ESTUN ROBOTICS

ER 系列工业机器人 ERI 操作手册

RCS2 V1.0

南京埃斯顿自动化股份有限公司
南京埃斯顿机器人工程有限公司

-本文档符合 ISO 体系要求，并会根据 IPD 体系变化更新-

目录

第 1 章 前言 1

1.1 读者对象 1

1.2 注意事项 1

1.3 安全说明 1

第 2 章 产品介绍 3

2.1 产品说明 3

2.2 通讯机制 3

2.2.1 **IO 数据交互方式** 3

2.2.2 **以太网通讯交互方式** 4

2.3 ERI 实时控制 5

2.3.1 **概况** 5

2.3.2 **关节控制** 6

2.3.3 **笛卡尔控制** 6

2.4 ERI 纠偏控制 6

2.4.1 **概况** 6

2.4.2 **纠偏坐标系** 6

2.5 ERI 信号处理 8

第 3 章 安全 9

3.1 安全提示 9

第 4 章 安装 10

4.1 系统要求 10

4.1.1 **硬件** 10

4.1.2 **软件** 10

4.2 ERI 软件包安装 10

4.3 通过示教器界面激活 10

第 5 章 配置 12

5.1 C++工程调用 SDK 库 12

5.1.1 环境要求 12
5.1.2 搭建步骤 12
第 6 章 编程 18
6.1 SDK 库接口列表 18
6.1.1 默认构造函数 18
6.1.2 设置机器人通信参数 18
6.1.3 获取机器人连接状态 19
6.1.4 命令接口列表 19
6.1.4.1 获取机器人错误信息 19
6.1.4.2 获取对机器人控制权 20
6.1.4.3 释放对机器人的控制权 20
6.1.4.4 开启实时控制 20
6.1.4.5 MovL 指令 21
6.1.4.6 MovJ 指令 21
6.1.4.7 设置采样周期 21
6.1.4.8 设置 ERI 纠偏参数 22
6.1.4.9 设置工具 22
6.1.4.10 设置用户坐标系 22
6.1.4.11 获取工具参数 23
6.1.4.12 获取用户坐标系 23
6.1.4.13 修改工具参数 23
6.1.4.14 修改用户坐标系 24
6.1.4.15 获取机器人当前笛卡尔位置 24
6.1.4.16 获取机器人当前关节轴数据 24
6.1.4.17 获取当前系统所有的 DO 状态 25
6.1.4.18 获取指定输出口的状态 25
6.1.4.19 设置指定输出口的状态 25
6.1.5 实时接口列表 26
6.1.5.1 实时关节运动 26

6.1.5.2 实时笛卡尔运动 26
6.1.5.3 实时偏移运动 27
6.1.6 状态数据回调接口 27
6.2 ERI 指令概览 27
6.2.1 SetERIProgID 28
6.2.2 EnableERIControl 28
6.2.3 SetERIParam 29
6.2.4 SetCorrParam 29
6.2.5 EnableERICorrect 30
第 7 章 示例配置和示例程序 31
7.1 接口使用样例 31
7.1.1 获取控制权 31
7.1.2 MOVJ 命令 31
7.1.3 MOVL 命令 32
7.1.4 实时运动 CPOS 32
7.1.5 实时运动 APOS 33
7.1.6 纠偏点位发送 34
7.1.7 释放控制权 35
7.1.8 UDP 状态包使用示例 35
第 8 章 典型案例 37
8.1 客户端连接机器人并实时控制 37
8.1.1 通过关节实时控制自行规划运动操作机器人到目标位置 38
8.1.2 通过笛卡尔实时控制控制机器人画圆 41
8.2 示教程序与外部实时数据交替运行 44
8.3 通过 SIMIO 控制机器人纠偏的使用样例 46
附录一：错误列表 48

第 1 章 前言

本手册适用于控制系统 RCS2 系列，介绍埃斯顿机器人 ERI 功能使用方法。

1.1 读者对象

本手册仅供埃斯顿机器人相关技术支持人员使用。

1.2 注意事项

- 在安装和调试这些组件时，操作人员必须严格遵循本文档的说明和解释。
- 相关负责人员必须确保所述产品的应用或使用满足所有安全要求，包括相关法律、法规、准则和标准。
- 尽管本文档经过精心编制，但由于其中所描述的产品仍处于不断更新换代中，我们可能不会在每次更新后都检查文档中所描述的产品性能数据、标准或其它特性总是与实际产品相一致。
- 本文档中难免会出现一些技术或者编辑错误，我们保留随时对文档信息做出修改之权力，恕不另行通知。对于已经变更的产品，如果本文档中的数据、图表以及文字描述没有修改，我们将不再特别加以声明。
- 任何人不得对软、硬件配置进行文本档中规定之外的修改，ESTUN 公司对因此而造成的一切后果不承担任何责任。
- 本文档中出现图示单位在没有特别标注说明时，默认单位为毫米 mm。

1.3 安全说明

⚠ 警告	受伤的危险 不遵守本标志相关的安全说明将危及个人生命和健康安全。
⚠ 注意	对环境和设备有危险 不遵守本标志相关安全说明可能明显危害环境和设备安全。

说明	说明或提示 该标志表示这些信息能够帮助您更好的理解安全说明。

第 2 章 产品介绍

2.1 产品说明

针对一些对实时性、自主扩展性有较高要求的高级应用现场，如实时位置引导、焊缝跟踪、力控装配等应用，埃斯顿机器人提供了一套外部控制接口（以下简称 ERI 库）供客户进行二次开发来实现对机器人的控制。

ERI 具备如下特性：

- 通讯协议： 通过 TCP/IP、总线与外部设备之间进行数据交互。
- 实时控制：按照外部系统轨迹规划方式进行周期性的轨迹控制。
- 纠偏控制：通过处理传感器数据修正机器人轨迹。

2.2 通讯机制

机器人控制系统和外部系统之间可用多种方式相互通讯和交换数据：

- 通过机器人控制系统的 SimIO
- 通过 SDK 接口与机器人交互

2.2.1 IO 数据交互方式

机器人控制系统的输入/输出系统进行数据交互的流程如下：

- 外部系统可通过向机器人控制系统的 SIMDI/SIMAI 中写入数据和信号。ERI 会周期性读取并处理这些数据和信号。机器人控制系统会将已处理的数据和结果通过 SIMDO/SIMAO 返回给外部系统。
- 外部系统可通过总线系统（EIP、PN 通讯）与机器人控制系统的输入/输出系统进行映射。

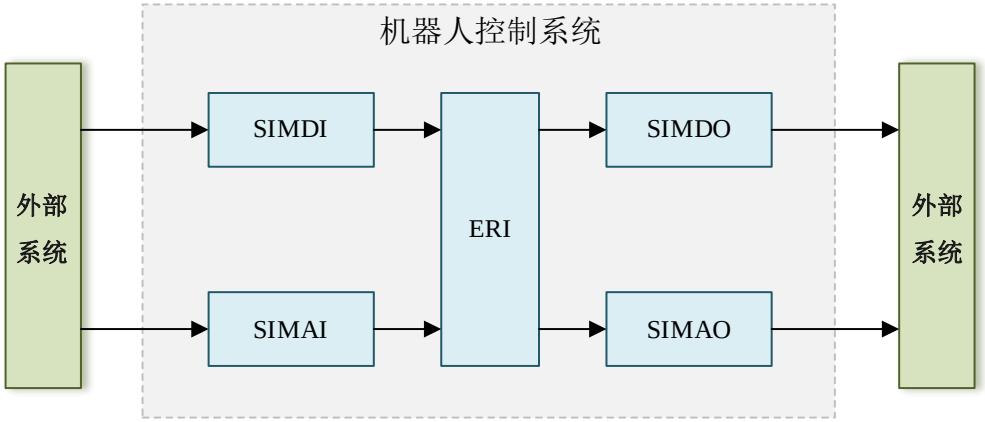


图 2-1 通过 IO 数据交互示意图

2.2.2 以太网通讯交互方式

SDK 库内部封装了与机器人系统的实时通讯交互机制，其中可支持的功能：

- 机器人控制系统会周期性向 SDK 中传输数据，例如笛卡尔坐标位置信息、关节坐标位置信息、运行状态、错误信息等。
- 在机器人控制系统切到 ERI 模式下时，外部系统可调用 SDK 中的实时函数接口向机器人控制系统传输数据，如传感器测量值、规划的点位信息等
- 在机器人控制系统切到 ERI 模式下时，外部系统可调用 SDK 中的命令函数接口向机器人控制系统发送命令数据，如运动命令、清除错误命令、实时功能开启/关闭命令等。

