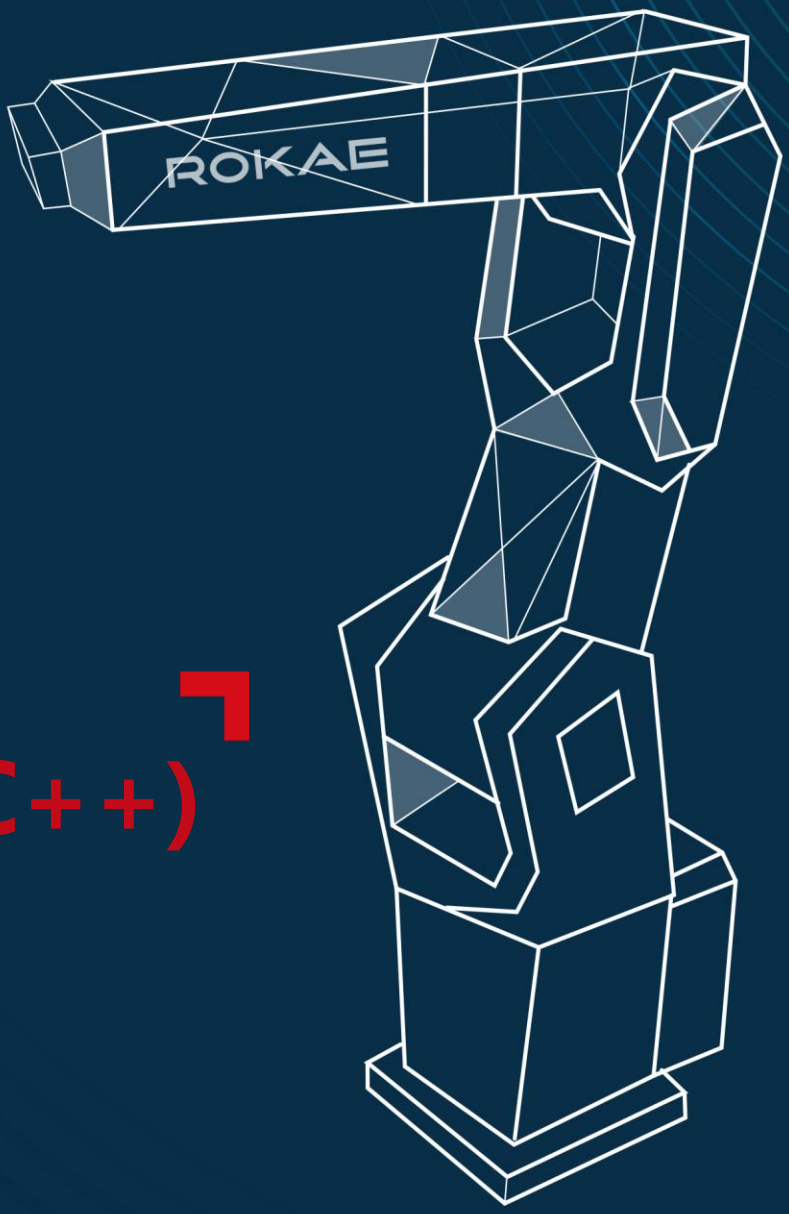
ROKAE 珞石

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：V2.2

文档版本：B

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;
V1.9	2023.10	新增接口，修复问题，适配 xCore 版本 v2.1
V2.0	2024.04	新增若干接口，修复问题，适配 xCore 版本 v2.2

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 v2.2 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 aarch64	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, MoveSP）
- 力控指令
- 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：Jog，设置碰撞检测，软限位，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、C#）请参考分手册。

3.1 设置

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 V2.2》。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。

3.2 编译

3.2.1 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
```

3.2.2 C++编译

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

3.2.2.1 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
```

3.2.2.2 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

3.2.2.3 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<path-to-library-directory> -lxCoreSDK

INCLUDEPATH += <path-to-include-directory>
DEPENDPATH += -L<path-to-include-directory>

