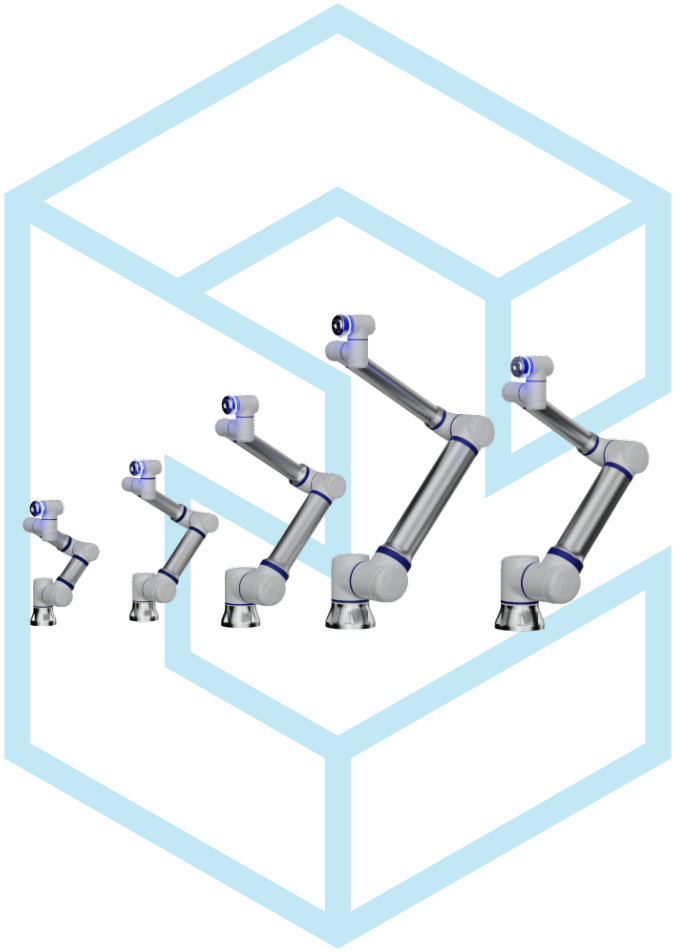
ELITE ROBOTS
艾利特机器人

ELITE ROBOTS
二次开发手册



C++SDK 快速使用手册

艾利特智能机器人股份有限公司

2026-01-19

版本：Ver2.15.0

使用前请仔细阅读本手册

此版本手册同控制软件版本保持一致，产品版本信息及所需软件版本请参阅第 1 章，使用前请仔细核对实际产品版本信息，确保一致和符合版本限制。

本手册会定期进行检查和修正，更新后的内容将出现在新版本中。本手册中的内容或信息如有变更, 恕不另行通知。

艾利特智能机器人股份有限公司对本手册中可能出现的任何错误概不负责。

艾利特智能机器人股份有限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概不负责。

安装、使用产品前，请阅读本手册。

请保管好本手册，以便可以随时阅读和参考。

本手册图片仅供参考，请以收到的实物为准。

目录

第 1 章 概述

艾利特机器人 SDK 接口库是艾利特机器人为 CS 和 ES 系列机械臂开发的一套 C++ 库，借助该接口库，开发者可以基于 C++ 实现驱动程序，从而充分利用艾利特机器人 CS 和 ES 系列机械臂的多功能性来开发外部应用程序。