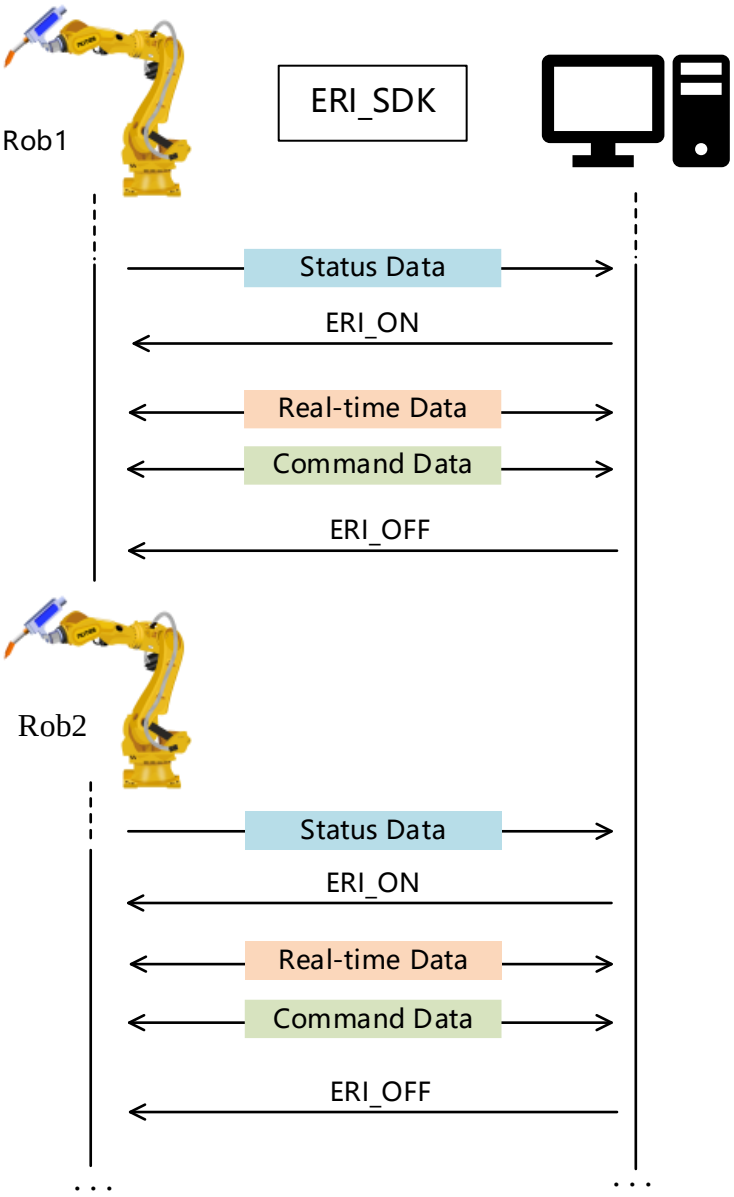


图 2-2 通过 SDK 数据交互示意图

2.3 ERI 实时控制

2.3.1 概况

ERI 实时控制可以让用户跳过控制器底层的轨迹规划，通过 ERI 接口向控制器高速写入关节位置或笛卡尔位置。

2.3.2 关节控制

关节控制实现了通过 ERI 实时向控制器下发规划的关节位置，机器人各轴按照下发的关节位置进行运动。

2.3.3 笛卡尔控制

笛卡尔控制实现了通过ERI实时向机器人下发规划的TCP位置，包括（X、Y、Z、A、B、C）其中姿态为 ZYX 顺序的欧拉角，机器人按照下发的 TCP 位置进行运动。

2.4 ERI 纠偏控制

2.4.1 概况

ERI纠偏控制可以使用户在编写好的示教轨迹基础上根据工艺需要实时调整轨迹。

2.4.2 纠偏坐标系

ERI纠偏控制的纠偏量将在示教轨迹的基础上基于指定的纠偏坐标系进行偏移，偏移量（dX、dY、dZ、dA、dB、dC），其中姿态为基于指定纠偏坐标系，按照 XYZ 顺序绕固定轴旋转的偏移，纠偏控制可以基于多个坐标系：

- WORLD

如图 2-3 所示为基于 WORLD 坐标系的纠偏，在机器人运行示教轨迹（橘色）的同时，缓慢增加 X 方向纠偏量，机器人实际轨迹（紫色）将为示教轨迹和沿 WORLD 坐标系 X 轴方向的纠偏量的叠加。

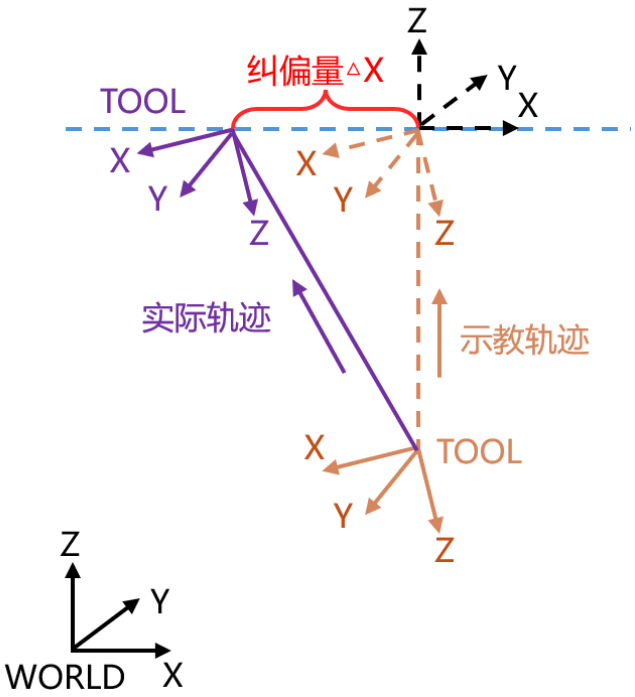


图 2-3 基于 WORLD 坐标系的纠偏

- TOOL

如图 2-4 所示为基于 TOOL 坐标系的纠偏，在机器人运行示教轨迹（橘色）的同时，缓慢增加 X 方向纠偏量，机器人实际轨迹（紫色）将为示教轨迹和沿 TOOL 坐标系 X 轴方向的纠偏量的叠加。

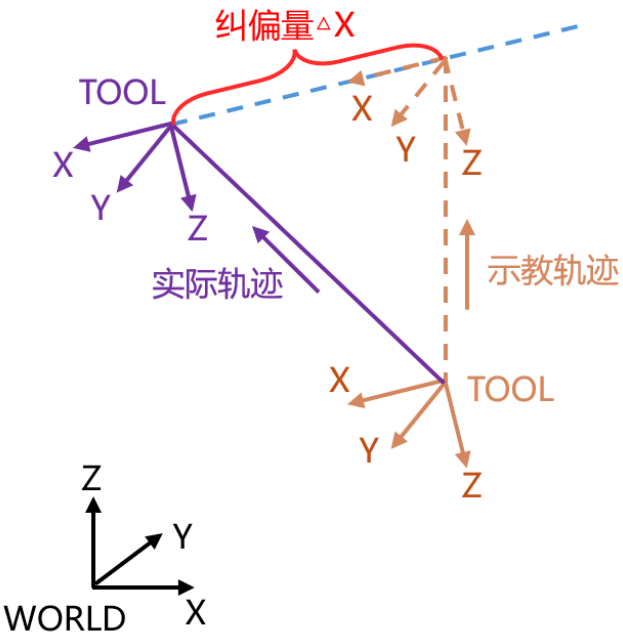


图 2-4 基于 TOOL 坐标系的纠偏

- TTS

TTS（Trajectory Tangent System）坐标系定义如下：原点始终绑定在机器人当前 TCP 位置，X 轴沿运动轨迹的瞬时切线方向（即路径前进方向），Y 轴由路径切线（X 轴）与工具坐标系的 X 轴叉积确定，Z 轴由该坐标系的 X 轴和 Y 轴叉积确定。如图 2-5 所示为基于 TTS 坐标系的纠偏，在机器人运行示教轨迹（橘色）的同时，缓慢增加 Z 方向纠偏量，机器人实际轨迹（紫色）将为示教轨迹和沿 TTS 坐标系 Z 轴方向的纠偏量的叠加。

