如果设置了 TCP，那么应该是工具相对于基坐标系的位姿。 [in] aux 机器人辅助点位姿。同理如果设置了 TCP，应是 TCP 相对于基坐标系的位姿 [in] target 机器人目标位姿。同理如果设置了 TCP，应是 TCP 相对于基坐标系的位姿 **异常** RealtimeParameterException 点位错误，无法计算出圆弧路径 RealtimeMotionException 机器人运动过程中发生错误

setFilterLimit()
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::setFilterLimit (bool limit_rate, double cutoff_frequency) 设置限幅滤波参数.
参数 [in] limit_rate true - 限幅开启 [in] cutoff_frequency 截止频率。范围是 0 ~ 1000Hz，建议 10~100Hz. **返回** true - 设定成功

setCartesianLimit()
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::setCartesianLimit (const std::array< double, 3 > & lengths, const std::array< double, 16 > & frame, error_code & ec) 设置笛卡尔空间运动区域，超过设置区域运动会停止。 非力控虚拟墙。若机器人末端或 TCP 末端超过安全区域，电机同样会做下电处理。
参数 [in] lengths 安全区域长方体长宽高，对应 XYZ，单位: 米 [in] frame 安全区域长方体中心相对于基坐标系位姿 [out] ec 错误码

setJointImpedance()
template <unsigned short DoF>

void RtMotionControlCobot<DoF>::setJointImpedance (const std::array< double, DoF > & factor, error_code & ec) 设置轴空间阻抗控制系数，轴空间阻抗运动时生效
参数 [in] factor 轴空间阻抗系数，单位：Nm/rad。xMateErPro 机型最大值为 { 1500, 1500, 1500, 1500, 100, 100, 100 }; 其他机型最大值为 { 1500, 1500, 1500, 100, 100, 100 } [out] ec 错误码

setCartesianImpedance()
template <unsigned short DoF> void RtMotionControlCobot<DoF>::setCartesianImpedance (const std::array< double, 6 > & factor, error_code & ec) 设置笛卡尔空间阻抗控制系数，笛卡尔阻抗运动时生效
参数 [in] factor 笛卡尔空间阻抗控制系数，最大值为 { 1500, 1500, 1500, 100, 100, 100 }，单位: N/m, Nm/rad [out] ec 错误码

setCollisionBehaviour()
template <unsigned short DoF> void RtMotionControlCobot<DoF>::setCollisionBehaviour (const std::array< double, DoF > & torqueThresholds, error_code & ec) 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 [in] torqueThresholds 关节碰撞检测阈值。 xMateErPro 机型最大值为 { 75, 75, 60, 45, 30, 30, 20 }， 其他机型最大值为{ 75, 75, 45, 30, 30, 20 } [out] ec 错误码

setEndEffectorFrame()
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::setEndEffectorFrame(const std::array< double, 16 > & frame, error_code & ec) 设置末端执行器相对于机器人法兰的位姿，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 [in] frame 末端执行器坐标系相对于法兰坐标系的齐次矩阵，单位: rad, m [out] ec 错误码

setLoad()
template <WorkType Wt, unsigned short DoF> void RtMotionControl<Wt, DoF>::setLoad (const Load & load, error_code & ec) 设置工具和负载的质量、质心和惯性矩阵。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 [in] load 负载信息 [out] ec 错误码

setFilterFrequency()

template <unsigned short DoF> void RtMotionControlCobot<DoF>::setFilterFrequency (double jointFrequency, double cartesianFrequency, double torqueFrequency, error_code & ec) 设置机器人控制器的滤波截止频率，用来平滑指令。允许的范围: 1 ~ 1000Hz，建议设置为 10 ~ 100Hz。
参数 [in] jointFrequency 关节位置的滤波截止频率，单位: Hz [in] cartesianFrequency 笛卡尔空间位置的滤波截止频率，单位: Hz [in] torqueFrequency 关节力矩的滤波截止频率，单位: Hz [out] ec 错误码

setCartesianImpedanceDesiredTorque()
template <unsigned short DoF> void RtMotionControlCobot<DoF>::setCartesianImpedanceDesiredTorque(const std::array< double, 6 > & torque, error_code & ec) 设置末端期望力，在笛卡尔空间阻抗运动时生效
参数 [in] torque 笛卡尔空间末端期望力，允许的范围为 { ±60, ±60, ±60, ±30, ±30, ±30 }，单位: N, N·m [out] ec 错误码