1.1 支持的平台与系统要求

1.1.1 支持的操作系统

艾利特机器人 SDK 同时支持 Windows 与 Linux 操作系统，配置方法参见第 2 章。

1.1.2 支持的机器人软件版本

艾利特机器人 SDK 版本与适配的机器人软件版本的对应关系如下：

表 1-1. SDK 版本与机器人软件版本对应关系

SDK 版本	机器人软件版本
v1.1.0	≥ 2.10.x
v1.2.0	≥ 2.13.4 或 ≥ 2.14.2
V1.3.0	≥ 2.15.0

如果您的 SDK 版本不是上述两种版本，请访问github 仓库选择对应的 SDK 版本的 tag，查阅 README 文件了解所需的机器人软件版本。

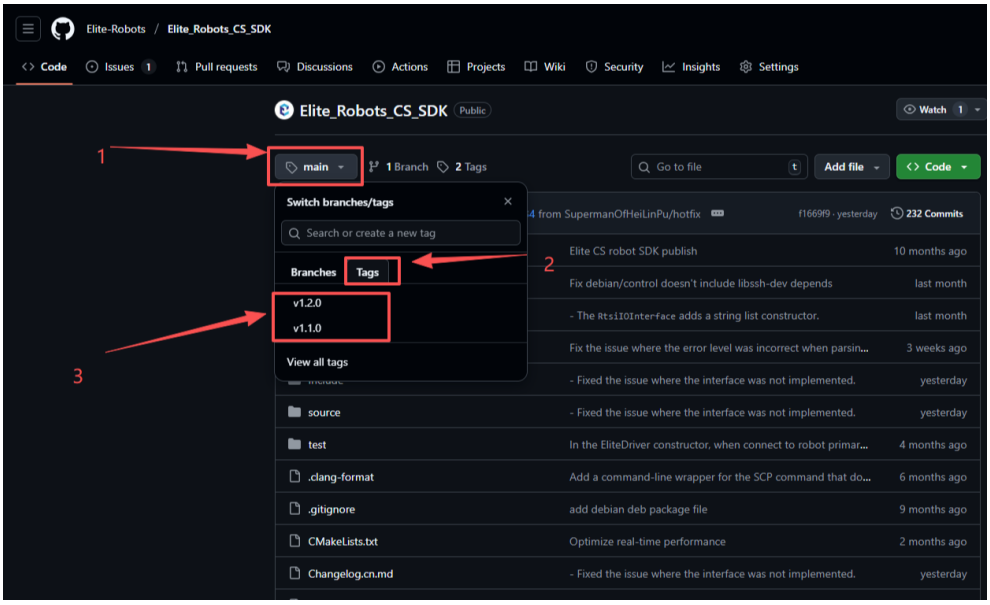


图 1-1: GitHub 标签选择

提示

如无法访问 github 网站，可前往 gitee 镜像仓库 查看版本 tag。

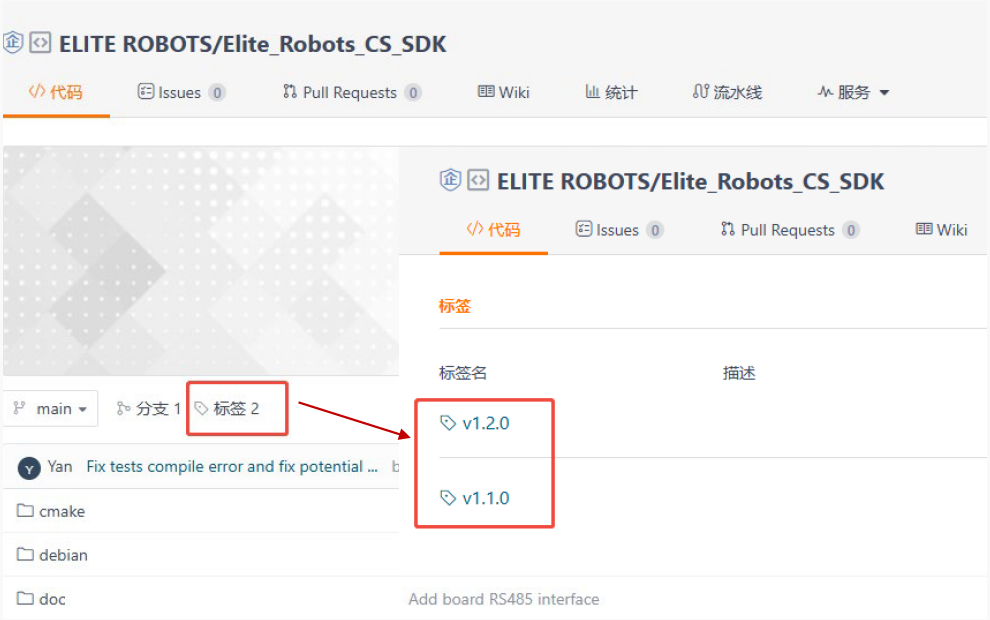


图 1-2: Gitee 镜像仓库标签选择

1.1.3 支持的 C++ 标准

艾利特机器人 SDK 包最低支持 C++14 标准。

提示

使用 C++14 标准时，SDK 依赖 boost 库（headonly 模式）；而使用 C++17 及以上标准则无需 boost 库。

第 2 章 开发环境部署

2.1 Linux 安装及配置

2.1.1 Ubuntu 安装 SDK

对于不同的 Ubuntu 发行版，推荐采用以下安装方式：

表 2-1. Ubuntu 发行版安装方式推荐

Ubuntu 发行版本	推荐安装方式
20.04、22.04、24.04	使用 apt 安装
16.04、18.04	使用预编译包安装

1. 使用 apt 安装

艾利特机器人 SDK 包已发布至 ppa 源，支持 x86_64 以及 arm64 架构。如果您的 Ubuntu 发行版为 20.04 至 24.04 版本，可使用以下指令直接安装：

```
sudo add-apt-repository ppa:elite-robots/cs-robot-series-sdk

sudo apt update

sudo apt install elite-cs-series-sdk
```

如需卸载，推荐使用以下指令：

```
sudo dpkg --remove elite-cs-series-sdk
```

如需升级 SDK，建议先卸载，再执行 **sudo apt install elite-cs-series-sdk** 命令重新安装。

提示

通过 apt 安装的 SDK 使用 C++17 编译标准。

2. 使用预编译包安装

如果您的 Ubuntu 发行版为 Ubuntu 16.04 或 18.04 版本，点击 预编译包下载链接 下载 X86_64 的预编译版本。下载完成后使用以下指令进行解压：

```
tar -zxvf <package.name.tar.gz>
```

使用以下指令将库拷贝到系统路径中（建议逐条执行，注意替换为解压后的实际路径）：

```
cd <package.name.directory/>

sudo cp -r cmake/elite-cs-series-sdk /usr/local/lib/cmake/

sudo cp -r include/Elite /usr/local/include/

sudo cp lib/libelite-cs-series-sdk.* /usr/local/lib/

sudo mkdir /usr/local/share/Elite/

sudo cp resource/external_control.script /usr/local/share/Elite/

sudo ldconfig
```

使用以下指令安装依赖库：

```
sudo apt install libboost-dev libssh-dev
```

提示

使用预编译包安装的 SDK 的 C++ 编译标准为 C++14。

可能遇到的问题：

执行拷贝指令时，如果出现**/usr/local/**目录下 **Not a directory** 类似的报错，直接创建一个对应的目录即可，例如：出现 **cp: cannot create regular file '/usr/local/lib/cmake/': Not a directory** 报错时，执行 **sudo mkdir /usr/local/lib/cmake/**创建目录。

2.1.2 其余 Linux 发行版

对于其他 Linux 发行版，目前暂不提供安装包或预编译包支持，需自行编译安装。

2.1.3 Linux 上的编译安装

本节以 Ubuntu 为例说明 Linux 上的编译安装操作。其他 Linux 发行版仅在依赖包安装和编译工具安装方式上存在差异。

执行以下指令安装编译工具：

```
sudo apt update

sudo apt install build-essential cmake git
```

执行以下指令安装依赖：

```
sudo apt install libboost-dev libssh-dev
```

执行以下指令下载代码（建议逐条执行）：

```
git clone https://github.com/Elite-Robots/Elite_Robots_CS_SDK.git

cd Elite_Robots_CS_SDK/

git checkout vX.Y.Z # 替换为您需要的版本例如 v1.2.0
```

提示

如无法访问 github 网站，可使用以下指令从 gitee 镜像仓库下载代码：

```
git clone https://gitee.com/elite-robots/Elite_Robots_CS_SDK.git
```

执行以下指令编译并安装（建议逐条执行，执行过程中无报错即视为成功）：

```
mkdir build && cd build

cmake -DELITE_COMPILE_EXAMPLES=TRUE ..

make -j4

sudo make install

sudo ldconfig
```

可能遇到的问题：

1. 执行 **cmake -DELITE_COMPILE_EXAMPLES=TRUE ..** 出现以下类似报错：

```
CMake Error at CMakeLists.txt:1 (cmake_minimum_required):
    CMake 3.16 or higher is required.  You are running version 3.xx
```