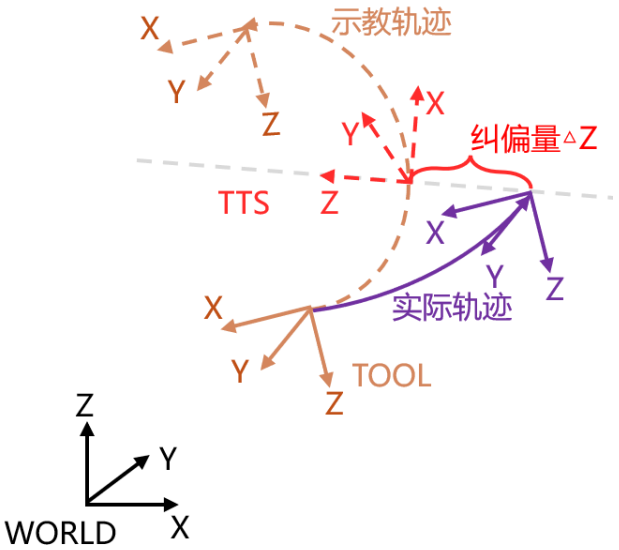


图 2-5 基于 TTS 坐标系的纠偏

2.5 ERI 信号处理

用户在使用 ERI 实时控制与纠偏控制时，可高速获取机器人状态，包括：笛卡尔位置、关节位置、纠偏状态、IO 状态、错误信息等。

ERI 可接收外部系统高速实时数据（周期默认 4ms），同时提供三种滤波模式处理规划数据，若用户运动规划比较光滑，可关闭滤波器以降低延迟。

第 3 章 安全

3.1 安全提示

错误使用ERI时，可能导致人员受伤和财产损失，如：

- 在实时模式或者纠偏模式下，发送了异常位置信息。
- 在硬件存在故障，如布线错误，外部传感器故障等。

在ERI模式下使用实时控制时，系统集成商有责任通过建立合适的安全措施将受伤危险以及财产损失降低到最小程度，例如：根据机器人的安全行程范围在外部系统中设立合理的工作区域。

纠偏模式下，ERI纠偏控制默认纠偏阈值为±6mm/°，用户可通过指令修改该阈值。当外部发来的纠偏量超阈值时，系统会按照设置的纠偏阈值来执行该纠偏量。

第 4 章 安装

ERI 软件功能包可借助于示教器界面的升级功能进行安装，待安装完成后需激活才可使用。

4.1 系统要求

4.1.1 硬件

- ERC60F 系列机器人控制器；
- 通过 SDK 库进行数据交换时，带实时操作系统和 100Mbit 实时网卡的、在全双工模式下受处理器保护的外部系统；
- 埃斯顿机器人

4.1.2 软件

V1.36.C01.A 及以上的系统软件版本。

4.2 ERI 软件包安装

通过实体示教器安装，操作步骤如下：

步骤 1 将准备更新的控制器运行程序（.runtime或.patch格式文件，该文件通常由厂家人员提供）放入U盘根目录。

步骤 2 将U盘插入示教器USB接口, 在示教器主界面中选择系统主页->设置->备份更新->控制器备份恢复，点击进入控制器升级界面。

步骤 3 点击【更新】功能按键，输入管理人员密码进行确认后，继续下一步操作。

步骤 4 选择更新文件后将进入自动更新状态，然后等待更新完成即可。

4.3 通过示教器界面激活

如需使用 ERI 软件包，需要对 ERI 功能进行授权激活，可以通过以下步

骤进行激活：



图 4-1 ERI 功能激活界面

步骤 1 在管理员权限下，依次进入“应用”-“ERI”界面，通过点击界面中的获取按钮获得产品码，将此产品码发送给厂家，由厂家处获取激活码（与产品码一一对应）；（如果进入界面时，激活状态为已激活，则不需要此激活配置过程）。

步骤 2 通过将由厂家获取的激活码输入到激活码编辑框中，点击激活按钮，如果输入的激活码正确，则状态变为授权；如果输入的激活码不正确，则状态仍为未授权。

第 5 章 配置

5.1 C++工程调用 SDK 库

5.1.1 环境要求

配置项	最低要求	推荐配置	备注
系统版本	Win7	Win10	
内存大小	4GB	8GB	
VisualStudio 版本	2015	2019	如有其他 VS 版本，可联系厂家

5.1.2 搭建步骤

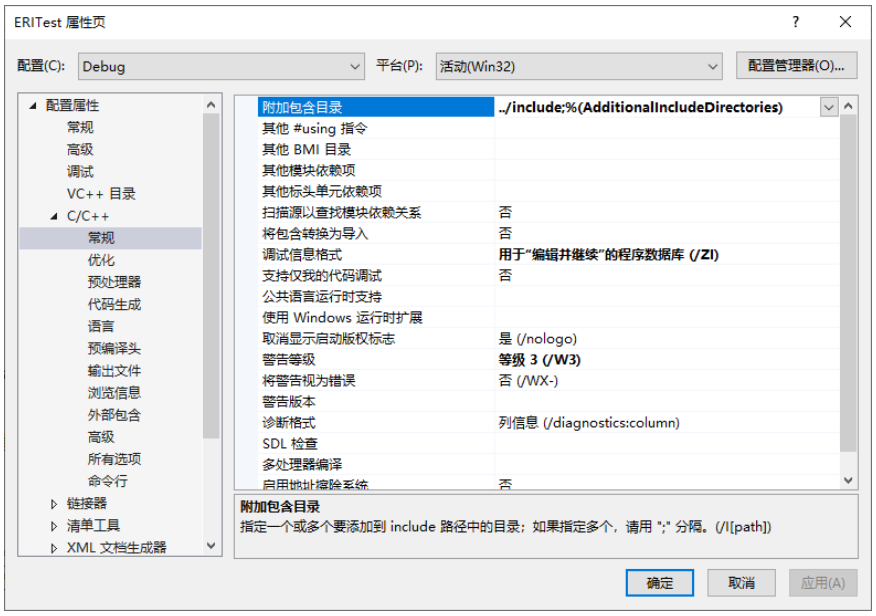
步骤 1 新建工程

打开 vs2019，点击左上角【文件】-【新建】-【项目】，选择项目类型为【控制台应用】，如图示为项目命名并选择其存放位置，完成新建工程。



步骤 2 配置头文件路径和库路径

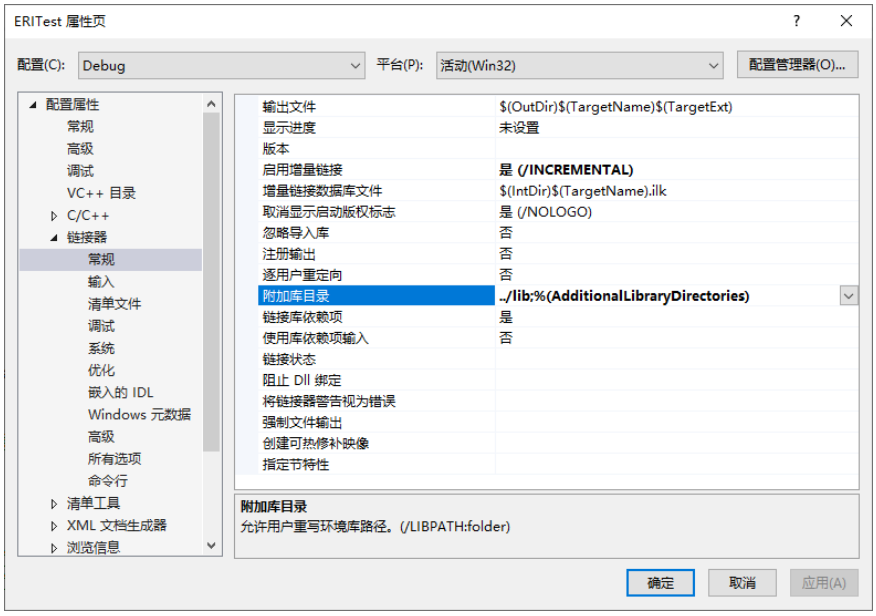
1）右键工程，点击【属性】，点击【配置属性】-【C/C++】-【常规】，在【附加包含目录】中添加 ERI 库的 include 目录。