setTorqueFilterCutOffFrequency()
template <unsigned short DoF> void RtMotionControlCobot<DoF>::setTorqueFilterCutOffFrequency (double frequency, error_code & ec) 设置滤波参数
参数 [in] frequency 允许的范围 1 ~ 1000Hz [out] ec 错误码

setFcCoor()
template <unsigned short DoF> void RtMotionControlCobot<DoF>::setFcCoor (const std::array< double, 16 > & frame, ForceControlFrameType type, error_code & ec) 设置机器人力控坐标系
参数 [in] frame 力控坐标系相对于法兰坐标系的变换矩阵 [in] type 类别，指定哪个坐标系为力控任务坐标系 [out] ec 错误码

automaticErrorRecovery()
void rokae::MotionControl< MotionControlMode::RtCommand >::automaticErrorRecovery (error_code & ec) 当错误发生后，自动恢复机器人。
参数 [out] ec 错误码

setRtNetworkTolerance()

template<unsigned short DoF> void rokae::Cobot< DoF >::setRtNetworkTolerance (unsigned percent, error_code & ec) 设置发送实时运动指令网络延迟阈值，即 RobotAssist - RCI 设置界面中的”包丢失阈值“。 请在切换到 RtCommand 模式前进行设置，否则不生效。
参数 [in] percent 允许的范围 0 - 100 [out] ec 错误码

useRciClient()
template<unsigned short DoF> void rokae::Cobot< DoF >::useRciClient (bool use, error_code & ec) 兼容 RCI 客户端设置的接口。通过 SDK 设置运动控制模式为实时模式之后，无法再使用原 RCI 客户端控制机器人。 若有使用原版的需求，可在切换到非实时模式后，调用此接口。然后再在 RobotAssist 上打开 RCI 功能，即可使用 RCI 客户端。
参数 [in] use true - 切换到使用第一代 [out] ec 错误码

8.3.4 通信相关

getDI()
bool rokae::BaseRobot::getDI (unsigned int board, unsigned int port, error_code & ec) 查询数字量输入信号值
参数 [in] board IO 板序号 [in] port 信号端口号 [out] ec 错误码 **返回** true-开 | false-关

getDO()
bool rokae::BaseRobot::getDO(unsigned int board, unsigned int port, error_code &ec) 查询数字输出量信号值
参数 [in] board IO 板序号 [in] port 信号端口号 [out] ec 错误码 **返回** true-开 | false-关

setDI()
void rokae::BaseRobot::setDI (unsigned int board, unsigned int port, bool state, error_code &ec) 设置数字量输入信号，仅当输入仿真模式打开时可以设置(见 setSimulationMode())
参数 [in] board IO 板序号

[in] port 信号端口号 [in] state true-开 | false-关 [out] ec 错误码

setDO()
void rokae::BaseRobot::setDO (unsigned int board, unsigned int port, bool state, error_code & ec) 设置数字量输出信号值
参数 [in] board IO 板序号 [in] port 信号端口号 [in] state true-开 | false-关 [out] ec 错误码

getAI()
double rokae::BaseRobot::getAI (unsigned board, unsigned port, error_code & ec) 读取模拟量输入信号值
参数 [in] board IO 板序号 [in] port 信号端口号 [out] ec 错误码 **返回** 信号值

setAO()
void rokae::BaseRobot::setAO (unsigned board, unsigned port, double value, error_code & ec) 设置模拟量输出信号
参数 [in] board IO 板序号 [in] port 信号端口号 [in] value输出值 [out] ec 错误码

readRegister()
template<typename T > void rokae::BaseRobot::readRegister (const std::string & name, unsigned index, T & value, error_code & ec) 读取寄存器值。可读取单个寄存器，寄存器数组，或按索引读取寄存器数组。 如果要读取整个寄存器数组，value 传入对应类型的 vector，index 值被忽略。
模板参数 T 读取数值类型 **参数** [in] name 寄存器名称 [in] index按索引读取寄存器数组中元素，从 0 开始。 下列两种情况会报错：1）索引超出数组