#Eigen 配置
INCLUDEPATH += <path-to-external-directory>
```

静态库编译示例：

```
#Eigen
INCLUDEPATH += D:\xCoreSDK-v0.4.0\external
#SDK
LIBS += -LD:\xCoreSDK-v0.4.0\lib\Windows\cpp\Debug\64bit -lxCoreSDK
INCLUDEPATH += D:\xCoreSDK-v0.4.0\include
DEPENDPATH += D:\xCoreSDK-v0.4.0\include
```

3.3 运行日志

运行 SDK 程序并连接到机器人之后，SDK 会在调用设置数据、执行操作等接口时打印日志到文件，包含调用参数和返回结果等信息。可在遇到问题时将日志提供给珞石技术支持人员，以便分析排查。

目前暂不支持用户配置日志，默认的设置如下：

- 文件路径：<当前工作目录>/_rokae_log_/<机器人 uid>/xCoreSDK_<YYYY-MM-DD>.log
- 最多保留 7 天的日志

3.4 语言

xCore SDK 的错误码信息和异常信息支持中文和英文，根据用户 PC 设置的系统语言而定。设置显示中文则返回中文错误信息，非中文则返回英文错误信息

3.5 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。

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	全部支持	全部支持
	通信	全部支持	全部支持	全部支持
	RL 工程	全部支持	全部支持	全部支持
	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::ForceControl	力控指令	全部支持	部分支持	全部支持
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

4.1 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
xMateCr5Robot	协作 CR5 轴
PCB4Robot	工业 4 轴
PCB3Robot	工业 3 轴

4.2 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()		
	connectToRobot(remoteIP, localIP)	remoteIP - 机器人 IP 地址 localIP - 本机地址。实时模式下收发交互数据用	
断开连接	disconnectFromRobot()		
查询机器人基本信息	robotInfo()		控制器版本，机型，轴数
机器人上下电急停状态	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(ct)	ct – 坐标系类型	[X, Y, Z, Rx, Ry, Rz]
获取当前末端/法兰位姿	cartPosture(ct)	ct – 坐标系类型	[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]
设置基坐标系	setBaseFrame(frame)	frame – 坐标系	
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	控制器日志列表
设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/柔顺都	
关闭碰撞检测功能	disableCollisionDetection()		
坐标系标定	calibrateFrame(type, points, is_held, base_aux)	type – 坐标系类型 points – 标定轴角度列表 is_held – 手持/外部工具工件 base_aux – 基坐标系标定辅助点	标定结果: 坐标系和偏差
获取当前软限位数值	getSoftLimit(limits)	limits - 各轴软限位	已打开/已关闭
设置软限位	setSoftLimit(enable, limits)	enable – 打开/关闭 limits -各轴软限位	
查询 SDK 版本号	sdkVersion()		版本号

4.3 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()		
运动重置	moveReset()		
暂停机器人运动	stop()		
添加运动指令	moveAppend(command, id)	command - 一条或多条 MoveL/MoveJ/MoveAbsJ/ MoveC/MoveCF/MoveSP 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	
设置是否使用 conf	setDefaultConfOpt(forced)	forced – 是/否使用	
设置最大缓存指令个数	setMaxCacheSize(number)	number – 个数	
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 callback – 处理事件的回调函数	
查询事件信息	queryEventInfo(eventType)	eventType – 事件类型	事件信息
执行运动指令	executeCommand(command)	command - 一条或多条 MoveL/MoveJ/MoveAbsJ/ MoveC/MoveCF/MoveSP 指令	
读取当前加速度	getAcceleration(acc, jerk)	acc – 加速度 jerk – 加加速度	
设置运动加速度	adjustAcceleration(acc, jerk)	acc – 加速度 jerk – 加加速度	
打开奇异规避功能	setAvoidSingularity(method, enable, threshold)	method – 奇异规避方式 enable – 打开/关闭 threshold – 阈值参数	
查询是否打开规避奇异功能	getAvoidSingularity(method)	method – 奇异规避方式	已打开/已关闭

4.4 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

4.4.1 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。

4.4.2 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。

4.4.3 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。

4.4.4 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，目前支持的机型有：xMateER 系列所有机型, XMC7/12/18/20, XMS3/4；兼容性上支持 Linux x86_64、Windows 64bit。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。

4.4.4.1 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。

4.4.5 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。

4.4.6 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
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	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。
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；

4.5 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	
获取末端按键状态	getKeypadState()		末端按键的状态

4.6 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

4.7 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	stopRecordPath()		
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
力传感器标定	calibrateForceSensor(all_axes, axis_index)	all_axes – 标定所有轴 axis_index – 单轴标定下标	

4.8 力控指令

简述	接口	参数	返回值
获取当前力矩信息	getEndTorque(ref_type, joint, external, cart_torque, cart_force)	ref_type -力矩相对的参考系 joint - 各轴测量 external -各轴外部力 cart_torque - 笛卡尔空间力	

		矩 cart_force - 笛卡尔空间力	
力控初始化	fcInit(frame_type)	frame_type - 力控坐标系	
开始力控	fcStart()		
停止力控	fcStop()		
设置阻抗控制类型	setControlType(type)	type - 阻抗类型	
设置力控模块使用的负载	setLoad(load)	load - 负载	
设置关节阻抗刚度	setJointStiffness(stiffness)	stiffness - 刚度	
设置笛卡尔阻抗刚度	setCartesianStiffness(stiffness)	stiffness - 刚度	
设置笛卡尔零空间阻抗刚度	setCartesianNullspaceStiffness(stiffness)	stiffness - 刚度	
设置关节期望力矩	setJointDesiredTorque(torque)	torque - 力矩值	
设置笛卡尔期望力/力矩	setCartesianDesiredForce(value)	value -期望力/力矩	
设置绕单轴旋转的正弦搜索运动	setSineOverlay(line_dir, amplify, frequency, phase, bias)	line_dir - 参考轴 amplify -幅值 frequency - 频率 phase - 相位 bias - 偏置	
设置平面内的莉萨如搜索运动	setLissajousOverlay (int plane, double amplify_one, double frequency_one, double amplify_two, double frequency_two, double phase_diff, error_code &ec)	plane - 参考平面 amplify_one - 一方向幅值 frequency_one - 一方向频率 amplify_two - 二方向幅值 frequency_two - 二方向频率 phase_diff 相位偏差	