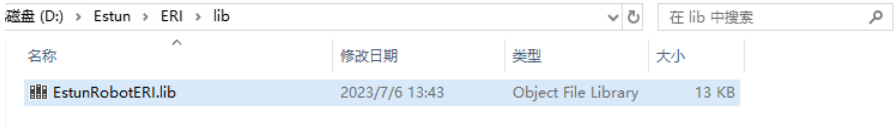
include 目录内包含 ERI 头文件，如下图所示。



2）右键工程，点击【属性】，点击【配置属性】-【链接器】-【常规】，在【附加库目录】中添加 ERI 库的 lib 目录。

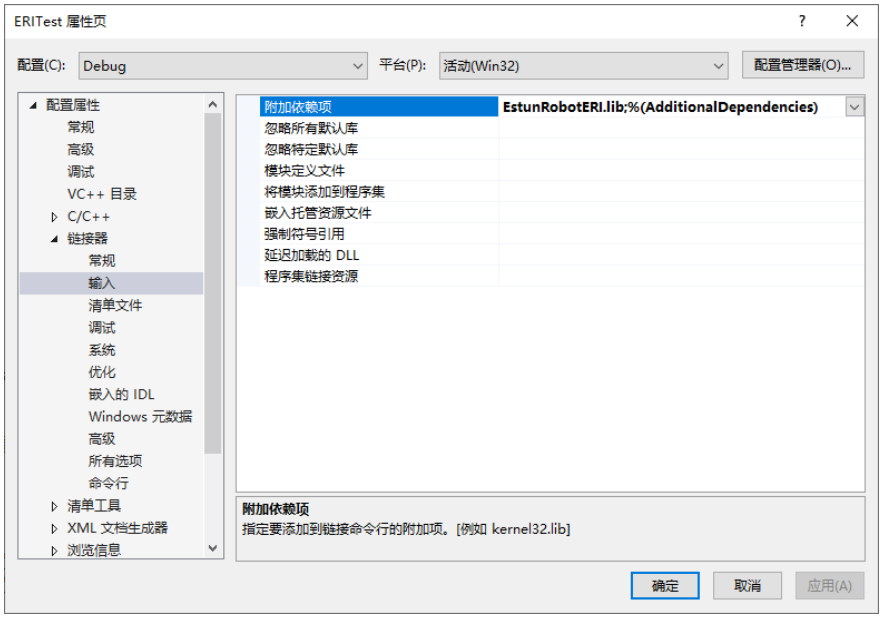


lib 目录内包含文件如下图所示。



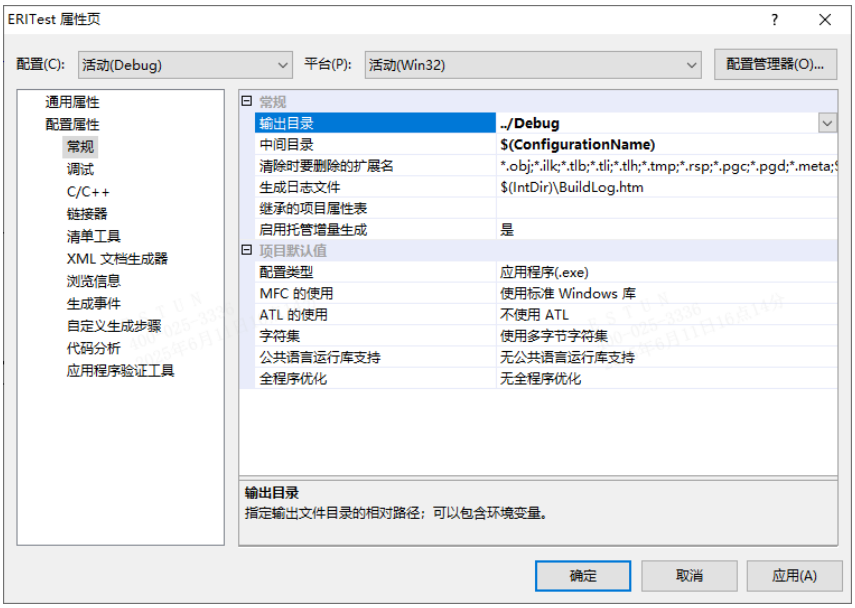
步骤 3 配置外部依赖库

右键工程，点击【属性】，点击【配置属性】-【链接器】-【输入】，在【附加依赖项】中添加 ERI 库的.lib 文件，文件名为“EstunRobotERI.lib”（32 位）。

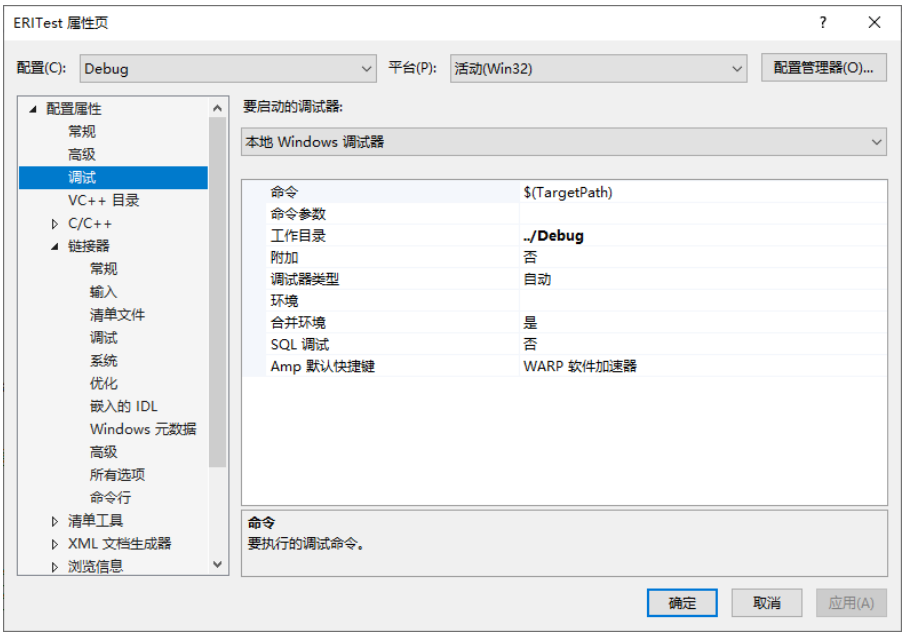


步骤 4 配置输出目录和工作目录

1）右键工程，点击【属性】，点击【配置属性】-【常规】-【输出目录】，可添加自定义的.exe 文件输出路径。（建议使用相对路径，例如“../Debug”，表示指定当前工程输出文件路径为：当前位置父目录下的 Debug 文件夹。）



2）右键工程，点击【属性】，点击【配置属性】-【调试】-【工作目录】，对工作目录路径进行配置。（**“工作目录路径”与”输出目录路径”建议保持一致**，此例中，二者路径均配置为 “ ../Debug ” ，表示指定工作目录路径为：当前位置父目录下的 Debug 文件夹。）