此问题由 cmake 版本低于要求版本引起，可尝试以下解决方法：

- 执行 **cmake –version** 查看当前 cmake 版本；
- 修改 SDK 根目录下 CMakeLists.txt 文件顶部的 **cmake_minimum_required(VERSION 3.16)** 语句，将版本号修改为您的 cmake 版本号；
- 再次执行 **cmake -DELITE_COMPILE_EXAMPLES=TRUE ..**
  如果出现其他错误，请将 CMakeLists.txt 文件复原，并升级 cmake 版本至 3.16 或更高。

2. 执行 **make -j4** 出现以下报错：

```
error: ‘variant’ in namespace ‘std’ does not name a template type
```

此问题由编译器不支持或不完全支持 C++17 标准引起，建议升级支持完整 C++17 标准的编译器。如果无法升级，请使用以下指令编译安装：

```
cmake -DELITE_COMPILE_EXAMPLES=TRUE -DCMAKE_CXX_STANDARD=14 ..

make -j4

sudo make install
```

提示

使用 C++14 标准时，SDK 依赖 headonly 模式的 boost 库。

2.1.4 Linux 中使用 SDK

创建 **dashboard_example.cpp** 文件，将以下代码复制到文件中：

```
#include <iostream>
#include <memory>
#include <string>
#include <Elite/DashboardClient.hpp>

using namespace ELITE;

int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cout << "Must provide robot IP. Example: ./dashboard_example aaa.bbb.ccc.ddd" << std::endl;
        return 1;
    }
    std::string robot_ip = argv[1];

    std::unique_ptr<DashboardClient> my_dashboard;
```

```
    my_dashboard.reset(new DashboardClient());

    if (!my_dashboard->connect(robot_ip)) {
        std::cout << "Could not connect to robot" << std::endl;
        return 1;
    } else {
        std::cout << "Connect to robot" << std::endl;
    }

    // Power on
    if (!my_dashboard->powerOn()) {
        std::cout << "Could not send Power on command" << std::endl;
        return 1;
    } else {
        std::cout << "Power on" << std::endl;
    }

    // Brake release
    if (!my_dashboard->brakeRelease()) {
        std::cout << "Could not send BrakeRelease command" << std::endl;
        return 1;
    } else {
        std::cout << "Brake release" << std::endl;
    }

    my_dashboard->disconnect();

    return 0;
}
```

使用以下指令编译此代码。

```
g++ dashboard_example.cpp -o dashboard_example -lelite-cs-series-sdk --
    std=c++17
```

如无报错，表示编译成功。如需运行编译后的程序，请参阅第 3 章。

如果您的编译器支持 C++14，需要修改命令中的 C++ 标准：

```
g++ dashboard_example.cpp -o dashboard_example -lelite-cs-series-sdk --
    std=c++14
```

如果出现以下报错：

```
1 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_create'
2 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_condattr_setclock'
3 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_sigmask'
4 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_setaffinity_np'
5 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_join'
6 //usr/local/lib/libelite-cs-series-sdk.so: undefined reference to `
    pthread_detach'
7 collect2: error: ld returned 1 exit status
```

可尝试添加-pthread 参数：

```
1 g++ dashboard_example.cpp -o dashboard_example -lelite-cs-series-sdk -
    pthread --std=c++14
```

2.2 Windows 安装及配置

2.2.1 使用预编译包安装

点击SDK 下载链接 下载 Windows X86_64 的预编译版本。下载完成后解压得到以下文件（用于配置 SDK 到 Visual Studio 工程）：

- include
  - Elite: SDK 的头文件
- lib
  - Debug：debug 版本的库文件（dll、lib）
  - Release：release 版本的库文件（dll、lib）
- resource
  - external_control.script： 控制脚本

2.2.2 Windows 下编译 SDK

1. 安装工具

1. 下载安装 Microsoft Visual Studio 编程平台软件，请访问VS 下载链接下载并安装。

提示

安装时需勾选” 使用 C++ 的桌面开发”。

图 2-1: Visual Studio 安装界面

2. 点击CMake 下载链接下载并安装 CMake。
3. 点击git 下载链接下载并安装 git。

2. 安装依赖

使用 vcpkg 作为 Windows 平台下的包管理器，在 Windows 终端中运行以下指令下载 vcpkg：

```
git clone https://github.com/microsoft/vcpkg.git

cd vcpkg

.\bootstrap-vcpkg.bat
```

若系统输出以下内容，则说明 vcpkg 已安装成功：

```
Downloading https://github.com/microsoft/vcpkg-tool/releases/download/2025-07-21/vcpkg.exe -> C:\Users\cc\vcpkg\vcpkg.exe (using IE proxy : 127.0.0.1:7890)... done.
Validating signature... done.

vcpkg package management program version 2025-07-21-d4b65a2b83ae6c3526acd1c6f3b51aff2a884533

See LICENSE.txt for license information.
Telemetry
---------
vcpkg collects usage data in order to help us improve your experience.
The data collected by Microsoft is anonymous.
You can opt-out of telemetry by re-running the bootstrap-vcpkg script with -disableMetrics,
passing --disable-metrics to vcpkg on the command line,
or by setting the VCPKG_DISABLE_METRICS environment variable.

Read more about vcpkg telemetry at docs/about/privacy.md
```

使用以下指令安装依赖库：

```
.\vcpkg install boost-asio

.\vcpkg install boost-program-options

.\vcpkg install libssh
```

如果安装成功，系统终端将输出：

```
All requested installations completed successfully in: 4.26 ms
```

执行以下指令配置到 Visual Studio：

```
.\vcpkg integrate install
```

如果配置成功，终端将输出：

```
CMake projects should use: "-DCMAKE_TOOLCHAIN_FILE=C:/Users/cc/vcpkg/
    scripts/buildsystems/vcpkg.cmake"

All MSBuild C++ projects can now #include any installed libraries.
    Linking will be handled automatically. Installing new libraries will
     make them instantly available.
```

复制输出中 **CMake projects should use** 后"=" 号后的路径，例如：

C:/Users/cc/vcpkg/scripts/buildsystems/vcpkg.cmake。

3. 下载代码并编译

执行以下指令下载代码（建议逐条执行）：

```
git clone https://github.com/Elite-Robots/Elite_Robots_CS_SDK.git

cd Elite_Robots_CS_SDK/

git checkout vX.Y.Z # 替换为您需要的版本例如 v1.2.0
```

提示

如无法访问 github 网站，可使用以下指令从 gitee 镜像仓库下载代码：

```
git clone https://gitee.com/elite-robots/Elite_Robots_CS_SDK.git
```