开启搜索运动	startOverlay()		
停止搜索运动	stopOverlay()		
暂停搜索运动	pauseOverlay()		
重新开启暂停的搜索运动	restartOverlay()		
设置与接触力有关的终止条件	setForceCondition (range, isInside, timeout)	range - 力限制 isInside - 超出/符合限制条件时停止等待 timeout - 超时时间	
设置与接触力矩有关的终止条件	setTorqueCondition (range, isInside, timeout)	range - 力矩限制 isInside - 超出/符合限制条件时停止等待 timeout - 超时时间	
设置与接触位置有关的终止条件	setPoseBoxCondition(supervising_frame, box, isInside, timeout)	supervising_frame - 长方体所在的参考坐标系 box - 长方体 isInside - 超出/符合限制条件时停止等待 timeout - 超时时间	
激活设置的终止条件并等待	waitCondition()		
启动/关闭力控模块保护监控	fcMonitor(enable)	enable - 打开|关闭	
设置力控模式下的轴最大速度	setJointMaxVel(velocity)	velocity – 轴速度	

设置机械臂末端相对基坐标系的最大速度	setCartesianMaxVel(velocity)	velocity – 末端速度	
设置力控模式下轴最大动量	setJointMaxMomentum(momentum)	momentum - 动量	
设置力控模式下轴最大动能	setJointMaxEnergy(energy)	energy - 动能	

4.9 错误码和异常

4.9.1 错误码

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 std::error_code 类型的参数，可通过 error_code::message()来获取错误码对应的信息。

数值	错误信息	原因及处理方法
0	操作成功完成	无
-1	发生错误	可能由于未识别的错误码，请反馈技术支持人员
-3	机器人急停按钮被按下, 请先恢复	恢复急停状态
-16	该操作不允许在当前模式下执行(手动/自动), 请切换到另一个模式	切换手动、自动模式
-17	该操作不允许在当前上下电状态下执行	切换上下电状态
-18	该操作不允许在机器人当前运行状态下执行	机器人非空闲，可能处于拖动/实时模式控制/辨识中等
-19	该操作不允许在当前控制模式(位置/力控)下执行	切换位置控制/力控
-20	机器人运动中	停止机器人运动
-32	计算逆解错误	传入正确位姿
-33	逆解是奇异点	规避奇异点
-34	逆解超出机器人软限位	检查软限位设置时候合适，传入限位内位姿
-35	目标点超出运动范围	传入可达点位
-36	目标点超出运动范围	传入可达点位
-37	单步距离过大，无法计算逆解	传入可达点位
-38	配置数据 cfx 错误逆解无解	检查 conf data
-39	输入数据错误，无法计算逆解	传入可达点位
-40	逆解参考点是奇异点	传入可达点位
-41	算法失效，无法计算逆解	可能由于控制器计算问题，请反馈技术支持人员
-257	通信协议解析失败	请反馈技术支持人员
-258	未找到匹配的数据名称	可能由于控制器和 SDK 版本不适配
-259	未找到匹配的指令名称	可能由于控制器和 SDK 版本不适配
-260	数据值错误, 可能是通信协议不匹配	可能由于控制器和 SDK 版本不适配
-272	未找到要查询的数据名称	可能由于控制器和 SDK 版本不适配
-273	数据不可读	可能由于控制器和 SDK 版本不适配
-288	数据不可写	可能由于控制器和 SDK 版本不适配
-289	设置数据失败	设置的数据不合理
-290	设置数据失败	设置的数据不合理
-291	监控数据失败	不会发生
-292	数据已被监控, 不支持重复监控	不会发生
-320	指令执行未完成	不会发生
-337	录制的路径过短, 数据点不足, 请重新录制	增长录制时间重新录制
-338	录制的路径中存在超过关节软限位的情况。请重新录制	在软限位内拖动
-339	录制的路径中存在关节速度过快的情况, 请重新录制	放慢拖动动作

-340	未从机器人静止状态开始录制路径	机器人静止时再开始录制
-341	停止录制路径时机器人未停止运动	机器人静止时再停止录制
-342	机器人处于下电状态, 保存路径失败	机器人上电
-352	缓冲区不存在有效的路径, 请录制	先录制路径再保存
-353	保存路径到磁盘失败	重新录制拖动轨迹，或重启机器人
-513	切换手动/自动操作模式失败	- 停止机器人运动 - 恢复急停 - 关闭拖动
-514	上下电失败	- 恢复急停 - 清除伺服报警
-515	打开/关闭拖动失败, 请检查机器人是否处于下电	机器人手动模式下电时打开拖动
-10005	标定中, 或标定时重置负载失败	- 等待标定完成 - 正确设置负载
-10030	未打开仿真模式或信号不存在	打开仿真模式后再设置 DI/AI
-10051	软限位超出机械限位	设置机械限位内的软限位
-28672	实时模式网络错误	- 本机 IP 地址不能为 localhost 或机器人地址 - 存在端口占用情况，请检查 RCI 设置-端口
-28688	切换运动控制模式失败	- 停止机器人运动 - 重启控制器
-28689	该频率不支持	状态数据发送频率支持 1kHZ, 500Hz, 250Hz, 125Hz
-28705	已经开始运动	停止运动后再开始
-28706	无法执行该操作，可能由于机器人未处于空闲状态	停止机器人运动
-28707	起始位置为奇异点	运动机器人到非奇异点
-28708	参数错误	重新设置数据
-28709	机器人处于碰撞停止	上电恢复碰撞状态
-28710	机器人处于急停状态	恢复急停
-28711	请求被拒绝	阻抗控制时多由于力控模型偏差，重新标定力矩传感器并设置正确的负载
-10079	力控模块处于错误状态	触发了力控保护，请检查力控模式下机器人状态是否正常，并设置合理的力控保护参数
-10141	负载质量超过了机器人额定负载	使用并设置额定负载范围内的负载
-14010	已绑定为系统 IO	- 取消绑定改信号为系统 IO - 使用其它信号
-14501	DO 信号不存在或为系统输出	检查 DO 信号是否已创建
-17001	读取寄存器错误	- 检查寄存器是否创建 - 是否超数组下标 - 用匹配的数据类型读取
-17002	写寄存器错误	- 检查寄存器是否创建 - 是否超数组下标 - 或用匹配的数据类型写入
-17320	执行过 RL 程序, 需要重置运动缓存后再开始运动	调用 moveReset 重置
-41400	尚有力控指令未完成	等待前序力控指令执行完毕
-41419	TCP 长度超限，力控初始化失败	末端工具长度限制为 0.3m
-41420	未进行力控初始化; 或未正确设置负载; 或力控模型偏差较大	- 标定零点、力矩传感器; - 设置正确的负载质量和质心; - 检查基坐标系是否正确设置; - 开启拖动时机器人没有收到外力
-41421	停止力控失败，当前不处于力控运行状态	机器人不处于力控状态
-41422	重新开启力控失败	先暂停或停止力控，再重新开启

-41425	非阻抗控制模式，或搜索运动在运行中	- 保证当前处于阻抗模式下，且搜索运动处于停止状态 - 请检查参数设置
-41426	非阻抗控制模式，或搜索运动在运行中	- 保证当前处于阻抗模式下，且搜索运动处于停止状态 - 请检查参数设置
-41427	当前不处于笛卡尔阻抗控制模式或未设置搜索运动	检查控制模式指令，设置搜索运动参数后再尝试
-41428	当前不处于笛卡尔阻抗控制模式或未开始搜索运动	请保证搜索运动已开启且处于笛卡尔阻抗运行状态
-41429	当前未处于笛卡尔阻抗模式或搜索运动未暂停	暂停当前的力控任务并检查控制模式，切换到笛卡尔阻抗控制模式后再尝试
-41430	不支持的阻抗控制类型或力控坐标系	停止当前的力控任务并重新参数初始化
-41431	当前未处于关节阻抗控制模式或未初始化	运行力控前设置轴空间阻抗控制模式
-41432	当前未处于笛卡尔阻抗控制模式或未初始化	运行力控前设置笛卡尔阻抗控制模式
-41433	当前未处于笛卡尔阻抗控制模式或未初始化	检查当前的阻抗控制模式后再进行尝试。确保力值输入正确，且设定笛卡尔阻抗控制模式
-41434	关节期望力超过限制	检查当前的阻抗控制模式后再进行尝试。确保期望力值输入正确，且设定轴空间阻抗控制模式
-41435	笛卡尔空间期望力超过限制	检查当前的阻抗控制模式后再进行尝试。确保期望力值输入正确，且设定笛卡尔阻抗控制模式
-41442	机器人不满足软限位要求，开启力控失败	请确认机器人开启阻抗时机器人在软限位内
-41444	阻抗刚度设置失败，数值不合理或当前状态不能设置刚度	请重新设置阻抗刚度数值在合理范围内
-41448	力控过程中下电，初始化失败	重新上电并执行力控初始化
-41449	力控指令尚未执行完毕，初始化失败	力控指令发送太频繁，请增加退出力控到重新开启之间的时间
-41457	执行过力控停止,需要重新开始	执行力控初始化
-41459	处于力控模式，不允许执行的操作	停止力控后再回放路径
-50000	两条轨迹夹角过小或长度过短，转弯区无法生成	- 增大前后两条轨迹夹角; - 增大前后两条轨迹长度; - 增大转弯区数值
-50001	控制器状态错误，无法生成轨迹	调用 moveReset()重置
-50002	目标点超出运动范围, 或为奇异点	- 检查目标点位置; - 用关节方式移动机器人 - 检查 confdata 配置
-50003	相邻两个目标点过近	检查相邻两条运动指令的目标点是否是同一个点
-50004	无法生成圆弧,起始点和目标点距离过近	调整圆弧起点或终点点位之间的距离
-50005	无法生成圆弧, 起始点和辅助点距离过近	调整圆弧起点或辅助点点位之间的距离
-50006	无法生成圆弧, 辅助点和目标点距离过近	调整圆弧终点或辅助点点位之间的距离
-50007	无法生成圆弧, 起始点/辅助点/目标点距离过近	调整圆弧起点、辅助点、终点点位间之间的距离
-50008	无法生成圆弧, 点位在一条直线	调整圆弧起点、辅助点、终点点位之间的距离
-50009	无法生成圆弧, 半径过小	调整圆弧起点、辅助点、终点点位之间的距离
-50010	无法生成圆弧	调整圆弧起点、辅助点、终点点位之间的距离或方位
-50019	生成轨迹失败	重新调整目标点的位置、姿态和臂角(仅 7 轴机器人需要考虑臂角)
-50021	指定 conf 参数下目标点无解，请检查数值或不设置 confData	- 调用 setDefaultConfOpt(false) 取消 confdata - 重新示教点位，传入正确的 confdata