	长度; 2) 寄存器不是数组但 index 大于 0 [out] value寄存器数值，允许的类型有 bool/int/float [out] ec 错误码

writeRegister()
template<typename T > void rokae::BaseRobot::writeRegister (const std::string & name, unsigned index, T value, error_code & ec) 写寄存器值。可写入单个寄存器，或按索引写入寄存器数组中某一元素。
模板参数 T 写入数值类型 **参数** [in] name 寄存器名称 [in] index数组索引，从 0 开始。 下列两种情况会报错：1) 索引超出数组长度; 2) 寄存器不是数组但 index 大于 0 [in] value写入的数值 [out] ec 错误码

setxPanelVout()
template<unsigned short DoF> void rokae::Cobot< DoF >::setxPanelVout(xPanelOpt::Vout opt, error_code & ec) 设置 xPanel 对外供电模式。注：仅部分机型支持 xPanel 功能，不支持的机型会返回错误码
参数 [in] opt 模式 [out] ec 错误码

setSimulationMode()
void rokae::BaseRobot::setSimulationMode(bool state, error_code &ec) 设置输入仿真模式
参数 [in] state true - 打开 | false - 关闭 [out] ec 错误码

8.3.5 RL 工程

projectsInfo()
std::vector< RLProjectInfo > rokae::BaseRobot::projectsInfo (error_code & ec) 查询工控机中 RL 工程名称及任务
参数 [out] ec 错误码 **返回** 工程信息列表，若没有创建工程则返回空列表

loadProject()
void rokae::BaseRobot::loadProject (const std::string & name, const std::vector< std::string > & tasks, error_code & ec) 加载工程
参数 [in] name 工程名称 [in] tasks 要运行的任务。该参数必须指定，不能为空，否则无法执行工程。

	[out] ec 错误码

ppToMain()
void rokae::BaseRobot:: ppToMain(error_code &ec)
程序指针跳转到 main。调用后，等待控制器解析完工程后返回，阻塞时间视工程大小而定，超时时间设定为 10 秒。
参数 [out] ec 错误码。错误码能提供的信息有限，不能反馈如 RL 语法错误、变量不存在等错误。可通过 queryControllerLog()查询错误日志。

runProject()
void rokae::BaseRobot::runProject (error_code &ec)
开始运行当前加载的工程
参数 [out] ec 错误码

pauseProject()
void rokae::BaseRobot::pauseProject (error_code &ec)
暂停运行工程
参数 [out] ec 错误码

setProjectRunningOpt()
void rokae::BaseRobot::setProjectRunningOpt (double rate, bool loop, error_code &ec)
更改工程的运行速度和循环模式
参数 [in] rate 运行速率，范围 0.01 - 1 [in] loop true - 循环执行 | false - 单次执行 [out] ec 错误码

toolsInfo()
std::vector< WorkToolInfo > rokae::BaseRobot::toolsInfo (std::error_code &ec)
查询当前加载工程的工具信息
参数 [out] ec 错误码
返回 工具信息列表，若未加载任何工程或没有创建工具，则返回默认工具 tool0 的信息

wobjsInfo()
std::vector< WorkToolInfo > rokae::BaseRobot::wobjsInfo (std::error_code &ec)
查询当前加载工程的工件信息
参数 [out] ec 错误码
返回 工件信息列表，若未加载任何工程或没有创建工件，则返回空 vector

8.3.6 协作相关

enableDrag()
template<unsigned short DoF> void rokae::Cobot< DoF >::enableDrag (DragParameter::Space space, DragParameter::Type type,

error_code & ec)
打开拖动
参数 [in] space 拖动空间. 轴空间拖动仅支持自由拖拽类型 [in] type 拖动类型 [out] ec 错误码

disableDrag()
template<unsigned short DoF> void rokae::Cobot< DoF >::disableDrag (error_code & ec)
关闭拖动
参数 [out] ec 错误码

startRecordPath()
template<unsigned short DoF> void rokae::Cobot< DoF >::startRecordPath (int duration, error_code & ec)
开始录制路径
参数 [in] duration 路径的时长，单位:秒，范围 1~1800.此时长只做范围检查用，到时后控制器不会停止录制，需要调用 stopRecordPath()来停止 [out] ec 错误码