步骤 5 添加引用

在步骤 4 所配置的工程输出目录下，添加 ERI 库的引用文件。（以输出目录为“ ../Debug ”时举例，在该目录下粘贴如下图所示的文件。）



步骤 6 添加程序逻辑，调用 ERI 接口

包含头文件

```
#include "ERIParamManager.h"
```

实现调用逻辑

```
//创建ERI功能库对象
ERIParamManager eriManager;
//注册状态回调函数
eriManager.setRobotStatusCallback(RobotStatusCallback);
//设置机器人连接参数，以连接机器人
string robotIp = "172.31.16.68";
UINT16 cmdPort = 61210;
UINT16 servoPort = 61211;
UINT16 statusPort = 61212;
eriManager.setCommunicationParam(robotIp.c_str(),cmdPort,servoPort,statusPort);
//获取机器人控制权限
bool resGetCtrl = eriManager.getControlRobot();
//执行MovJ
double targetJoint[6];
targetJoint[0] = -100.262151;
targetJoint[1] = 23.006887;
targetJoint[2] = 42.440697;
```

```
targetJoint[3] = 3.420175;
targetJoint[4] = 88.415222;
targetJoint[5] = 29.849081;
bool resMovj = eriManager.movj(targetJoint);
//释放控制权
eriManager.releaseControlOverRobot();
```

步骤 7 编译，运行

右键项目，启动实例，如果提示找不到 xxx.dll，参考步骤 5。

第 6 章 编程

6.1 SDK 库接口列表

ERI 的 SDK 库函数分为命令接口、实时接口、状态接口三种方式，其中:

- 命令接口：命令执行为阻塞式机制，即发送数据后必须等待执行结束后才会返回结果。特别是运动指令，必须等当前运动结束了才能下发新运动指令。
- 实时接口：实时接口为发送完数据即返回结果，具体的发送频率由用户决定，但需要保证与控制器采样周期的频率保持一致。如外部设备每间隔 20ms 调用实时接口向控制器发送点位数据，那应该先调用“设置采样周期”函数将采样周期设为 20ms。
- 状态接口：提供 UDP 状态回调函数，SDK 库会每隔 2ms 调用该函数推送控制器相关的状态数据。也可通过命令接口中的部分函数获取，如通过调用获取笛卡尔位置/关节位置函数获取坐标。

6.1.1 默认构造函数

名称	ERIParamManager::ERIParamManager()
功能	类的默认构造接口。

6.1.2 设置机器人通信参数

名称	int ERIParamManager::setCommunicationParam(char *robotIp,UINT16 cmdPort,UINT16 servoPort,UINT16 statusPort)

功能	设置机器人通信参数，设置完毕客户端会根据设置参数，自动连接机器人。
返回值	返回值为 0，设置成功；返回值为 1，设置失败。
参数说明	* @robotIp 机器人 ip 地址 * @cmdPort 命令通道端口号 * @servoPort servo 通道端口号 * @statusPort 状态包通道端口号 * @返回值 0 设置成功 1 设置失败

6.1.3 获取机器人连接状态

名称	void getRobotConnStatus(char &cmdStatus,char &servoStatus,char &udpStatus)
功能	获取机器人连接情况
返回值	void
参数说明	* @ cmdStatus 命令通道连接状态 成功 1 * @ servoStatus 实时通道连接状态 成功 1 * @ udpStatus 机器人状态反馈连接状态 成功 1

6.1.4 命令接口列表

此分类下为命令相关接口。

6.1.4.1 获取机器人错误信息

名称	bool ERIParamManager::getCurErrMsg(int &errId, std::string &errMsg)

功能	获取机器人错误信息。
返回值	成功返回 true，失败返回 false。
参数说明	* @errId 错误 ID * @errMsg 错误消息内容

6.1.4.2 获取对机器人控制权

名称	bool ERIParamManager::getControlRobot()
功能	获取对机器人控制权。
返回值	成功返回 true，失败返回 false。
参数说明	无

6.1.4.3 释放对机器人的控制权

名称	bool ERIParamManager::releaseControlOverRobot()
功能	释放对机器人的控制权。
返回值	成功返回 true，失败返回 false。
参数说明	无

6.1.4.4 开启实时控制

名称	bool ERIParamManager::startServo(int motionType = 0)
功能	开始实时控制。
返回值	成功返回 true，失败返回 false。

参数说明 * @motionType 运动类型，0 为关节，1 为笛卡尔

6.1.4.5 MovL 指令

名称 bool ERIParamManager::movL(double pos[])

功能 执行 MovL 运动。

返回值 成功返回 true，失败返回 false。

参数说明 * @pos 点位信息

6.1.4.6 MovJ 指令

名称 bool ERIParamManager::movj(double pos[])

功能 执行 MovJ 运动。

返回值 成功返回 true，失败返回 false。

参数说明 * @pos 点位信息

6.1.4.7 设置采样周期

名称 bool ERIParamManager::setERIMotionParam(UINT16 period,UINT16 filterType)

功能 设置采样周期。

返回值 成功返回 true，失败返回 false。

参数说明 * @period 采样周期，单位 ms
* @filterType 滤波类型， 0-关闭滤波 1-平均滤波值 2-二阶低通滤波 3-椭圆滤波

6.1.4.8 设置 ERI 纠偏参数

名称	bool ERIParamManager::setCorrectParam(double alarmThreshold[6],UINT16 corrCoord)
功能	设置 ERI 纠偏参数。
返回值	成功返回 true，失败返回 false。
参数说明	* @alarmThreshold 纠偏量报警阈值，单位 mm/° * @corrCoord 纠偏坐标系， 0-世界坐标系 1-工具坐标系 2-TTS 坐标系

6.1.4.9 设置工具

名称	bool ERIParamManager::setTool(const UINT16 toolId)
功能	设置工具。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 TOOL。
返回值	成功返回 true，失败返回 false。
参数说明	* @toolId 工具 Id

6.1.4.10 设置用户坐标系

名称	bool ERIParamManager::setUser(const UINT16 userId)
功能	设置用户坐标系。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 USERCOORD。
返回值	成功返回 true，失败返回 false。

参数说明	* @userId 用户坐标系 Id

6.1.4.11 获取工具参数

名称	bool ERIParamManager::getTool(int toolId,double toolData[6])
功能	获取工具参数。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 TOOL。
返回值	成功返回 true，失败返回 false。
参数说明	* @toolId 工具 Id * @toolData 对应的工具坐标参数，分别对应 x,y,z,a,b,c

6.1.4.12 获取用户坐标系

名称	bool ERIParamManager::getUser(int userId,double userData[6])
功能	获取用户坐标系。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 USERCOORD）。
返回值	成功返回 true，失败返回 false。
参数说明	* @userId 用户坐标系 Id * @userData 对应的用户坐标系参数

6.1.4.13 修改工具参数

名称	bool ERIParamManager::modifyTool(int toolId,double toolData[6])
功能	修改工具参数。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 TOOL。

返回值	成功返回 true，失败返回 false。
参数说明	* @toolId 要修改的工具 Id * @toolData 对应的工具坐标参数，分别对应 x,y,z,a,b,c

6.1.4.14 修改用户坐标系

名称	bool ERIParamManager:: modifyUser(int userId,double userData[6])
功能	获取用户坐标系。详细含义见《ER 系列机器人操作手册》10.3 系统数据类型 USERCOORD。
返回值	成功返回 true，失败返回 false。
参数说明	* @userId 要修改的用户坐标系 Id * @userData 对应的用户坐标系参数

6.1.4.15 获取机器人当前笛卡尔位置

名称	bool getWorldCpos(double pos[16])
功能	获取机器人世界坐标系下当前位置
返回值	成功返回 true，失败返回 false。
参数说明	* @pos 输出型参数，存放机器人当前位置

6.1.4.16 获取机器人当前关节轴数据

名称	bool getJointValue(double pos[16])

功能　　获取机器人当前各关节轴值

返回值　　成功返回 true，失败返回 false。

参数说明　　* @@pos 输出型参数，存放机器人当前位置

6.1.4.17 获取当前系统所有的 DO 状态

名称　　UINT64 getCurrentDO()

功能　　以十进制格式返回当前系统 64 个 DO 的值

参数说明　　@val 以十进制格式显示当前系统所有的 DO 值

6.1.4.18 获取指定输出口的状态

名称　　UINT32 getCurrentDO(const UINT32 port)

功能　　获取指定输出口的输出状态，仅能获取单个 DO 状态

返回值　　@val 端口值

参数说明　　@port 端口号，返回值为 0 或 1

6.1.4.19 设置指定输出口的状态

名称　　bool setDO(UINT32 port,UINT32 val)