-50027	轨迹短于最小转弯区半径, 自动拼接路径	- 该轨迹需要前后都衔接转弯区，但是轨迹长度小于最小转弯区半径的两倍; - 该轨迹设定衔接一条转弯区，但是轨迹长度小于最小转弯区半径。 该功能可以使得运动更平滑，如想关闭该功能，可将最

		小转弯区半径设为 0
-50033	机器人处于锁轴状态，目标点锁轴角度发生了偏离，请调整目标点或关闭锁轴状态	调整轨迹目标点或者关闭锁轴指令，锁轴状态下四轴角度要求为 0 度或 180 度或-180 度
-50034	锁轴状态下或 5 轴机型无法到达目标点	6 轴机型在锁轴状态下或 5 轴机型无法到达目标点,请更改点位或关闭锁轴
-50101	轴角度超出运动范围，尝试取消软限位后恢复各轴到允许的范围内	- 取消软限位 - 手动将机器人各轴移动到正常的工作范围内
-50102	存在穿越奇异点的轨迹	- 请规避奇异点 - 重新示教点位,更换目标点; - 将笛卡尔空间运动指令改为关节空间运动指令
-50103	笛卡尔路径终点不符合给定的 ConfData	- 调用 setDefaultConfOpt(false)不使用 confdata - 改为 MoveJ 或 MoveAbsJ - 更改目标点 confdata
-50104	关节力矩超限	- 检查负载数值是否符合实际负载情况 - 检查机器人摩擦力系数、电机过载系数、传动过载系数等参数 - 尝试更换指令形式，例如将笛卡尔空间指令更换为关节空间指令
-50112	位姿有误计算逆解失败	- 重新示教位姿 - 如果当前机型为三轴或者四轴机器人,请检查输入位姿与当前机型特征是否相符 - 如果使用了 setAvoidSingularity(lock4axis)，请检查目标是是否符合锁轴要求
-50113	改变的姿态超过设定的阈值	- 请规避奇异点 - 重新设置奇异规避姿态改变的阈值 - 重新示教点位,更换目标点 - 使用其他方式的奇异规避
-50114	前瞻中轴运动超出运动范围，提前停止运动	- 取消软限位 - 手动将机器人各轴移动到正常的工作范围内
-50115	轨迹前瞻过程中遇到奇异点	- 请规避奇异点 - 重新示教点位,更换目标点 - 更换 Jog 方式，尝试轴空间 Jog - 将笛卡尔空间运动指令改为关节空间运动指令
-50118	改变姿态后求解的终点与所需要终点不一致	- 请规避奇异点 - 重新示教点位,更换目标点 - 更换奇异规避方式
-50120	奇异规避下搜索路径终点角度失败	- 请规避奇异点 - 重新示教点位,更换目标点 - 更换奇异规避方式 - 如果 Conf 开启，可以关闭 Conf
-50121	奇异规避下搜索路径终点角度失败	- 请规避奇异点 - 重新示教点位,更换目标点 - 更换奇异规避方式
-50204	规划过程中采样点不足, 请重新运行	工控机状态不稳定，重新运行或尝试重启
-50205	相邻位置指令相差过大	- 重新示教轨迹目标点位 - 尝试更改指令形式，例如 MoveL 改为 MoveJ
-50208	内部轨迹错误	- 请调用 moveReset()重置运动指令缓存 - 增大或减小转弯区
-50401	检测到碰撞	- 请检查机器人运行环境，确认人员、设备安全后，重

		新上电，再恢复运行 - 检查当前工具设置是否与实际一致，设定的工具质量质心是否合理
-50501	工具工件坐标系设置失败，可能由于工具或工件不存在	设置已创建的工具工件，且分别为手持和外部
-50513	速度设置无效，机器人无法运动	传入 0.01~1 范围内的速度参数
-50514	步长设置无效，机器人无法运动	传入大于 0 的步长参数
-50515	参考坐标系设置无效，机器人无法运动	传入支持的坐标系类型参数
-50516	运动轴设置无效，机器人无法运动	按照说明传入 index 参数
-50518	运动失败，目标点可能是当前点	目标点位可能是当前点
-50519	生成轨迹失败, 目标点位可能超出机器人工作范围	请在机器人正常工作范围内，重新示教点位
-50525	该机型不支持奇异规避模式运动, 或机器人锁轴失败，4 轴角度不为 0 不运行运动, 请调整 4 轴角度	调整四轴角度至 0 或者正负 180 度
-60005	RL 工程或指定任务不存在	运行已创建的工程和任务
-60014	控制器当前状态不允许开始运动	确保机器人上电并处于空闲中
-60200	负载信息错误, 设置失败	- 负载信息错误,请检查负载重量是否超过额定负载，质心不超过 0.3m - 力控不支持外部工具或手持工件
-60511	路径不存在	使用已保存的回放路径
-60611	正在生成诊断数据不能运动,请等待	等待 10 秒后再次尝试启动
-60702	当前四轴角度不为 0，不允许切换到四轴固定模式，或机型不支持	请检查第四轴当前角度是否为 0°或 180°
-60704	不满足牺牲姿态奇异规避的开启条件	请检查当前状态是否满足牺牲姿态奇异规避开启状态
-60706	不满足轴空间插补奇异规避的开启条件	请检查当前状态是否满足轴空间插补奇异规避开启状态
256	网络连接错误	- 工控机未开机 - 机器人地址错误，或未处于同一局域网 - 实例类型错误或 SDK 未授权，连接被拒绝
257	无法解析机器人消息，可能由于 SDK 版本与控制器版本不匹配	可能由于控制器和 SDK 版本不适配
258	参数错误,数值超出范围	按照函数说明检查参数数值范围
259	参数错误,参数类型或个数错误	按照函数说明检查参数类型或数组长度
260	不是合法的变换矩阵	检查传入参数是否符合其次变换矩阵要求
261	数组元素个数与机器人轴数不符	传入和机器人轴数一致的数组长度
262	运动控制模式错误,请切换到正确的模式	根据实际情况调用 setMotionControlMode 切换模式
263	超时前未收到机器人回复, 可能由于网络通信问题	检查网络连接状态，或反馈技术支持人员
264	重复操作	碰撞检测已打开，需要先关闭
265	通过 UDP 端口接收数据失败，请检查网络及防火墙配置	- 检查本机地址设置是否正确 - 检查防火墙设置，是否允许 UDP 连接
266	客户端校验失败，请检查控制器版本，授权状态和机器人型号	- 按照手册或 README 将控制器升级到匹配版本 - 创建类型匹配的机器人实例 - 联系技术支持人员授权 SDK 功能
272	事件未监听	先调用 setEventWatcher 开始监听数据
273	点位距离过近，坐标系标定失败	参考《xCore 控制系统使用手册》工具工件标定方法，重新标定
512	IP 地址或端口设置错误, 或端口被占用	- 本机 IP 地址不能为 localhost 或机器人地址 - 存在端口占用情况，请检查 RCI 设置-端口
513	设置了不支持的字段, 或总长度超出限制	支持的字段见 RtSupportedFields，总长度限制为 1024 字节
768	没有可执行的运动指令	先调用 moveAppend 下发运动指令，再开始运动

4.9.2 异常

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。

异常信息	原因及处理方法
运动控制模式错误, 非实时模式	- 调用 setMotionControlMode(RtCommand)切换模式 - 前一次阻抗控制发生错误，控制器切回到非实时模式，需重新打开
创建实时控制客户端失败	网络连接错误，检查工控机运行状态
已开始接收数据	先停止接收状态数据，再重新开始接收
控制器无法设置发送的状态数据	可能由于版本不匹配，请反馈技术支持人员
设置控制指令失败	可能由于版本不匹配，请反馈技术支持人员
UDP socket 打开失败	端口占用，或绑定本机地址失败
控制器无法发送机器人状态信息.	控制器 UDP 端口设置出错，请反馈技术支持人员查看控制器日志
未开始接收数据	调用 startReceiveRobotState 开始接收数据
超时前未收到机器人状态反馈,请检查网络配置	- 检查本机地址设置是否正确 - 检查防火墙设置，是否允许 UDP 连接
接收机器人状态数据错误	状态数据解析失败，可能由于版本不匹配，请反馈技术支持人员
无法收到运动错误反馈	数据解析失败，可能由于版本不匹配，请反馈技术支持人员
xCore 最低版本要求	按照提示信息升级控制器版本
未授权 SDK 功能	联系技术支持人员授权 SDK 功能
机器人实例类型与连接的机型不符	按照手册说明创建对应的机器人实例
本机 IPv4 地址设置失败，不能为 127.0.0.1 或机器人地址	设置正确的本机地址
位姿初始化接收 6 或 16 个参数	初始化 trans&rpy 应传入长度为 6 的数组；初始化 pos（实时模式笛卡尔指令）应传入长度为 16 的数组
加载模型失败	除非网络连接问题，一般不会出现
xMate 模型库暂不支持的机型	请参考 xMateModel 类注释，查看支持的机型
已经开始运动, 请勿重复调用	同时只允许一个运动回调运行，调用 stopMove 之后再开始下一段运动·
未调用开始运动	先调用 startMove，再调用 startLoop
设置模式错误, 开始运动失败	除非网络连接问题，一般不会出现
开始运动失败	- 确保机器人空闲，未处于急停等非正常使用的状态 - 多出现于阻抗控制，请标定零点、力矩传感器，并正确设置负载后再次尝试 - 或将完整异常信息反馈技术支持人员。
停止运动失败	一般是网络连接问题
工业机型仅支持位置控制	工业机型使用位置控制
起点和终点的臂角格式不一样	笛卡尔指令 hasElbow 值不一致
指令和控制模式不一致	回调函数返回的数据类型 (JointPosition/ CartesianPosition/ Torque) 应和控制类型匹配
发送指令失败	指令数据序列化失败，请反馈技术支持人员
圆弧起点/终点/辅助点重合	调整圆弧起点、辅助点、终点点位之间的距离
圆弧路径求解时，三点共线	调整圆弧起点、辅助点、终点
圆弧路径求解时，三点夹角过小	调整圆弧起点、辅助点、终点

圆弧路径求解时，圆弧半径过小失败	调整圆弧起点、辅助点、终点点位之间的距离
未初始化	FollowPosition 默认构造需要调用 init()函数之后在使用
滤波错误,输入数据非正数	检查指令数据，回调函数是否返回了未赋值的指令
滤波错误, 需滤波数据无限大	检查指令数据，回调函数是否返回了未赋值的指令

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； ● 非实时模式运动的暂停； ● 工具工件组: 更改 RobotAssist 右上角选择的工具工件会改变 toolset；通过 setToolset()设置了工具工件，右上角会显示”toolx”, “wobjx”
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； ● 大部分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。

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 v2.2（v1.7 及之后的版本也同样可以）之后直接使用 xCoreSDK。

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 入站规则是否是允许的状态。

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：机器人基本操作，信息查询，Jog，拖动等