stopRecordPath()
template<unsigned short DoF> void rokae::Cobot< DoF >::stopRecordPath (error_code & ec)
停止录制路径，若录制成功(无错误码)则路径数据保存在缓存中
参数 [out] ec 错误码

cancelRecordPath()
template<unsigned short DoF> void rokae::Cobot< DoF >::cancelRecordPath (error_code & ec)
取消录制，缓存的路径数据将被删除
参数 [out] ec 错误码

saveRecordPath()
template<unsigned short DoF> void rokae::Cobot< DoF >::saveRecordPath (const std::string & name, error_code & ec, const std::string & saveAs = "")
保存录制好的路径
参数 [in] name 路径名称 [out] ec 错误码 [in] saveAs 重命名，可选参数。 如果已录制好一条路径但没有保存，则用该名字保存路径。如果没有未保存的路径，则将已保存的名为"name"的路径重命名为"saveAs"

replayPath()
template<unsigned short DoF>

void rokae::Cobot< DoF >::replayPath (const std::string &name, double rate, error_code & ec) 运动指令-路径回放
参数 [in] name 要回放的路径名称 [in] rate 回放速率，应小于 3.0，1 为路径原始速率。注意当速率大于 1 时，可能产生驱动器无法跟随错误 [out] ec 错误码

removePath()
template<unsigned short DoF> void rokae::Cobot< DoF >::removePath (const std::string &name, error_code & ec, bool removeAll = false) 删除已保存的路径
参数 [in] name 要删除的路径名称 [out] ec 错误码。若路径不存在，错误码不会被置位 [in] removeAll 是否删除所有路径，可选参数，默认为否

queryPathLists()
template<unsigned short DoF> std::vector< std::string > rokae::Cobot< DoF >::queryPathLists (error_code &ec) 查询已保存的所有路径名称
参数 [out] ec 错误码 **返回** 名称列表，若没有路径则返回空列表

8.3.7 路径规划

CartMotionGenerator()
rokae::CartMotionGenerator::CartMotionGenerator (double speed_factor, double s_goal) 根据关节目标位置和速度系数生成一条轴空间轨迹，可用来回零或到达指定位置。
参数 [in] speed_factor 速度系数，范围[0, 1]. [in] s_goal 目标关节角度

calculateSynchronizedValues()
void rokae::CartMotionGenerator::calculateSynchronizedValues (double s_init) s 规划，笛卡尔空间是弧长 s 规划
参数 [in] s_init 初始弧长

calculateDesiredValues()
bool rokae::CartMotionGenerator::calculateDesiredValues (double t, double * delta_s_d) const 计算时间 t 时的弧长 s
参数 [in] t 距开始规划的时间，单位：秒 [out] delta_s_d 计算结果 **返回** false: 运动规划没有结束 | true: 运动规划结束

getTime()
double rokae::CartMotionGenerator::getTime ( ) 获得总运动时间
返回 运动时间，单位：秒

setMax()
void rokae::CartMotionGenerator::setMax (double ds_max, double dds_max_start, double dds_max_end) 设置笛卡尔空间运动参数
参数 [in] ds_max 最大速度 [in] dds_max_start 最大开始加速度 [in] dds_max_end最大结束加速度

JointMotionGenerator()
rokae::JointMotionGenerator::JointMotionGenerator(double speed_factor, std::array< double, 7 > q_goal) 根据关节目标位置和速度系数生成一条轴空间轨迹，可用来回零或到达指定位置。
参数 [in] speed_factor 速度系数，范围[0, 1] [in] q_goal 目标关节角度

calculateSynchronizedValues()
void rokae::JointMotionGenerator::calculateSynchronizedValues (const std::array< double, 7 > & q_init) s 规划
参数 [in] q_init初始关节角度

calculateDesiredValues()
bool rokae::JointMotionGenerator::calculateDesiredValues (double t, std::array< double, 7 > & delta_q_d) const 计算时间 t 时的关节角度增量
参数 [in] t 时间点 [out] delta_q_d 计算结果 **返回** false: 运动规划没有结束 | true: 运动规划结束

getTime()
double rokae::JointMotionGenerator::getTime ( ) 获得总运动时间
返回 运动时间，单位：秒

setMax()
void rokae::JointMotionGenerator::setMax (const std::array< double, 7 > & dq_max, const std::array< double, 7 > & ddq_max_start, const std::array< double, 7 > & ddq_max_end) 设置轴空间运动参数
参数 [in] dq_max 最大速度