功能　　设置指定输出口的输出状态

返回值	成功返回 true，失败返回 false。
参数说明	@port 待设置的端口号 @val 待设置的端口值，参数可设为 0 或 1

6.1.5 实时接口列表

此分类下为实时相关接口。

6.1.5.1 实时关节运动

名称	int ERIParamManager::servoToAPOS(UINT64& timeStamp,double *pos,bool IsEndPoint = false)
功能	执行实时关节运动。
返回值	返回值为-1，执行失败；其他，执行成功
参数说明	* @timeStamp 时间戳 * @pos 点位信息 * @IsEndPoint 是否结束点

6.1.5.2 实时笛卡尔运动

名称	int ERIParamManager::servoToCPOS(UINT64& timeStamp,double *pos,bocol IsEndPoint = false)
功能	执行实时笛卡尔运动。
返回值	返回值为-1，执行失败；其他，执行成功
参数说明	* @timeStamp 时间戳 * @pos 点位信息 xyzabc（姿态为基于欧拉角表示） * @IsEndPoint 是否结束点

6.1.5.3 实时偏移运动

名称	int ERIParamManager::servoToOFFSET(UINT64& timeStamp, double *pos,bool IsEndPoint = false)
功能	执行实时偏移运动。
返回值	返回值为-1，执行失败；其他，执行成功
参数说明	* @timeStamp 时间戳 * @pos 点位信息：xyzabc（姿态为基于固定坐标系表示） * @IsEndPoint 是否结束点

6.1.6 状态数据回调接口

名称	void setRobotStatusCallback(P_RobotStatusCallback pfunc)
功能	UDP 状态回调函数，通过该设置回调函数方法，设置需要回调的函数信息。
返回值	无
参数说明	* @pfunc 函数指针，指向需要的状态包回调函数
补充说明	typedef void (*P_RobotStatusCallback)(bool isRoborError, char isStartCorrect, bool isEndPoint) /** * @isRobotError 机器人端是否有错误 * @isStartCorrect 是否开始纠偏 * @ isRobotError 是否是结束点 **/

6.2 ERI 指令概览

ERI 的指令需搭配 ERI 的外部接口一起使用，可设置实时纠偏参数与切换

机器人的控制权。

ERI 指令列表：

控制指令	SetERIProgID
	EnableERIControl
	SetERIParam
	SetCorrParam
	EnableERICorrect

6.2.1 SetERIProgID

该指令用于设置 ERI 执行外部程序 ID。通常搭配 EnableERIControl 指令一起使用。当系统的控制权切换到外部控制时，外部系统使用设置的程序编号或者工艺来控制机器人运动。

SetERIProgID ID

• ID：外部程序 ID

示例：

```
SetERIProgID(1)   // 设置外部程序
```

6.2.2 EnableERIControl

该指令用于将机器人控制权切换到ERI。远程模式下执行该指令后，可将机器人的执行权限切换给 ERI 控制，只有在该模式下，外部设备才可控制机器人进行实时运动。

示例：

```
EnableERIControl()   // 将机器人的执行权限切换到 ERI 控制，该命令会阻塞执行，待执行权限切换回机器人控制后才会执行结束。
```

6.2.3 SetERIParam

该指令用于设置采样周期和滤波器类型

SetERIParam period filterType

• period：采样周期 单位 ms

• filterType：滤波器类型，系统内置四种滤波器，可对外部发来的实时位置按照设定的滤波类型进行滤波处理。

“DisEnable” 关闭滤波器

“Average” 平均滤波器

“LowPass” 二阶低通滤波

“Ellipse” 椭圆滤波

示例：

SetERIParam(8, ”Average”) // 系统将按照 8ms 来运行点位数据，并利用平均滤波器对数据进行滤波处理。

6.2.4 SetCorrParam

设置报警阈值、纠偏坐标系等

SetCorrParam XCThreshold YCThreshold ZCThreshold ACThreshold BCThreshold CCThreshold corrCoord

• XCThreshold：相对纠偏坐标系 X 轴方向最大纠偏量

• YCThreshold：相对纠偏坐标系 Y 轴方向最大纠偏量

• ZCThreshold：相对纠偏坐标系 Z 轴方向最大纠偏量

• ACThreshold：相对纠偏坐标系 A 轴方向最大纠偏量

• BCThreshold：相对纠偏坐标系 B 轴方向最大纠偏量

• CCThreshold：相对纠偏坐标系 C 轴方向最大纠偏量

• corrCoord：纠偏数据所参考的坐标系类型

“World” 世界坐标系

“Tool” 工具坐标系

“TTS” TTS 坐标系

示例：

SetCorrParam(5,5,5,5,5,5,0) // 将 XYZABC 的最大纠偏量设为 5mm，同时设置计算纠偏数据所基于的坐标系类型为世界坐标系。

6.2.5 EnableERICorrect

该指令用于开启/关闭 ERI 的实时纠偏功能，通常搭配外部实时接口使用。开启后系统会根据发来的纠偏数据调整机器人的运行轨迹。

EnableERICorrect EnableSts

• EnableSts：使能状态

示例：

EnableERICorrect (ON) // 开启纠偏功能

第 7 章 示例配置和示例程序

7.1 接口使用样例

本节主要通过基本样例进一步介绍 ERI 库主要接口的使用。

7.1.1 获取控制权

```
ERIParamManager eriManager;
if (eriManager.getControlRobot())
{
   printf("获取控制权成功!\n");
}
else
{
   printf("获取控制权失败!\n");
}
```

7.1.2 MOVJ 命令

```
ERIParamManager eriManager;
double targetJoint[6];
targetJoint[0] = -1.262151;
targetJoint[1] = 23.006887;
targetJoint[2] = 32.440697;
targetJoint[3] = 3.420175;
targetJoint[4] = 48.415222;
targetJoint[5] = 14.849081;
if (eriManager.movj(targetJoint))
{
```

```
   printf("movj success \n");
}
else
{
   printf("movj failed \n");
}
```

7.1.3 MOVL 命令

```
ERIParamManager eriManager;
double targetJoint[6];
targetJoint[0] = 855.84;
targetJoint[1] = -10.379000000000019;
targetJoint[2] = 561.518;
targetJoint[3] = -154.89800000000014;
targetJoint[4] = 25.520999999999958;
targetJoint[5] = -162.25800000000004;
if (eriManager.movL(targetJoint))
{
   printf("MovL success \n");
}
else
{
   printf("MovL failed \n");
}
```

7.1.4 实时运动 CPOS

```
if (eriManager.startServo(1))
{
```

```
g_eriSDKLog->info("start servo cpos success");
 while(true)
{
        bool isEndPoint = false;
        double jointData[6] = {0};
        UINT64 currentTimeStamp = 0;
        if (-1 == eriManager.servoToCPOS(currentTimeStamp,jointData,isEndPoint))
        {
            g_eriSDKLog->error("servoToCpos error stop!");
            return;
        }

}
}
else
{
   g_eriSDKLog->debug("start servo cpos failed");
}
```

7.1.5 实时运动 APOS

```
if (eriManager.startServo(0))
{
   printf("start servo movj success \n");
   while(true)
   {
bool isEndPoint = false;
        double jointData[6] = {0};
        UINT64 currentTimeStamp = 0;
        if (-1 == eriManager.servoToAPOS(currentTimeStamp,jointData,isEndPoint))
```

```
        {
            g_eriSDKLog->error("servoToCpos error stop!");
            return;
        }

    }

}
else
{
    printf("start servo movj failed \n");
}
```

7.1.6 纠偏点位发送

```
While(true)
{
bool isEndPoint= false;
double jointData[6] = {0.0};
UINT64 currentTimeStamp = 0;
if (-1 == eriManager.servoToOFFSET(currentTimeStamp,jointData,isEndPoint))
{
    g_eriSDKLog->error("servoToOFFSET error stop!");
    return;

}
}
```

7.1.7 释放控制权

```
ERIParamManager eriManager;


if (eriManager.releaseControlOverRobot())
{
   printf("释放控制权成功!\n");
}
else
{
   printf("释放控制权失败!\n");
}
```