```
/**
 * @brief 示例 - 基础的信息查询，计算正逆解
 */
template <WorkType wt, unsigned short dof>
void example_basicOperation(Robot_T<wt, dof> *robot){
  error_code ec;
  // *** 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());

  // *** 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); // 轴角度 [rad]
  auto joint_vel = robot->jointVel(ec); // 轴速度 [rad/s]
  auto joint_torque = robot->jointTorque(ec); // 轴力矩 [Nm]
  auto tcp_xyzabc = robot->posture(CoordinateType::endInRef, ec);
  auto flan_cart = robot->cartPosture(CoordinateType::flangeInBase, ec);
  robot->baseFrame(ec); // 基坐标系
  print(os, "末端相对外部参考坐标系位姿", tcp_xyzabc);
  print(os, "法兰相对基坐标系 -", flan_cart);

  // *** 计算逆解 & 正解 ***
  auto model = robot->model();
  auto ik = model.calcIk(tcp_xyzabc, ec);
  auto fk_ret = model.calcFk(ik, ec);
}

/**
 * @brief 示例 - 打开关闭拖动
 */
void example_drag(BaseCobot *robot) {
  error code ec;
  robot->setOperateMode(rokae::OperateMode::manual, ec);
  robot->setPowerState(false, ec); // 打开拖动之前，需要机械臂处于手动模式下电状态
  // 笛卡尔空间，自由拖动
  robot->enableDrag(DragParameter::cartesianSpace, DragParameter::freely, ec);
  print(os, "打开拖动", ec, "按回车继续");
  std::this_thread::sleep_for(std::chrono::seconds(2)); //等待切换控制模式
  while(getchar() != '\n');
  robot->disableDrag(ec);
  std::this_thread::sleep_for(std::chrono::seconds(2)); //等待切换控制模式
}

/**
 * @brief 示例 - 读写 IO，寄存器
 */
void example_io_register(BaseRobot *robot) {
  error_code ec;
  print(os, "DO1_0 当前信号值为:", robot->getDO(1,0,ec));
  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);
}

/**
 * @brief 示例 - Jog 机器人
 * @param robot
 */
void example_jog(BaseRobot *robot) {
  error_code ec;
  robot->setMotionControlMode(rokae::MotionControlMode::NrtCommand, ec);
  robot->setOperateMode(rokae::OperateMode::manual, ec); // 手动模式下 jog
  print(os, "准备 Jog 机器人，需手动模式上电，请确认已上电后按回车键");
  // 对于有外接使能开关的情况，需要按住开关手动上电
  robot->setPowerState(true, ec);

  print(os, "-- 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");
  robot->startJog(JogOpt::world, 0.5, 50, 2, true, ec);
  while(getchar() != '\n');
  print(os, "轴空间，6 轴负向连续转动，速率 5%，按回车停止 Jog");
  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
}

/**
 * @brief 示例 - 打开和关闭碰撞检测
 */
template <unsigned short dof>
void example_setCollisionDetection(Cobot<dof> *robot) {
  error_code ec;
  // 设置各轴灵敏度，范围 0.01 ~ 2.0，相当于 RobotAssist 上设置的 1% ~ 200%
  // 触发行为：安全停止；回退距离 0.01m
  robot->enableCollisionDetection({1.0, 1.0, 0.01, 2.0, 1.0, 1.0, 1.0}, StopLevel::stop1,
0.01, ec);
  std::this_thread::sleep_for(std::chrono::seconds(2));
  // 关闭碰撞检测
  robot->disableCollisionDetection(ec);
}
```

6.1.2 示例二：非实时运动指令

```
/**
 * @brief 示例 - 设置轴配置数据(confData)处理多逆解问题，点位适用机型 xMateCR7
 */
void multiplePosture(xMateRobot &robot) {
  error_code ec;
  std::string id;

  // 本段示例使用默认工具工件
  Toolset defaultToolset;
  robot.setToolset(defaultToolset, ec);

  MoveJCommand moveJ({0.2434, -0.314, 0.591, 1.5456, 0.314, 2.173});
  // 同样的末端位姿，confData 不同，轴角度也不同
  moveJ.target.confData =  {-67, 100, 88, -79, 90, -120, 0, 0};
  robot.moveAppend({moveJ}, id, ec);
  moveJ.target.confData =  {-76, 8, -133, -106, 103, 108, 0, 0};
  robot.moveAppend({moveJ}, id, ec);
  moveJ.target.confData =  {-70, 8, -88, 90, -105, -25, 0, 0};
  robot.moveAppend({moveJ}, id, ec);
  robot.moveStart(ec);
  waitForFinish(robot, id, 0);
}
```

```
/**
 * @brief 示例 - 七轴冗余运动 & 发生碰撞检测后恢复运动，点位适用机型 xMateER3 Pro
 */
void redundantMove(xMateErProRobot &robot) {
  error_code ec;
  std::string id;

  // 本段示例使用默认工具工件，速度 v500，转弯区 fine
  Toolset defaultToolset;
  robot.setToolset(defaultToolset, ec);
  robot.setDefaultSpeed(500, ec);
  robot.setDefaultZone(0, ec);

  // 可选：设置碰撞检测事件回调函数
  robot.setEventWatcher(Event::safety, [&](const EventInfo &info){
    recoverFromCollision(robot, info);
  }, ec);


  MoveAbsJCommand moveAbsj({0, M_PI/6, 0, M_PI/3, 0, M_PI_2, 0});
  // ** 1) 变臂角运动 **
  MoveLCommand moveL1({0.562, 0, 0.432, M_PI, 0, -M_PI});
  moveL1.target.elbow = 1.45;
  robot.moveAppend({moveAbsj}, id, ec);
  robot.moveAppend({moveL1}, id, ec);
  moveL1.target.elbow = -1.51;
  robot.moveAppend({moveL1}, id, ec);
  robot.moveStart(ec);
  // 最后一次 moveAppend()发送一条指令，故 index = 0
  waitForFinish(robot, id, 0);

  // ** 2) 60°臂角圆弧 **
  CartesianPosition circle_p1({0.472, 0, 0.342, M_PI, 0, -M_PI}),
  circle_p2({0.602, 0, 0.342, M_PI, 0, -M_PI}),
  circle_a1({0.537, 0.065, 0.342, M_PI, 0, -M_PI}),
  circle_a2({0.537, -0.065, 0.342, M_PI, 0, -M_PI});
  // 臂角都是 60°
  circle_p1.elbow = M_PI/3;
  circle_p2.elbow = M_PI/3;
  circle_a1.elbow = M_PI/3;
  circle_a2.elbow = M_PI/3;