[in] ddq_max_start 最大开始加速度 [in] ddq_max_end 最大结束加速度

FollowPosition()
template<unsigned short DoF> FollowPosition<DoF>::FollowPosition(Cobot<DoF>& robot, xMateModel<DoF>& model, const Eigen::Transform<double, 3, Eigen::Isometry>& endInFlange) 点位跟随功能，点位可以是笛卡尔位姿或轴角度。
参数 robot rokae::Robot 实例 model rokae::xMateModel 实例，通过 robot.model()获取 [in] endInFlange 末端相对于法兰的位姿

FollowPosition()
template<unsigned short DoF> FollowPosition<DoF>::FollowPosition() 点位跟随功能，点位可以是笛卡尔位姿或轴角度。默认构造函数，必须调用 init()来初始化

init()
template<unsigned short DoF> void FollowPosition<DoF>::init(Cobot<DoF>& robot, XMateModel<DoF>& model) 初始化 FollowPosition。
参数 robot rokae::Robot 实例 model rokae::xMateModel 实例，通过 robot.model()获取

start() [1/2]
template<unsigned short DoF> void FollowPosition<DoF>::start(const std::array<double, DoF> &jnt_desire) 开始目标跟随 - 笛卡尔位姿。该接口非阻塞。
参数 [in] bMe_desire 期望的目标位姿，为末端相对于基坐标系，即 TCP 位姿.

start() [2/2]
template<unsigned short DoF> void FollowPosition<DoF>::start(const Eigen::Transform<double, 3, Eigen::Isometry>& bMe_desire) 开始目标跟随 - 轴角度。该接口非阻塞
参数 [in] jnt_desire 期望的轴角度

stop()
template<unsigned short DoF> void FollowPosition<DoF>::stop() 停止目标跟随

update() [1/2]
template<unsigned short DoF> void FollowPosition<DoF>::update(const Eigen::Transform<double, 3, Eigen::Isometry>& bMe_desire); 更新期望的位姿
参数 [in] bMe_desire 末端相对于基坐标系，即 TCP 位姿

update() [2/2]
template<unsigned short DoF> void FollowPosition<DoF>::update(const std::array<double, DoF>& jnt_desired); 更新期望的目标轴角度
参数 [in] jnt_desired 轴角度，单位: 弧度

setScale()
template<unsigned short DoF> void FollowPosition<DoF>::setScale(double scale); 设置速度比例，可在目标跟随的过程中随时调整。
参数 [in] scale 速度比例，默认为 0.5

8.3.8 xMate 运动学与动力学计算模型库

model()
template<unsigned short DoF> XMateModel< DoF > rokae::Cobot< DoF >::model ( )
获取 xMate 模型类
返回 XMateModel
异常 ExecutionException 从控制器读取模型参数失败

getCartAcc()
template<unsigned short DoF> std::array< double, 6 > rokae::XMateModel< DoF >::getCartAcc (const std::array< double, DoF > & jntPos, const std::array< double, DoF > & jntVel, const std::array< double, DoF > & jntAcc, SegmentFrame nr = SegmentFrame::flange) 获取笛卡尔空间加速度
参数 [in] jntPos 需要计算笛卡尔空间速度的关节角度 [in] jntVel 需要计算笛卡尔空间速度的关节角速度 [in] jntAcc 需要计算笛卡尔空间速度的关节角加速度 [in] nr 指定坐标系 **返回** 计算结果

getCartPose()
template<unsigned short DoF> std::array< double, 16 > rokae::XMateModel< DoF >::getCartPose(const std::array< double, DoF > & jntPos, SegmentFrame nr = SegmentFrame::flange) 获取笛卡尔空间位置
参数 [in] jntPos 需要计算笛卡尔位姿的关节角度 [in] nr 指定坐标系，缺省值为 flange **返回** 向量化 4x4 位姿矩阵，行优先

getCartVel()

template<unsigned short DoF>
std::array< double, 6 > rokae::XMateModel< DoF >::getCartVel(const std::array< double, DoF > & jntPos,
const std::array< double, DoF > & jntVel,
SegmentFrame nr = SegmentFrame::flange)
获取笛卡尔空间速度