执行以下指令，注意将/your/path 替换为刚才复制的路径：

```
mkdir build

cd build

cmake -DCMAKE_TOOLCHAIN_FILE=/your/path -DELITE_COMPILE_EXAMPLES=TRUE ..

cmake --build . --config Release # 编译 release 版本

cmake --build . --config Debug # 编译 debug 版本
```

如果没有 error 输出，说明编译已成功。成功后请注意以下文件，后续配置 SDK 到 Visual Studio 工程将使用这些文件：

- ./Debug：debug 版本的库文件（dll、lib）
- ./Release：release 版本的库文件（dll、lib）
- ./include/Elite：SDK 的头文件
- ../source/resources/external_control.script：控制脚本

2.2.3 配置 SDK 到 Visual Studio 工程

1. 打开 Visual Studio 软件，选择「创建新项目」。

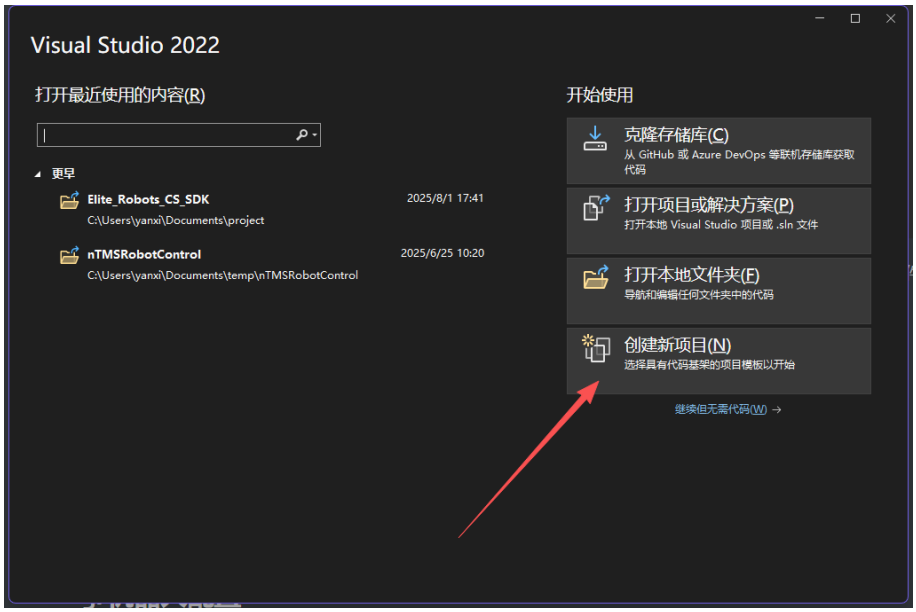


图 2-2: Visual Studio 新建项目

2. 选择语言为「C++」，选择「控制台应用」，最后点击「下一步」。

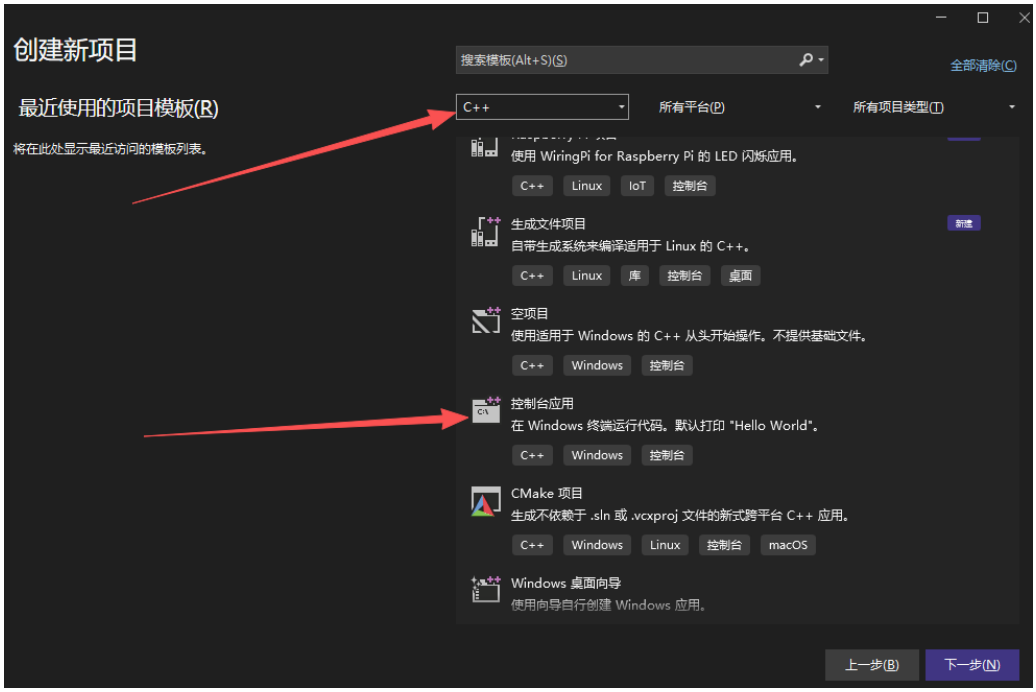


图 2-3: 选择项目模板

3. 设置项目名称、保存路径等，点击「创建」。

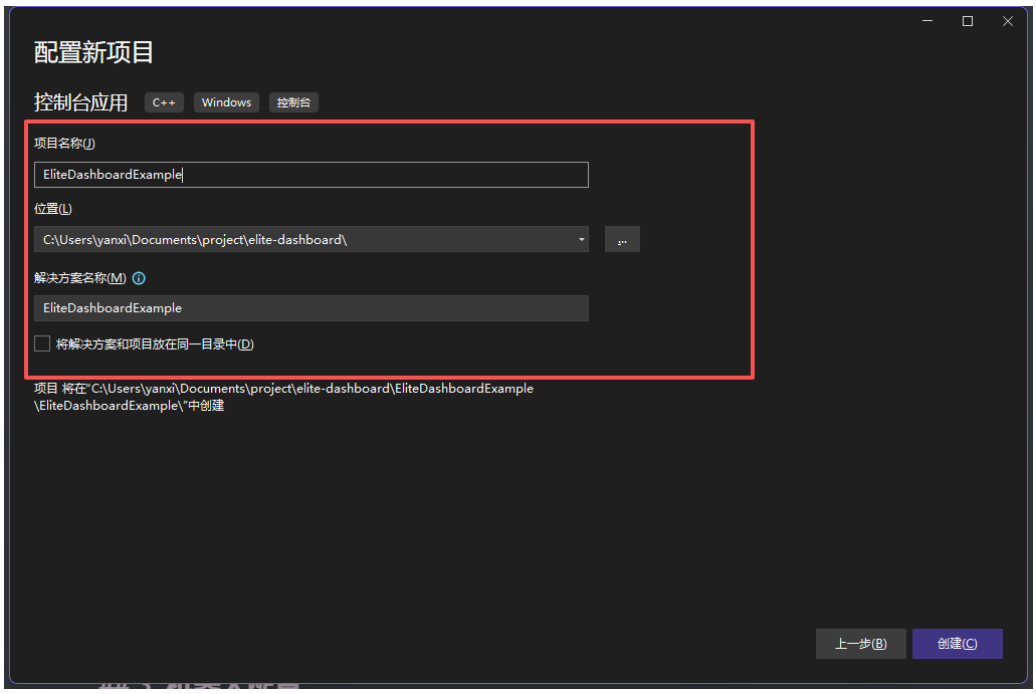


图 2-4: 项目配置

4. 拷贝库文件到项目中
   - 找到项目的存储位置，在项目目录中新建一个文件夹 **elite-cs-sdk**。

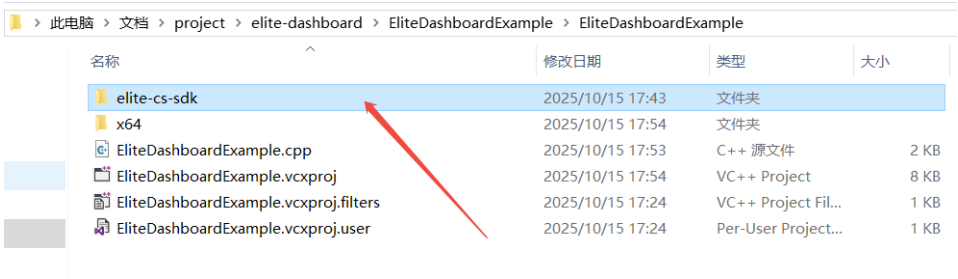


图 2-5: 创建新文件夹

- 对于下载预编译包的用户：
  - 拷贝 **lib** 中的 **Debug** 和 **Release** 文件夹到 **elite-cs-sdk** 中
  - 拷贝 **resource** 到 **elite-cs-sdk** 中
  - 拷贝 **include** 中的 **Elite** 文件夹到 **elite-cs-sdk** 中
- 对于手动编译的用户（**Elite_Robots_CS_SDK** 根目录）：
  - 拷贝 **build/Debug** 和 **build/Release** 文件夹到 **elite-cs-sdk** 中
  - 拷贝 **source/resource** 到 **elite-cs-sdk** 中
  - 拷贝 **build/include/Elite** 文件夹到 **elite-cs-sdk** 中
- 拷贝完成后，**elite-cs-sdk** 文件夹的结构如下：

```
.
├── Debug
├── Elite
├── Release
└── resources
```