  MoveLCommand moveL2(circle_p1);
  robot.moveAppend({moveL2}, id, ec);
  MoveCCommand moveC1(circle_p2, circle_a1), moveC2(circle_p1, circle_a2);
  std::vector<MoveCCommand> movec_cmds = {moveC1, moveC2};
  robot.moveAppend(movec_cmds, id, ec);
  robot.moveStart(ec);
  // 最后一次 moveAppend()发送 2 条指令，故需要等待第二个点完成后返回，index 为第二个点的下标
  waitForFinish(robot, id, (int)movec_cmds.size() - 1);
}
```

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 状态(未运动)
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
jogging	Jog 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

Idle	空闲
NrtCommand	非实时模式执行运动指令
NrtRLTask	非实时模式运行 RL 工程
RtCommand	实时模式控制

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
suppleStop	柔顺停止，仅适用于协作机型

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 坐标系类型 rokae::FrameType

world	世界坐标系
base	基坐标系
flange	法兰坐标系
tool	工具坐标系
wobj	工件坐标系
path	路径坐标系, 力控任务坐标系需要跟踪轨迹变化的过程

8.1.11 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
singularityAvoidMode	奇异规避模式，适用于工业六轴，xMateCR 和 xMateSR 六轴机型
baseParallelMode	平行基座模式，适用于工业六轴，xMateCR 和 xMateSR 六轴机型

8.1.12 奇异规避方式 rokae::AvoidSingularityMethod

lockAxis4	四轴锁定
wrist	牺牲姿态
jointWay	轴空间短轨迹插补

8.1.13 MoveCF 全圆姿态旋转类型 rokae::MoveCFCommand::RotType

constPose	不变姿态
rotAxis	动轴旋转
fixedAxis	定轴旋转

8.1.14 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

8.1.15 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

8.1.16 事件类型 rokae::Event

moveExecution	非实时运动指令执行信息
safety	安全 (是否碰撞)

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	XYZ 欧拉角, [A, B, C], 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 坐标系标定结果 rokae::FrameCalibrationResult

Frame frame	标定结果
std::array<double, 3> errors	样本点与 TCP 标定值的偏差，依次为最小值,平均值,最大值，单位 m

8.2.10 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

8.2.11 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

8.2.12 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

8.2.13 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

8.2.14 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

8.2.15 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

8.2.16 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	全圆执行角度, 单位: 弧度
CartesianPosition ::Offset targetOffset	目标点偏移量
CartesianPosition ::Offset auxOffset	辅助点偏移量
RotType rotType	全圆姿态旋转模式

8.2.17 运动指令 MoveSP rokae:: MoveSPCommand

CartesianPosition target	终点笛卡尔点位
CartesianPosition::Offset targetOffset	偏移选项
double radius	初始半径, 单位: 米
double radius_step	每旋转单位角度，半径的变化，单位: 米/弧度
double angle	合计旋转角度, 单位: 弧度
bool direction	旋转方向, true - 顺时针 | false - 逆时针

8.2.18 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

8.2.19 末端按键状态 rokae::KeyPadState

bool key1_state	CR1 号
bool key2_state	CR2 号
bool key3_state	CR3 号
bool key4_state	CR4 号
bool key5_state	CR5 号
bool key6_state	CR6 号
bool key7_state	CR7 号

8.3 方法

8.3.1 机器人基本操作及信息查询

connectToRobot() [1/2]	
template<WorkType Wt, unsigned short DoF>	
void rokae::Robot_T< Wt, DoF >::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

connectToRobot() [2/2]	
template<WorkType Wt, unsigned short DoF>	
void rokae::Robot_T< Wt, DoF >::connectToRobot (const std::string &remoteIP,	
const std::string &localIP = "")	
连接到机器人	
参数	remoteIP 机器人 IP 地址
	localIP 本机地址。实时模式下收发交互数据用，可不设置；PCB3/4 轴机型不支持

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] on true-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

setBaseFrame()
void rokae::BaseRobot::setBaseFrame(const Frame &frame, error_code &ec)
设置基坐标系, 设置后仅保存数值，重启控制器后生效
参数 [in] frame 坐标系，默认使用自定义安装方式 [out] ec 错误码

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。设置后 RobotAssist 右上角会显示“toolx", "wobjx", 状态监控显示的末端坐标也会变化。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

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

calibrateFrame ()

```
template<WorkType Wt, unsigned short DoF>
FrameCalibrationResult rokae::Robot_T< Wt, DoF >::calibrateFrame ( FrameType type,
  const std::vector< std::array< double, DoF > > &points,
  bool is_held,
  error_code &ec,
  const std::array< double, 3 > & base_aux = {} )
```

坐标系标定 (N 点标定)

注解	各坐标系类型支持的标定方法及注意事项： 1) 工具坐标系: 三点/四点/六点标定法 2) 工件坐标系: 三点标定。标定结果不会相对用户坐标系做变换，即，若为外部工件，返回的结果是相对于基坐标系的。 3) 基坐标系: 六点标定。标定前请确保动力学约束和前馈已关闭。 若标定成功(无错误码)，控制器会自动保存标定结果，重启控制器后生效。
参数	[in] points 轴角度列表，列表长度为 N。例如，使用三点法标定工具坐标系，应传入 3 组轴角度。轴角度的单位是弧度。 [in] is_held true - 机器人手持 | false - 外部。仅影响工具/工件的标定 [out] ec 错误码 [in] base_aux 基坐标系标定时用到的辅助点, 单位[米]
返回	标定结果，当错误码没有被置位时，标定结果有效

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_compliance,
                                          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），suppleStop(柔顺停止) [in] fallback_compliance 1) 碰撞后行为是安全停止或触发暂停时，该参数含义是碰撞后回退距离，单位: 米 2) 碰撞后行为是柔顺停止时，该参数含义是柔顺度，范围 [0.0, 1.0] [out] ec 错误码

disableCollisionDetection()