参数	[in] jntPos 需要计算笛卡尔空间速度的关节角度
	[in] jntVel 需要计算笛卡尔空间速度的关节角速度
	[in] nr 指定坐标系，缺省值为 flange
返回	计算结果

getJointAcc()

template<unsigned short DoF>
std::array< double, DoF > rokae::XMateModel< DoF >::getJointAcc (const std::array< double, 6 > & cartAcc,
const std::array< double, DoF > & jntPos,
const std::array< double, DoF > & jntVel)
逆解获得关节空间加速度

参数	[in] cartAcc 法兰笛卡尔空间加速度
	[in] jntPos 此时关节角度
	[in] jntVel 此时关节角速度
返回	计算结果

getJointPos()

template<unsigned short DoF>
int rokae::XMateModel< DoF >::getJointPos(const std::array< double, 16 > & cartPos,
double elbow,
const std::array< double, DoF > & jntInit,
std::array< double, DoF > & jntPos)
逆解获得关节空间位置。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。

参数	[in] cartPos 法兰笛卡尔空间位姿
	[in] elbow 臂角
	[in] jntInit 初始关节角度
	[out] jntPos 关节空间位置
返回	计算逆解结果 - 1) -1, -2, -3: 无解，原因是 cartPos 超出机器人工作空间; 2) -4, -5: jntPos 与 jntInit 相差较大，一般认为 jntInit 代表机器人当前位置，jntPos 与 jntInit 之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; 3) -6, -7: jntPos 超过软限位; 4) -8: 机器人奇异;

getJointVel()

template<unsigned short DoF>
std::array< double, DoF > rokae::XMateModel< DoF >::getJointVel (const std::array< double, 6 > & cartVel,
const std::array< double, DoF > & jntPos)
逆解获得关节空间速度

参数	[in] cartVel 法兰笛卡尔空间速度
	[in] jntPos 此时关节角度
返回	计算结果

getTorque()
template<unsigned short DoF> std::array< double, DoF > rokae::XMateModel< DoF >::getTorque (const std::array< double, DoF > & jntPos, const std::array< double, DoF > & jntVel, const std::array< double, DoF > & jntAcc, TorqueType torque_type) 由模型计算关节力矩
参数 [in] jntPos 关节角度 [in] jntVel 关节角速度 [in] jntAcc 关节角加速度 [in] torque_type 指定力矩类型 **返回** 计算结果，单位: Nm

getTorqueNoFriction()
template<unsigned short DoF> void rokae::XMateModel< DoF >::getTorqueNoFriction (const std::array< double, DoF > & jntPos, const std::array< double, DoF > & jntVel, const std::array< double, DoF > & jntAcc, std::array< double, DoF > & trq_full, std::array< double, DoF > & trq_inertia, std::array< double, DoF > & trq_coriolis, std::array< double, DoF > & trq_gravity) 由模型计算无摩擦力的关节力矩，计算结果单位: Nm。如有负载，先通过 setLoad()设置负载参数。
参数 [in] jntPos 关节角度 [in] jntVel 关节角速度 [in] jntAcc 关节角加速度 [out] trq_full 总关节力矩 [out] trq_inertia 离心力 [out] trq_coriolis科氏力 [out] trq_gravity重力矩

getTorqueWithFriction()
template<unsigned short DoF> void rokae::XMateModel< DoF >::getTorqueWithFriction (const std::array< double, DoF > & jntPos, const std::array< double, DoF > & jntVel, const std::array< double, DoF > & jntAcc, std::array< double, DoF > & trq_full, std::array< double, DoF > & trq_inertia, std::array< double, DoF > & trq_coriolis, std::array< double, DoF > & trq_friction, std::array< double, DoF > & trq_gravity) 由模型计算关节力矩
参数 [in] jntPos 关节角度 [in] jntVel 关节角速度 [in] jntAcc 关节角加速度