5. 菜单栏选择「项目」> 「属性」。

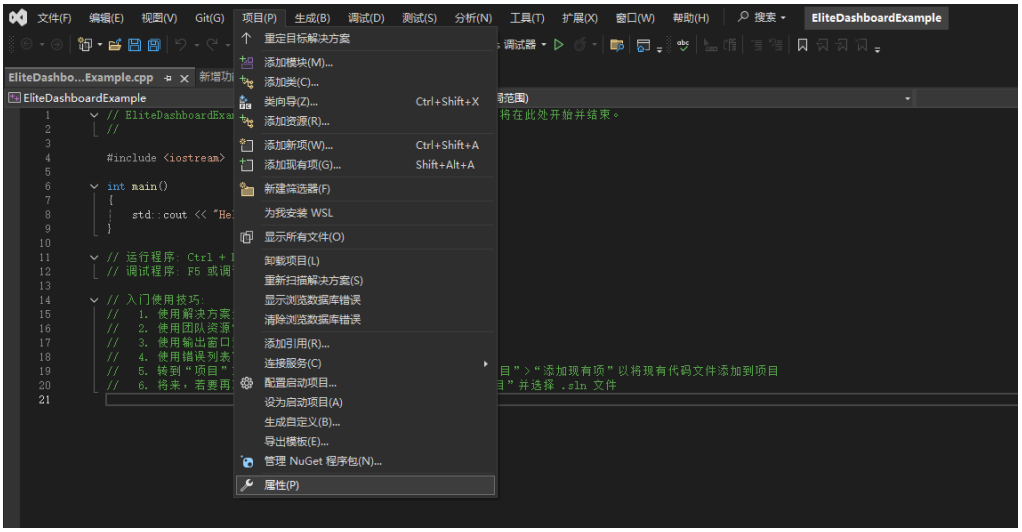


图 2-6: 项目属性设置

6. 选择「配置」为「所有配置」，选择「常规」，设置「C++ 语言标准」为 C++17。

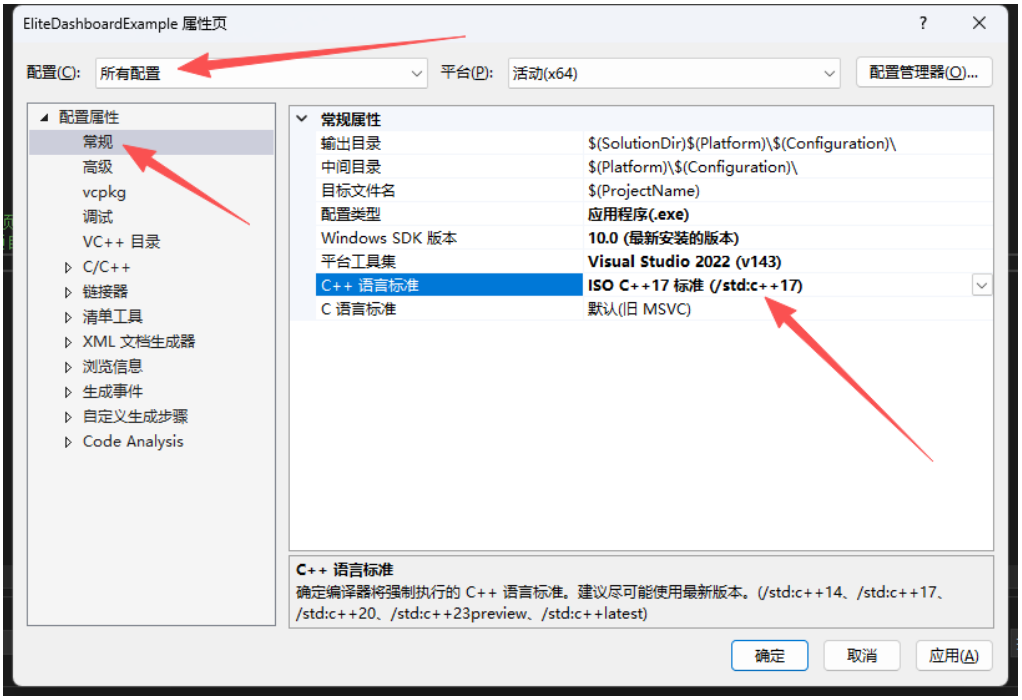


图 2-7: C++ 标准设置

7. 选择「VC++ 目录」>「外部包含目录」>「编辑」。

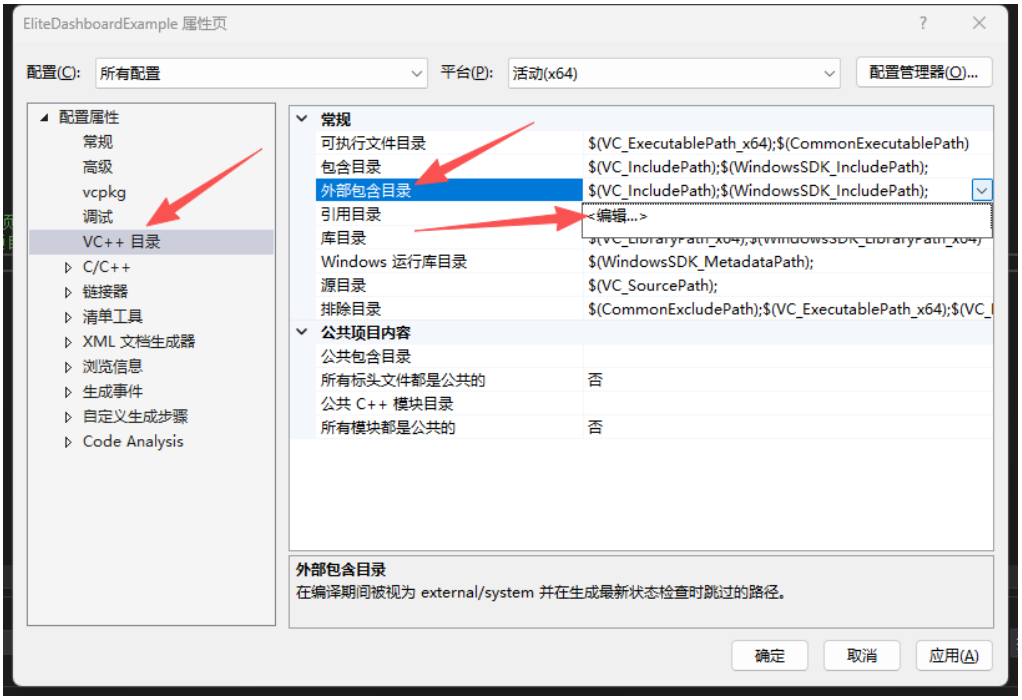


图 2-8: 设置包含目录

8. 点击空白处 >...，设置路径为第四步中的 **elite-cs-sdk** 文件夹。

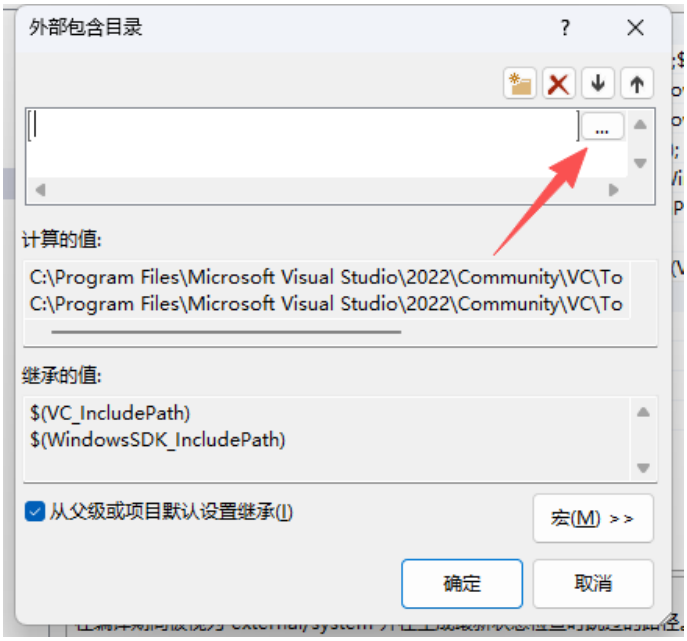


图 2-9: 设置包含目录路径

9. 调整配置为「Debug」（需先保存上述设置），选择「VC++ 目录」，选择「库目录」。

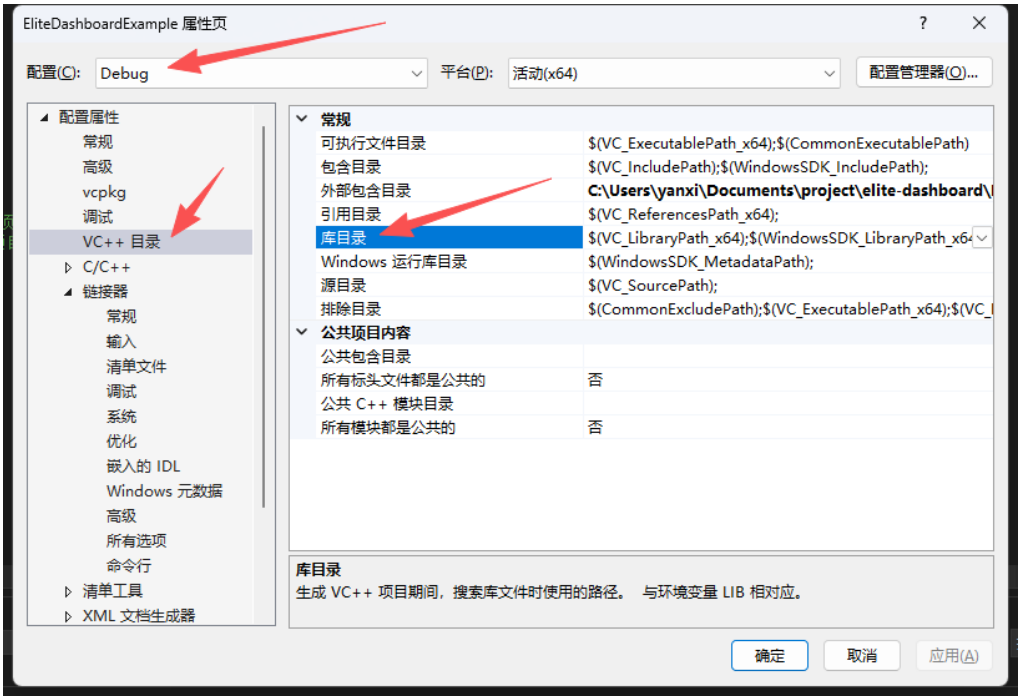


图 2-10: 设置 Debug 库目录

10. 点击空白处 >...，设置路径为第四步中的 **elite-cs-sdk/Debug** 文件夹。

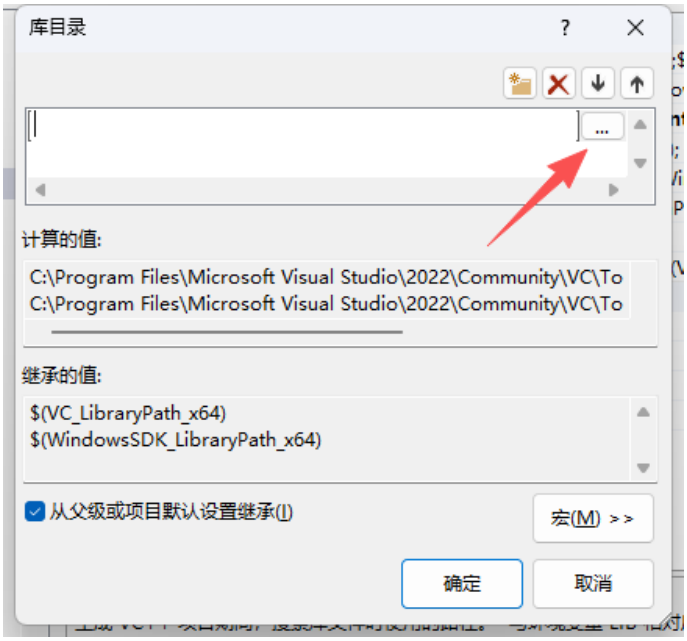