```
void BaseCobot::disableCollisionDetection(error_code &ec)
```

关闭碰撞检测功能

参数	[out] ec 错误码

getSoftLimit()	
template<WorkType Wt, unsigned short DoF> bool rokae::Robot_T< Wt, DoF >::getSoftLimit (std::array< double[2], DoF > &limits, error_code &ec) 获取当前软限位数值	
参数	[out] limits各轴软限位 [下限, 上限]，单位: 弧度 [out] ec 错误码
返回	true - 已打开 | false - 已关闭

setSoftLimit()	
template<WorkType Wt, unsigned short DoF> void rokae::Robot_T< Wt, DoF >::setSoftLimit (bool enable, error_code &ec, const std::array< double[2], DoF > & limits = {{DBL_MAX, DBL_MAX}}) 设置软限位。软限位设定要求： 1) 打开软限位时，机械臂应下电且处于手动模式; 2) 软限位不能超过机械硬限位 3) 机械臂当前各轴角度应在设定的限位范围内	
参数	[in] enable true - 打开 | false - 关闭。 [out] ec 错误码 [in] limits各轴[下限, 上限]，单位：弧度。 1) 当 limits 为默认值时，视为仅打开软限位不修改数值; 不为默认值时，先修改软限位再打开 2) 关闭软限位时不会修改限位数值

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, 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） 运动重置, 清空已发送的运动指令, 清除执行信息	
注解	Robot 类在初始化时会调用一次运动重置。RL 程序和 SDK 运动指令切换控制，需要先运动重置。
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	
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 | MoveSPCommand
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 | MoveSPCommand
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	
模板参数	Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand | MoveCFCommand | MoveSPCommand;

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	
模板参数	Command 运动指令类: MoveJCommand | MoveAbsJCommand | MoveLCommand | MoveCCommand | MoveCFCommand | MoveSPCommand;
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), 自动计算对应关节速度
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

setDefaultConfOpt()	
void rokae::BaseRobot::setDefaultConfOpt(bool forced, error_code &ec) 设置是否使用轴配置数据(confData)计算逆解。初始值为 false	
参数	[in] forced true -使用运动指令的 confData 计算笛卡尔点位逆解, 如计算失败则返回错误; false - 不使用，逆解时会选取机械臂当前轴角度的最近解 [out] ec 错误码

setMaxCacheSize()	
void rokae::BaseRobot::setMaxCacheSize(int number, error_code &ec) 设置最大缓存指令个数，指发送到控制器待规划的路径点个数，允许的范围[1,300]，初始值为 30。	
注解	如果轨迹多为短轨迹，可以调大这个数值，避免因指令发送不及时导致机器人停止运动(停止后如果有未执行的指令，可 moveStart()继续;
参数	[in] number 个数 [out] ec 错误码

adjustSpeedOnline()	
void rokae::BaseRobot::adjustSpeedOnline(double scale, error_code &ec) 动态调整机器人运动速率，非实时模式时生效。	
参数	[in] scale 运动指令的速度的比例，范围 0.01 - 1。当设置 scale 为 1 时，机器人将以路径原本速度运动。 [out] ec 错误码

getAcceleration()	
void rokae::BaseRobot::getAcceleration(double &acc, double &jerk, error_code &ec) 读取当前加/减速度和加加速度	
参数	[out] acc 系统预设加速度的百分比 [out] jerk 系统预设的加加速度的百分比 [out] ec 错误码

adjustAcceleration()	
void rokae::BaseRobot::adjustAcceleration(double acc, double jerk, error_code &ec) 调节运动加/减速度和加加速度。如果在机器人运动中调用，当前正在执行的指令不生效，下一条指令生效	
参数	[in] acc 系统预设加速度的百分比，范围[0.2, 1.5], 超出范围不会报错，自动改为上限或下限值 [in] jerk 系统预设的加加速度的百分比，范围[0.1, 2], 超出范围不会报错，自动改为上限或下限值 [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 [in] callback 处理事件的回调函数。说明: 1) 对于 Event::moveExecution, 回调函数在同一个线程执行, 请避免函数中有执行时间较长的操作; 2) Event::safety 则每次独立线程回调, 没有执行时间的限制 [out] ec 错误码

queryEventInfo()	
EventInfo rokae::BaseRobot::queryEventInfo(Event eventType, error_code &ec) 查询事件信息。与 setEventWatcher()回调时的提供的信息相同，区别是这个接口是主动查询的方式	
参数	[in] eventType 事件类型 [out] ec 错误码
返回	事件信息

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。 1) 工具/工件坐标系使用原则同 setToolset(); 2) 工业六轴机型和 xMateCR/SR 六轴机型支持两种奇异规避方式 Jog：Space::singularityAvoidMode, Space::baseParallelMode 3) CR5 轴机型支持平行基座模式 Jog：Space::baseParallelMode [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 根据不同的 space，该参数含义如下： 1) 世界坐标系,基坐标系,法兰坐标系,工具工件坐标系: 0~5 分别对应 X, Y, Z, Rx, Ry, Rz 2) 轴空间: 关节序号，从 0 开始计数 3) 奇异规避模式,平行基座模式: a) 6 轴机型：0~5 分别对应 X, Y, Z, J4(4 轴), Ry, J6(6 轴); b) 5 轴机型：0~4 分别对应 X, Y, Z, Ry, J5(5 轴) [in] direction 根据不同的 space 和 index，该参数含义如下： 1) 奇异规避模式 J4: true - ±180° | false - 0°; 2) 平行基座模式 J4 & Ry: true - ±180° | false - 0° 3) 其它，true - 正向 | false - 负向 [out] ec 错误码

setAvoidSingularity()
void rokae::xMateRobot::setAvoidSingularity (AvoidSingularityMethod method, bool enable, double limit, error_code &ec) void rokae::StandardRobot::setAvoidSingularity (AvoidSingularityMethod method, bool enable, double limit, error_code &ec) 打开/关闭奇异点规避功能。只适用于部分机型: 1) 四轴锁定: 支持工业六轴，xMateCR 和 xMateSR 六轴机型； 2) 牺牲姿态: 支持所有六轴机型； 3) 轴空间插补: 支持工业六轴机型
参数 [in] method 奇异规避方式 [in] enable true - 打开功能 | false - 关闭。对于四轴锁定方式, 打开之前要确保 4 轴处于零位。 [in] limit 不同的规避方式，该参数含义分别为: 1) 牺牲姿态: 允许的姿态误差, 范围 (0, PI*2], 单位弧度 2) 轴空间插补: 规避半径, 范围[0.005, 10], 单位米 3) 四轴锁定: 无参数 [out] ec 错误码

getAvoidSingularity()
bool rokae::xMateRobot::getAvoidSingularity (AvoidSingularityMethod method, error_code &ec) bool rokae::StandardRobot::getAvoidSingularity (AvoidSingularityMethod method, error_code &ec) 查询是否处于规避奇异点的状态
参数 [in] method 奇异规避的方式

	[out] ec 错误码
返回	true - 已打开 | false – 已关闭

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。支持的数据及名称见 data_types.h, RtSupportedFields
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 机型最大刚度为 { 3000, 3000, 3000, 3000, 300, 300, 300 } 其他机型最大刚度为 { 3000, 3000, 3000, 300, 300, 300 } 实际有效的最大值和传感器等硬件状态有关系，如发生抖动等现象，请尝试减小阻抗系数。 [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 { 3000, 3000, 3000, 300, 300, 300 }, 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 } 5 轴机型最大值为{ 75, 75, 45, 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,
FrameType 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, 支持:
1) 世界坐标系 FrameType::world;
2) 工具坐标系 FrameType::tool;
3) 路径坐标系 FrameType::path (力控任务坐标系需要跟踪轨迹变化的过程)
[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()
void rokae::BaseCobot::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()
void rokae::BaseCobot::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()
void rokae::BaseCobot::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

getKeypadState()
KeyPadState rokae::BaseRobot::getKeypadState(error_code& ec) 获取末端按键状态，不支持的机型会返回错误码
参数 [out] ec 错误码 **返回** 末端按键的状态。末端按键编号见《xCore 机器人控制系统使用手册》末端把手的图示。

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()
void rokae::BaseCobot::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()
void rokae:: BaseCobot::disableDrag (error_code & ec) 关闭拖动
参数 [out] ec 错误码

startRecordPath()
void rokae::BaseCobot::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()
void rokae::BaseCobot::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()
void rokae::BaseCobot::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()
void rokae::BaseCobot::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()
void rokae::BaseCobot::replayPath (const std::string &name, double rate,

error_code & ec)
运动指令-路径回放
和其它运动指令类似，调用 replayPath 之后，需调用 moveStart 才会开始运动。
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()
void rokae: BaseCobot::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()
std::vector< std::string > rokae::BaseCobot::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

calibrateForceSensor()
template<unsigned short DoF> void rokae::Cobot<DoF>::calibrateForceSensor(bool all_axes, int axis_index, error_code &ec) 打开拖动
参数 [in] all_axes true - 标定所有轴 | false - 单轴标定 [in] axis_index 轴下标, 范围[0, DoF), 仅当单轴标定时生效 [out] ec 错误码

8.3.7 力控指令

forceControl()
template<unsigned short DoF> ForceControl_T<DoF> rokae::Cobot<DoF>::forceControl ( ) 力控指令类
返回 ForceControl_T