7.1.8 UDP 状态包使用示例

- 定义回调函数

```
//机器人状态信息
 Struct RobotStatus
{
   bool isRobotError; //机器人端是否有错误
   char isStartCorrect;//是否开始纠偏
   bool isEndPoint;//机器人处理完毕最后一个点位
   double worldCpos[6];//当前机器人世界坐标系下位置
   double jointValue[6];//当前机器人各关节轴值
};
void RobotStatusCallback(RobotStatus robotStatus)
{
}
```

- 注册回调函数

```
ERIParamManager eriManager;
```

eriManager.setRobotStatusCallback(RobotStatusCallback);

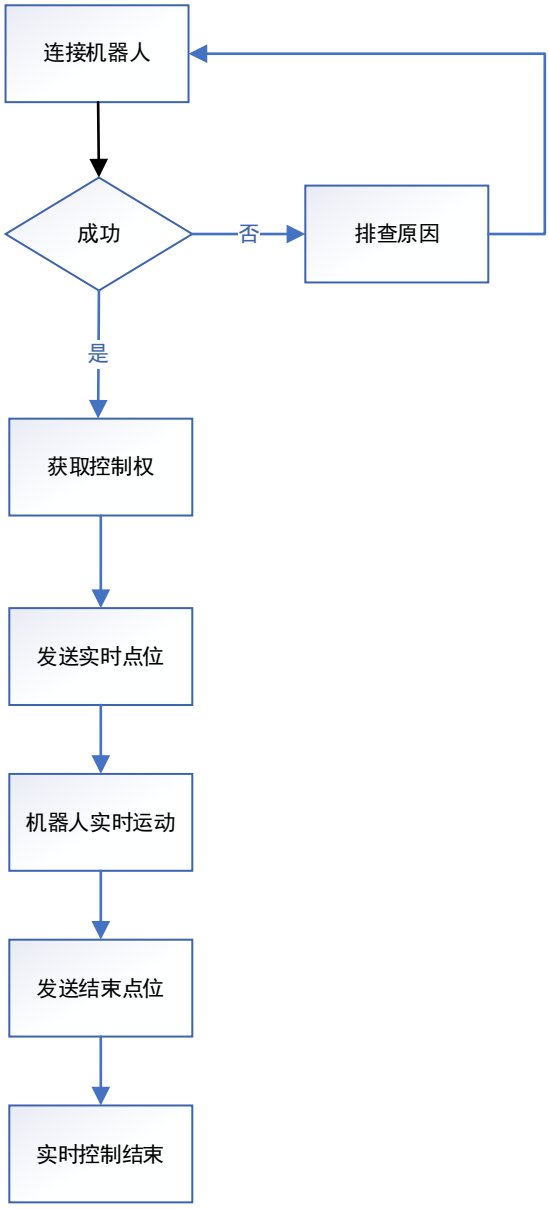
在该回调函数中可周期性接收，机器人有无错误(robotStatus.isRobotError)、纠偏是否开始(robotStatus.isStartCorrect)、实时运动是否结束(robotStatus.isEndPoint)

第 8 章 典型案例

8.1 客户端连接机器人并实时控制

在进行该操作前，请确保机器人系统一切正常，无任何错误，机器人可正常点动。

整体流程：



8.1.1 通过关节实时控制自行规划运动操作机器人到目标位置

示例代码如下：

```
// ============================================================================
// 示例名称：ERI 实时控制插补示例
// 功能：连接机器人 → 获取控制权 → 设置运动参数 → MoveJ 到起点 → 启动实时控制
//       → 多维五次多项式轨迹插补 → 实时发送目标位姿 → 结束运动 → 释放控制权
// ============================================================================

#include "stdafx.h"
#include <iostream>
#include "ERIParamManager.h"

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// ------------------------ ERI 控制器实例 ------------------------
ERIParamManager eriManager;

// ------------------------ 五次多项式插值函数 ------------------------
double compute5thPoly(double p0, double p1, double T, double t) {
    double s = t / T;
    double s3 = s * s * s;
    double s4 = s3 * s;
    double s5 = s4 * s;
    return p0 + (p1 - p0) * (10.0 * s3 - 15.0 * s4 + 6.0 * s5);
}

// ------------------------ 实时插补运动主函数 ------------------------
void processRealtimeData() {
    // ===== 控制与滤波参数 =====
    const int sampleTime = 4;   // 控制周期（ms）
    const int filterType = 0;   // 0 = 无滤波

    // ===== 起点与终点位姿 =====
    double startPos[6] = { 0, 0, 0, 0, 0, 0 };          // 初始位姿（可替换为实际读取
值）
    double goalPos[6] = { 20, 30, 40, 50, 60, 70 };   // 目标位姿
    double target[6];                                  // 实时目标位姿
    UINT64 timeStamp = 0;                              // 时间戳

    // ===== 获取控制权 =====
```

```
if (!eriManager.getControlRobot()) {
    std::cerr << "获取机器人控制权失败!" << std::endl;
    return;
}
std::cout << "成功获取机器人控制权!" << std::endl;

// ===== 设置实时控制参数 =====
if (!eriManager.setERIMotionParam(sampleTime, filterType)) {
    std::cerr << "设置ERI运动参数失败!" << std::endl;
    return;
}
std::cout << "已设置实时控制参数。" << std::endl;

// ===== MoveJ 到起始位置 =====
if (!eriManager.movJ(startPos)) {
    std::cerr << "MoveJ 到起始点失败!" << std::endl;
    return;
}
std::cout << "MoveJ 成功！等待稳定…" << std::endl;
Sleep(1000);

// ===== 启动实时控制（APOS模式） =====
if (!eriManager.startServo(0)) {
    std::cerr << "启动实时控制失败!" << std::endl;
    return;
}
std::cout << "已启动实时控制。" << std::endl;

// ===== 初始化插补时间参数 =====
const double T = 2000.0;  // 插补总时间（ms）
double t = 0.0;           // 当前时间（ms）
std::copy(startPos, startPos + 6, target); // 初始目标位姿赋值

// ===== 实时插补主循环（五次多项式）=====
while (t <= T) {
    for (int i = 0; i < 6; ++i) {
        target[i] = compute5thPoly(startPos[i], goalPos[i], T, t);
    }

    if (eriManager.servoToAPOS(timeStamp, target, false) == -1) {
        std::cerr << "实时目标发送失败!" << std::endl;
        break;
    }
```

```
        timeStamp += sampleTime;
        t += sampleTime;
    }

    // ===== 发送最终点，标记为终点 =====
    eriManager.servoToAPOS(timeStamp, target, true);
    Sleep(1000);

    // ===== 释放机器人控制权 =====
    if (eriManager.releaseControlOverRobot())
        std::cout << "控制权释放成功。" << std::endl;
    else
        std::cout << "释放控制权失败。" << std::endl;
}

// ------------------------ 主程序入口 ------------------------
int main() {
    // 设置机器人通信参数（根据实际修改）
    std::string robotIp = "172.31.16.204";
    uint16_t cmdPort = 61210;
    uint16_t servoPort = 61211;
    uint16_t statusPort = 61212;

    // 配置通信
    if (!eriManager.setCommunicationParam(robotIp.c_str(), cmdPort, servoPort,
statusPort)) {
        std::cout << "机器人通信参数设置失败!" << std::endl;
        return -1;
    }
    Sleep(50); // 等待通信稳定

    // 用户输入控制界面
    char input = '0';
    while (true) {
        std::cout << "输入指令: (3:开始插补, q:退出): ";
        std::cin >> input;

        if (input == 'q') break;
        if (input == '3') processRealtimeData();
    }

    return 0;
}
```