[out] trq_full 总关节力矩 [out] trq_inertia 离心力 [out] trq_coriolis科氏力 [out] trq_friction关节摩擦力 [out] trq_gravity重力矩

jacobian() [1/2]
template<unsigned short DoF> std::array< double, DoF *6 > rokae::XMateModel< DoF >::jacobian (const std::array< double, DoF > & jntPos, const std::array< double, 16 > & f_t_ee, const std::array< double, 16 > & ee_t_k, SegmentFrame nr = SegmentFrame::flange) 获取指定坐标系相对于基坐标系的雅克比矩阵，行优先
参数 [in] jntPos 关节角度. [in] f_t_ee 末端执行器相对于法兰坐标系的位姿. [in] ee_t_k 刚度坐标系相对于末端执行器的位姿. [in] nr 指定坐标系 **返回** 计算结果

jacobian() [2/2]
template<unsigned short DoF> std::array< double, DoF *6 > rokae::XMateModel< DoF >::jacobian (const std::array< double, DoF > & jntPos, SegmentFrame nr = SegmentFrame::flange) 获取指定坐标系相对于基坐标系的雅克比矩阵，行优先
参数 [in] jntPos 关节角度. [in] nr 指定坐标系 **返回** 计算结果

setLoad()
template<unsigned short DoF> void rokae::XMateModel< DoF >::setLoad (double mass, const std::array< double, 3 > & cog, const std::array< double, 3 > & inertia) 设置负载参数，只在计算时使用，并不将参数传给机器人控制器，设置后动力学计算结果相应改变
参数 [in] mass 质量 [in] cog 质心，单位: m [in] inertia 惯量 **返回** 参见 Load

setTcpCoor()
template<unsigned short DoF> void rokae::XMateModel< DoF >::setTcpCoor (const std::array< double, 16 > & f_t_ee, const std::array< double, 16 > & ee_t_k) 设置 TCP 工具，只在计算时使用，并不将参数传给机器人控制器，设置 TCP 后，正逆解结果和输入参数相应改变
参数 [in] f_t_ee 末端执行器相对于法兰的位姿

	[in] ee_t_k 刚度坐标系相对于末端执行器的位姿

8.3.9 其他

degToRad() [1/2]	
template<size_t S> static std::array< double, S > rokae::Utils::degToRad (const std::array< double, S > & degrees) 数组度转弧度	
参数	[in] degrees 度 弧度

degToRad() [2/2]	
static double rokae::Utils::degToRad(double degrees) 度转弧度	
参数	[in] degrees 度
返回	弧度

radToDeg() [1/2]	
template<size_t S> static std::array< double, S > rokae::Utils::radToDeg (const std::array< double, S > & rad) 数组弧度转度	
参数	[in] rad 弧度
返回	度

radToDeg() [2/2]	
static double rokae::Utils::radToDeg (double rad) 弧度转度	
参数	[in] rad 弧度
返回	度

arrayToTransMatrix()	
static void rokae::Utils::arrayToTransMatrix (const std::array< double, 16 > & array, Eigen::Matrix3d & rot, Eigen::Vector3d & trans) 数组转为变换矩阵	
参数	[in] array 数组，行优先 [out] rot 旋转矩阵 [out] trans 平移向量

transMatrixToArray()	
static void rokae::Utils::transMatrixToArray (const Eigen::Matrix3d & rot, const Eigen::Vector3d & trans, std::array< double, 16 > &array) 变换矩阵转为数组	
参数	[in] rot 旋转矩阵 [in] trans 平移向量 [out] array 转换结果，行优先

eulerToMatrix()
static void rokae::Utils::eulerToMatrix (const Eigen::Vector3d & euler, Eigen::Matrix3d & matrix) 欧拉角转为旋转矩阵
参数 [in] euler 欧拉角，顺序[z, y, x]，单位：弧度 [out] matrix 旋转矩阵

postureToTransArray()
static void rokae::Utils::postureToTransArray(const std::array<double, 6> &xyz_abc, std::array<double, 16> &transMatrix) 将表示位姿的数组{X, Y, Z, Rx, Ry, Rz}转换成行优先齐次变换矩阵
参数 [in] xyz_abc 输入位姿，{X, Y, Z, Rx, Ry, Rz} [out] transMatrix 转换结果

transArrayToPosture ()
static void rokae::Utils::transArrayToPosture(const std::array<double, 16> &transMatrix, std::array<double, 6> &xyz_abc) 将行优先齐次变换矩阵转换成{X, Y, Z, Rx, Ry, Rz}数组
参数 [in] transMatrix 行优先齐次变换矩阵 [out] xyz_abc 转换结果

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