getEndTorque()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::getEndTorque (FrameType ref_type, std::array< double, DoF > & joint_torque_measured, std::array< double, DoF > & external_torque_measured, std::array< double, 3 > & cart_torque, std::array< double, 3 > & cart_force,

error_code &ec)
获取当前力矩信息
参数 [in] ref_type力矩相对的参考系： 1) FrameType::world - 末端相对世界坐标系的力矩信息; 2) FrameType::flange - 末端相对于法兰盘的力矩信息; 3) FrameType::tool - 末端相对于 TCP 点的力矩信息 [out] joint_torque_measured 各轴测量力 [out] external_torque_measured 各轴外部力 [out] cart_torque 笛卡尔空间力矩 [X, Y, Z], 单位 Nm [out] cart_force 笛卡尔空间力 [X, Y, Z], 单位 N [out] ec 错误码

fcInit()
void rokae::BaseForceControl::fcInit(FrameType frame_type, error_code &ec) 力控初始化
参数 [in] frame_type 力控坐标系，支持 world/wobj/tool/base/flange。工具工件坐标系使用 setToolset()设置的坐标系 [out] ec 错误码

fcStart()
void rokae::BaseForceControl::fcStart(error_code &ec) 开始力控，fcInit()之后调用。 如需在力控模式下执行运动指令，fcStart()之后可执行。 注意，如果在 fcStart()之前通过 moveAppend()下发了运动指令但未开始运动，fcStart 之后就会执行这些运动指令。
参数 [out] ec 错误码

fcStop()
void rokae::BaseForceControl::fcStop(error_code &ec) 停止力控
参数 [out] ec 错误码

setControlType()
void rokae::BaseForceControl::setControlType(int type, error_code &ec) 设置阻抗控制类型
参数 [in] type 0 - 关节阻抗 | 1 - 笛卡尔阻抗 [out] ec 错误码

setLoad()
void rokae::BaseForceControl::setLoad(const Load &load, error_code &ec) 设置力控模块使用的负载信息，fcStart()之后可调用。
参数 [in] load 负载 [out] ec 错误码

setJointStiffness()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::setJointStiffness (const std::array< double, DoF > & stiffness, error_code &ec)

设置关节阻抗刚度。fcInit()之后调用生效 各机型的最大刚度不同，请参考《xCore 控制系统手册》 SetJntCtrlStiffVec 指令的说明
参数 [in] stiffness 各轴刚度 [out] ec 错误码

setCartesianStiffness()
void rokae::BaseForceControl::setCartesianStiffness(const std::array<double, 6> &stiffness, error_code &ec) 设置笛卡尔阻抗刚度。fcInit()之后调用生效 各机型的最大刚度不同，请参考《xCore 控制系统手册》 SetCartCtrlStiffVec 指令的说明
参数 [in] stiffness 依次为：X Y Z 方向阻抗力刚度[N/m], X Y Z 方向阻抗力矩刚度[Nm/rad] [out] ec 错误码

setCartesianNullspaceStiffness()
void rokae::BaseForceControl::setCartesianNullspaceStiffness(double stiffness, error_code &ec) 设置笛卡尔零空间阻抗刚度。fcInit()之后调用生效
参数 [in] stiffness 范围[0,4], 大于 4 会默认设置为 4, 单位 Nm/rad [out] ec 错误码

setJointDesiredTorque()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::setJointDesiredTorque(const std::array<double, DoF> &torque, error_code &ec) 设置关节期望力矩。fcStart()之后可调用
参数 [in] torque 力矩值, 范围[-30,30], 单位 Nm [out] ec 错误码

setCartesianDesiredForce()
void rokae::BaseForceControl::setCartesianDesiredForce(const std::array<double, 6> &value, error_code &ec) 设置笛卡尔期望力/力矩。fcStart()之后可调用
参数 [in] value 依次为: X Y Z 方向笛卡尔期望力, 范围[-60,60], 单位 N; X Y Z 方向笛卡尔期望力矩, 范围[-10,10], 单位 Nm [out] ec 错误码

setSineOverlay()
void rokae::BaseForceControl::setSineOverlay(int line_dir, double amplify, double frequency, double phase, double bias, error_code &ec) 设置绕单轴旋转的正弦搜索运动。 设置阻抗控制类型为笛卡尔阻抗(即 setControlType(1))之后, startOverlay()之前调用生效。 各机型的搜索运动幅值上限和搜索运动频率上限不同，请参考《xCore 控制系统手册》 SetSineOverlay 指令的说明。
参数 [in] line_dir 搜索运动参考轴: 0 - X | 1 - Y | 2 - Z [in] amplify 搜索运动幅值, 单位 Nm [in] frequency 搜索运动频率, 单位 Hz [in] phase 搜索运动相位, 范围[0, PI], 单位弧度 [in] bias 搜索运动偏置, 范围[0, 10], 单位 Nm

	[out] ec 错误码

setLissajousOverlay()	
void rokae::BaseForceControl::setLissajousOverlay(int plane, double amplify_one, double frequency_one, double amplify_two, double frequency_two, double phase_diff, error_code &ec) 设置平面内的莉萨如搜索运动 设置阻抗控制类型为笛卡尔阻抗(即 setControlType(1))之后, startOverlay()之前调用生效。	
参数	[in] plane 搜索运动参考平面: 0 - XY | 1 - XZ | 2 - YZ [in] amplify_one 搜索运动一方向幅值, 范围[0, 20], 单位 Nm [in] frequency_one 搜索运动一方向频率, 范围[0, 5], 单位 Hz [in] amplify_two 搜索运动二方向幅值, 范围[0, 20]单位 Nm [in] frequency_two 搜索运动二方向频率, 范围[0, 5], 单位 Hz [in] phase_diff 搜索运动两个方向相位偏差, 范围[0, PI], 单位弧度 [out] ec 错误码

startOverlay()	
void rokae::BaseForceControl::startOverlay(error_code &ec) 开启搜索运动。fcStart()之后调用生效 搜索运动为前序设置的 setSineOverlay()或 setLissajousOverlay()的叠加	
参数	[out] ec 错误码

stopOverlay()	
void rokae::BaseForceControl::stopOverlay(error_code &ec) 停止搜索运动	
参数	[out] ec 错误码

pauseOverlay()	
void rokae::BaseForceControl::pauseOverlay(error_code &ec) 暂停搜索运动。startOverlay()之后调用生效	
参数	[out] ec 错误码

restartOverlay()	
void rokae::BaseForceControl::restartOverlay(error_code &ec) 重新开启暂停的搜索运动。pauseOverlay()之后调用生效。	
参数	[out] ec 错误码

setForceCondition()	
void rokae::BaseForceControl::setForceCondition(const std::array<double, 6> &range, bool isInside, double timeout, error_code &ec) 设置与接触力有关的终止条件	
参数	[in] range 各方向上的力限制 { X_min, X_max, Y_min, Y_max, Z_min, Z_max }, 单位 N。 设置下限时, 负值表示负方向上的最大值; 设置上限时, 负值表示负方向上的最小值。 [in] isInside true - 超出限制条件时停止等待; false - 符合限制条件时停止等待 [in] timeout 超时时间, 范围[1, 600], 单位秒 [out] ec 错误码

setTorqueCondition()
void rokae::BaseForceControl::setTorqueCondition(const std::array<double, 6> &range, bool isInside, double timeout, error_code &ec) 设置与接触力矩有关的终止条件
参数 [in] range 各方向上的力矩限制 { X_min, X_max, Y_min, Y_max, Z_min, Z_max }, 单位 Nm。设置下限时, 负值表示负方向上的最大值; 设置上限时, 负值表示负方向上的最小值。 [in] isInside true - 超出限制条件时停止等待; false - 符合限制条件时停止等待 [in] timeout 超时时间, 范围[1, 600], 单位秒 [out] ec 错误码

setPoseBoxCondition()
void rokae::BaseForceControl::setPoseBoxCondition(const Frame &supervising_frame, const std::array<double, 6> &box, bool isInside, double timeout, error_code &ec) 设置与接触位置有关的终止条件
参数 [in] supervising_frame 长方体所在的参考坐标系, 相对于外部工件坐标系。外部工件坐标系是通过 setToolset()设置的 (Toolset::ref) [in] box 定义一个长方体 { X_start, X_end, Y_start, Y_end, Z_start, Z_end }, 单位米 [in] isInside true - 超出限制条件时停止等待; false - 符合限制条件时停止等待 [in] timeout 超时时间, 范围[1, 600], 单位秒 [out] ec 错误码

waitCondition()
void rokae::BaseForceControl::waitCondition(error_code &ec) 激活前序设置的终止条件并等待，直到满足这些条件或者超时
参数 [out] ec 错误码

fcMonitor()
void rokae::BaseForceControl::fcMonitor(bool enable, error_code &ec) 启动/关闭力控模块保护监控。 设置监控参数后，不立即生效，调用 fcMonitor(true)后开始生效，并且一直保持，直到调用 fcMotion(false)后结束。 结束后保护阈值恢复成默认值，即仍然会有保护效果，关闭监控后不再是用户设置的参数。
参数 [in] enable true - 打开 | false - 关闭 [out] ec 错误码 **参见** setCartesianMaxVel() setJointMaxVel() setJointMaxMomentum() setJointMaxEnergy()

setJointMaxVel()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::setJointMaxVel(const std::array<double, DoF> &velocity, error_code &ec) 设置力控模式下的轴最大速度
参数 [in] velocity 轴速度 [rad/s]，范围 >=0 [out] ec 错误码

setJointMaxMomentum()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::setJointMaxMomentum(

const std::array<double, DoF> &momentum, error_code &ec) 设置力控模式下轴最大动量。 计算方式：F*t，可以理解为冲量，F 为力矩传感器读数，t 为控制周期，如果超过 30 个周期都超过动量阈值则触发保护
参数 [in] momentum 动量 [N·s]，范围 >=0 [out] ec 错误码

setJointMaxEnergy()
template<unsigned short DoF> void rokae::ForceControl_T<DoF>::setJointMaxEnergy(const std::array<double, DoF> &energy, error_code &ec) 设置力控模式下轴最大动能。 计算方式：F*v，可以理解为功率，F 为力矩传感器读数，v 为关节速度, 如果超过 30 个周期都超过动能阈值则触发保护
参数 [in] energy 动能 [N·rad/s]，范围 >=0 [out] ec 错误码

setCartesianMaxVel()
void rokae::BaseForceControl::setCartesianMaxVel(const std::array<double, 6> &velocity, error_code &ec) 设置力控模式下, 机械臂末端相对基坐标系的最大速度
参数 [in] velocity 依次为：X Y Z [m/s], A B C [rad/s], 范围 >=0 [out] ec 错误码

8.3.8 路径规划

CartMotionGenerator()
rokae::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

8.3.9 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

8.3.10 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