8.1.2 通过笛卡尔实时控制控制机器人画圆

示例代码如下：

```
// ================================================================================
// 示例名称：ERI 实时圆轨迹控制
// 功能：控制机器人 TCP 沿一个平面圆轨迹运动，使用五次多项式进行平滑插值
// 步骤：连接 → 获取控制权 → 参数设置 → MoveJ → 获取起点 → 实时圆轨迹控制 → 终点
→ 释放控制权
// ================================================================================

#include "stdafx.h"
#include <iostream>
#include "ERIParamManager.h"

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// ------------------------ ERI 控制器实例 ------------------------
ERIParamManager eriManager;

// ------------------------ 插值函数（五次多项式） ------------------------
double computeInterpolatedTheta(double t, double T, double theta0, double theta1) {
    double s = t / T;
    return theta0 + (theta1 - theta0) * (10 * pow(s, 3) - 15 * pow(s, 4) + 6 * pow(s,
5));
}

// ------------------------ 实时圆轨迹执行函数 ------------------------
void processRealtimeData() {
    // ===== 控制与滤波参数 =====
    const int sampleTime = 4;    // 控制周期（ms）
    const int filterType = 0;    // 滤波器类型（0 = 无滤波）

    // ===== 初始位姿（MoveJ 目标） =====
    double startPos[6] = { 1.224, -0.999, 19.75, 0, 71.249, 1.209 };  // 自定义起点
    double startTCP[6];    // 实际TCP起点（从控制器读取）
    double target[6];      // 每周期的实时目标位置
    UINT64 timeStamp = 0;  // 实时控制时间戳

    // ===== 圆轨迹参数 =====
    const double r = 50.0;          // 圆半径（mm）
    double centerX, centerY, centerZ; // 圆心坐标
```

```
const double theta_0 = 0.0;
const double theta_1 = 2 * M_PI; // 绕一整圈

// ===== 插值参数 =====
const double T = 1000.0 * sampleTime;  // 总时长（ms）
const double dt = 1.0 * sampleTime;    // 每周期步进（ms）
double t = 0.0;            // 当前时间

// ===== 获取机器人控制权 =====
if (!eriManager.getControlRobot()) {
    std::cerr << "获取机器人控制权失败!" << std::endl;
    return;
}
std::cout << "已获取控制权。" << std::endl;

// ===== 设置实时控制参数 =====
if (!eriManager.setERIMotionParam(sampleTime, filterType)) {
    std::cerr << "设置实时控制参数失败!" << std::endl;
    return;
}

// ===== MoveJ 到初始位置 =====
if (!eriManager.movJ(startPos)) {
    std::cerr << "MoveJ 失败！" << std::endl;
    return;
}
std::cout << "MoveJ 成功。" << std::endl;
Sleep(1000); // 等待稳定

// ===== 获取当前 TCP 位置作为轨迹起点 =====
if (!eriManager.getWorldCpos(startTCP)) {
    std::cerr << "获取当前TCP位置失败!" << std::endl;
    return;
}

// 打印当前 TCP 位置
std::cout << "当前TCP位置：";
for (int i = 0; i < 6; ++i) {
    std::cout << startTCP[i] << " ";
    target[i] = startTCP[i]; // 初始化目标位姿
}
std::cout << std::endl;

// ===== 计算圆心位置（假设圆心在 X 方向左移 r）=====
```

```
    centerX = startTCP[0] - r;
    centerY = startTCP[1];
    centerZ = startTCP[2]; // 保持Z不变

    // ===== 启动实时控制（CPOS模式）=====
    if (!eriManager.startServo(1)) {
        std::cerr << "开启实时控制失败!" << std::endl;
        return;
    }
    std::cout << "实时控制已开启。" << std::endl;

    // ===== 主控制循环：发送轨迹点 =====
    while (t <= T) {
        // 插值计算当前角度
        double theta = computeInterpolatedTheta(t, T, theta_0, theta_1);

        // 计算当前圆周点（X-Y 平面圆）
        target[0] = centerX + r * std::cos(theta); // X
        target[1] = centerY + r * std::sin(theta); // Y
        target[2] = centerZ;                       // Z 保持不变

        // 发送实时目标位姿
        if (eriManager.servoToCPOS(timeStamp, target, false) == -1) {
            std::cerr << "实时控制发送失败!" << std::endl;
            break;
        }

        // 更新时间戳与时间变量
        timeStamp += sampleTime;
        t += dt;
    }

    // ===== 最后一点作为终点 =====
    eriManager.servoToCPOS(timeStamp, target, true);
    Sleep(1000);

    // ===== 释放机器人控制权 =====
    if (eriManager.releaseControlOverRobot())
        std::cout << "控制权释放成功。" << std::endl;
    else
        std::cerr << "释放控制权失败。" << std::endl;
}

// ---------------------- 主程序入口 ----------------------
```

```
int main() {
    // 设置机器人连接参数
    std::string robotIp = "172.31.16.204";
    uint16_t cmdPort = 61210;
    uint16_t servoPort = 61211;
    uint16_t statusPort = 61212;

    if (!eriManager.setCommunicationParam(robotIp.c_str(), cmdPort, servoPort,
statusPort)) {
        std::cerr << "机器人通信参数设置失败!" << std::endl;
        return -1;
    }
    Sleep(50); // 等待通信稳定

    // 用户交互命令行界面
    char input = '0';
    while (true) {
        std::cout << "输入指令: (3: 开始画圆, q: 退出): ";
        std::cin >> input;

        if (input == 'q') break;
        if (input == '3') processRealtimeData();
    }

    return 0;
}
```

8.2 示教程序与外部实时数据交替运行

在进行该操作前，请确保示教器处于再现模式，且示教器程序可正常运行。

整体流程：



示例代码如下：

```
  Start：
MovJ(G:P0)
MovL(G:P1)
MovL(G:P2)
SetERIProgID(1) //设置执行外部程序ID
EnableERIControl() //控制权切换到外部,程序卡在此处，直到外部程序执行完毕,控制权收回
  End;
```

8.3 通过 SIMIO 控制机器人纠偏的使用样例

ERI 和外部应用通过虚拟 IO 接口交互数据，使用到的虚拟接口有 SIMAI(100-120)，SIMDO（100-120）。

其中系统默认使用的端口为：

SIMDO100：新数据标志位，1 表示有新数据，0 表示数据已被处理

SIMAI100：基于纠偏坐标系 x 轴纠偏量

SIMAI101：基于纠偏坐标系 y 轴纠偏量

SIMAI102：基于纠偏坐标系 z 轴纠偏量

SIMAI103：基于纠偏坐标系 x 轴旋转的固定角纠偏量 a

SIMAI104：基于纠偏坐标系 y 轴旋转的固定角纠偏量 b

SIMAI105：基于纠偏坐标系 z 轴旋转的固定角纠偏量 c

纠偏功能由纠偏指令搭配外部接口一起使用样例：

- **在示教器端编写纠偏指令**

先在示教器端编写如下程序指令：

```
Start:
EnableERICorrect(“ON”) //开启纠偏后面运动段将会根据纠偏数据进行纠偏
MovJ(L:P0)
MovJ(L;P1)
EnableERICorrect(“OFF”)//关闭纠偏后续运动不会再执行纠偏
Movj(L:P2)
Goto Start
End;
```

- **在客户应用程序中设置 SIMIO**

客户应用程序通过 EIP /PN 总线通讯与机器人进行数据交互，周期性在 SIMIO 对应的映射表中写入纠偏数据，实时控制机器人的纠偏运动。

```
while(true)
```

```
{
    Int do100Val = getSimDO(100);
    If(0 == do100Val) //可发送纠偏数据
    {
        Int x,y,z,a,b,c;
        setSimAI(100,x);
        setSimAI(101,y);
        setSimAI(102,z);
        setSimAI(103,a);
        setSimAI(104,b);
        setSimAI(105,c);
        setSimDO(100,1);//通知 ERI 服务，纠偏数据已发送完毕
    }
    Else
    {
        Sleep(1);//等待 1 毫秒，具体根据实际情况可调整
    }
}
```

- **运行示教器程序与客户应用程序**

示教程序运行到纠偏指令开启纠偏后，机器人走P0与P1时会同步叠加客户应用程序写入的偏移位置，实现实时纠偏运动。

附录一：错误列表

错误号	错误信息	处理方法
88817	实时运动中不允许释放控制权	等待实时运动结束
88818	未获取控制权不允许 movj 运动	先获取控制前
88819	实时运动中不允许 movj 运动	等待实时运动结束
88820	机器人状态获取失败	未注册状态回调函数
88821	startServo 失败	获取控制权后再执行