图 2-11: 设置 Debug 库目录路径

11. 调整配置为「Release」（需先保存上述设置），选择「VC++ 目录」，选择「库目录」。

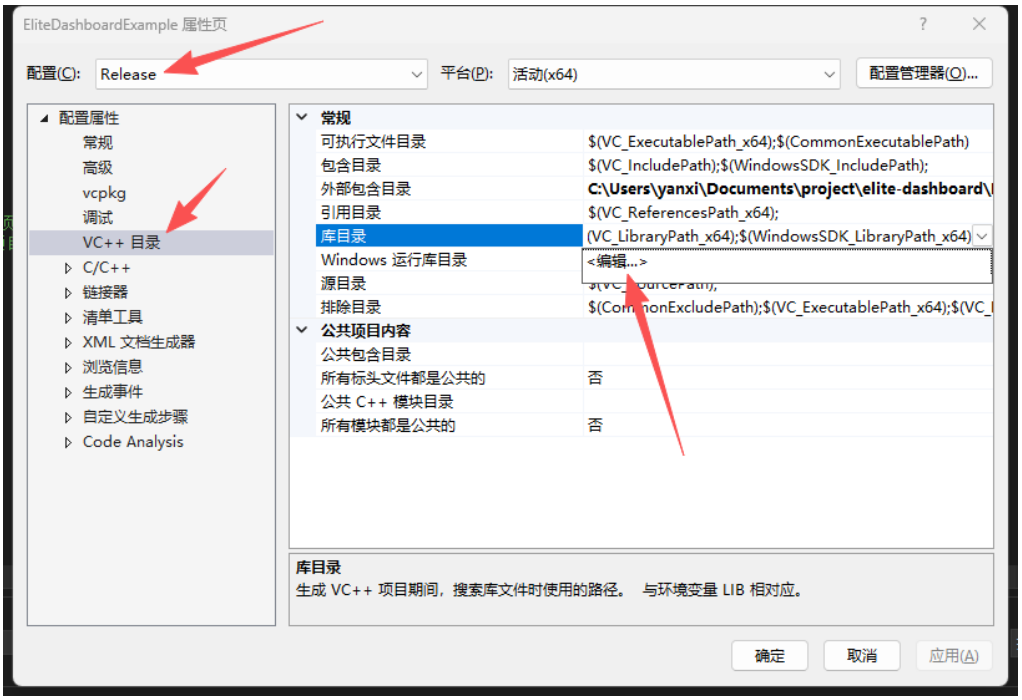


图 2-12: 设置 Release 库目录

12. 设置路径为第四步中的 **elite-cs-sdk/Release** 文件夹。

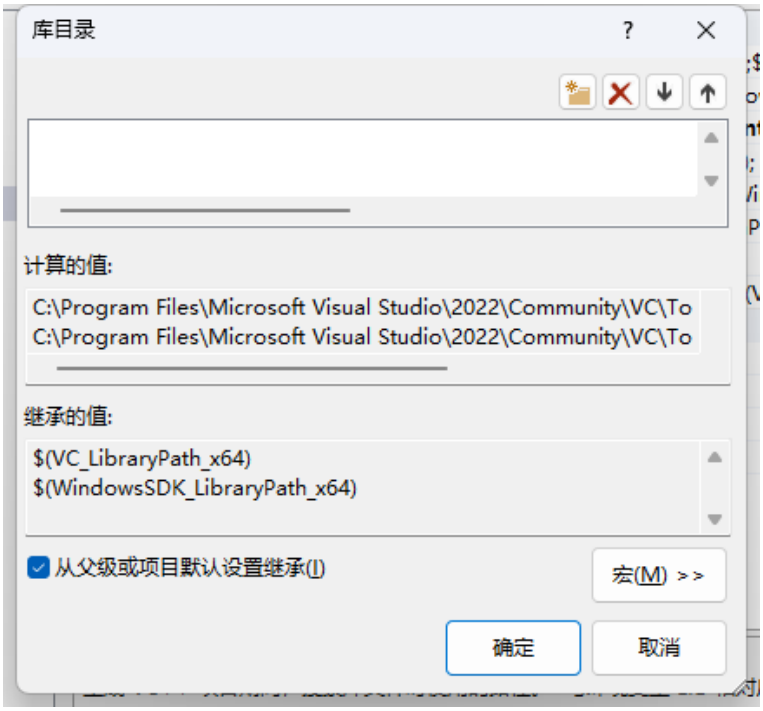


图 2-13: 设置 Release 库目录路径

13. 调整配置为「所有配置」（需先保存上述设置），选择「链接器」–> 「输入」–> 「附加依赖项」–> 「编辑」。

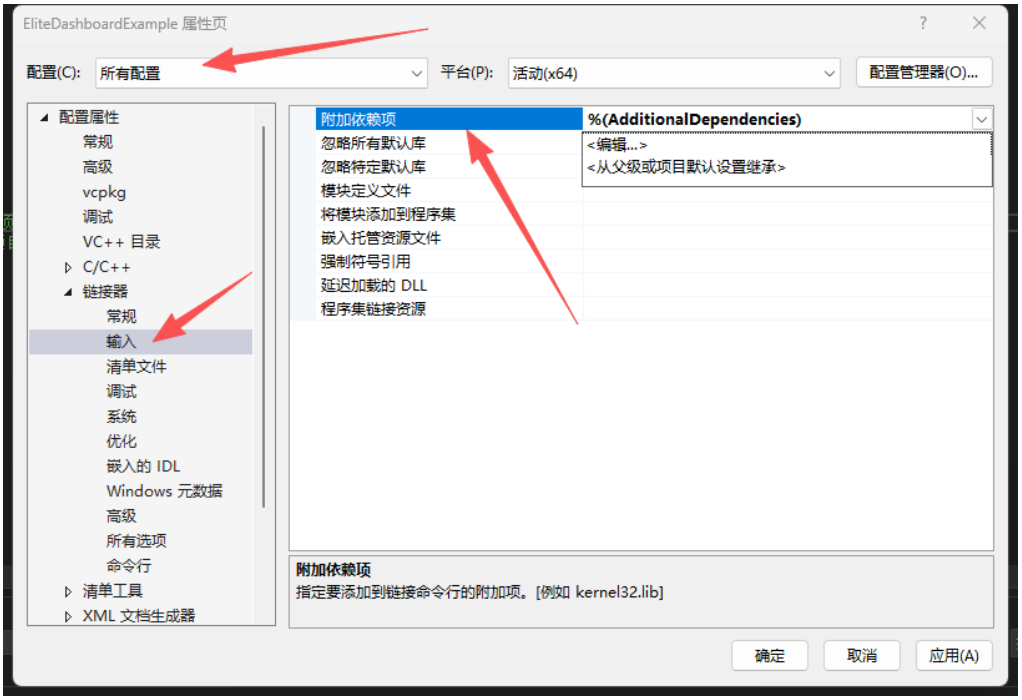


图 2-14: 设置附加依赖项

14. 输入 **elite-cs-series-sdk.lib**

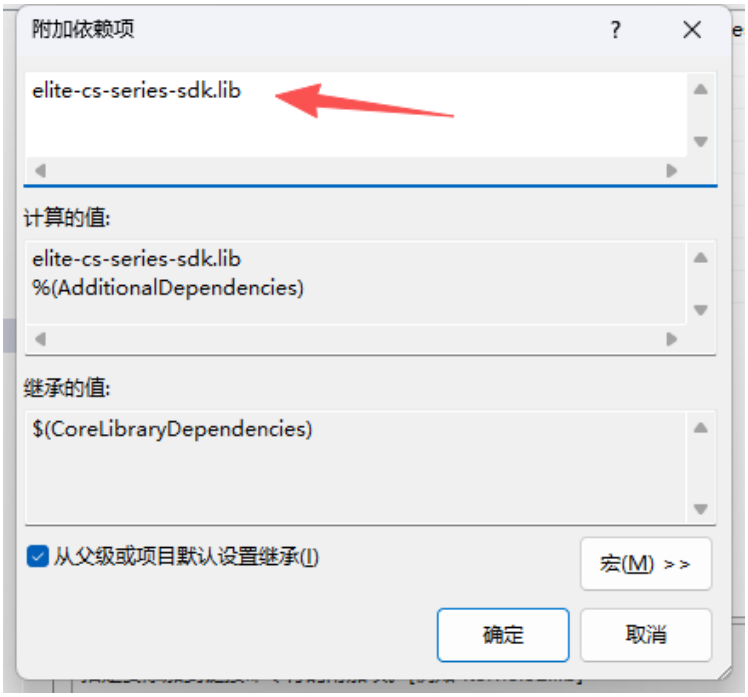


图 2-15: 输入库文件名

15. 删除 VS 中自动生成的代码，复制以下代码，粘贴到 VS 中：

```
#include <iostream>
#include <memory>
#include <string>
#include <Elite/DashboardClient.hpp>

using namespace ELITE;

int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cout << "Must provide robot IP. Example: ./dashboard_example aaa.bbb.ccc.ddd" << std::endl;
        return 1;
    }
    std::string robot_ip = argv[1];

    std::unique_ptr<DashboardClient> my_dashboard;
    my_dashboard.reset(new DashboardClient());

    if (!my_dashboard->connect(robot_ip)) {
        std::cout << "Could not connect to robot" << std::endl;
        return 1;
    } else {
        std::cout << "Connect to robot" << std::endl;
    }
```

```
24
25      // Power on
26      if (!my_dashboard->powerOn()) {
27          std::cout << "Could not send Power on command" << std::endl
        ;
28          return 1;
29      } else {
30          std::cout << "Power on" << std::endl;
31      }
32
33      // Brake release
34      if (!my_dashboard->brakeRelease()) {
35          std::cout << "Could not send BrakeRelease command" << std::
        endl;
36          return 1;
37      } else {
38          std::cout << "Brake release" << std::endl;
39      }
40
41      my_dashboard->disconnect();
42
43      return 0;
44  }
```

16. 菜单栏选择「生成」>「生成解决方案」，编译成功将不会报错。

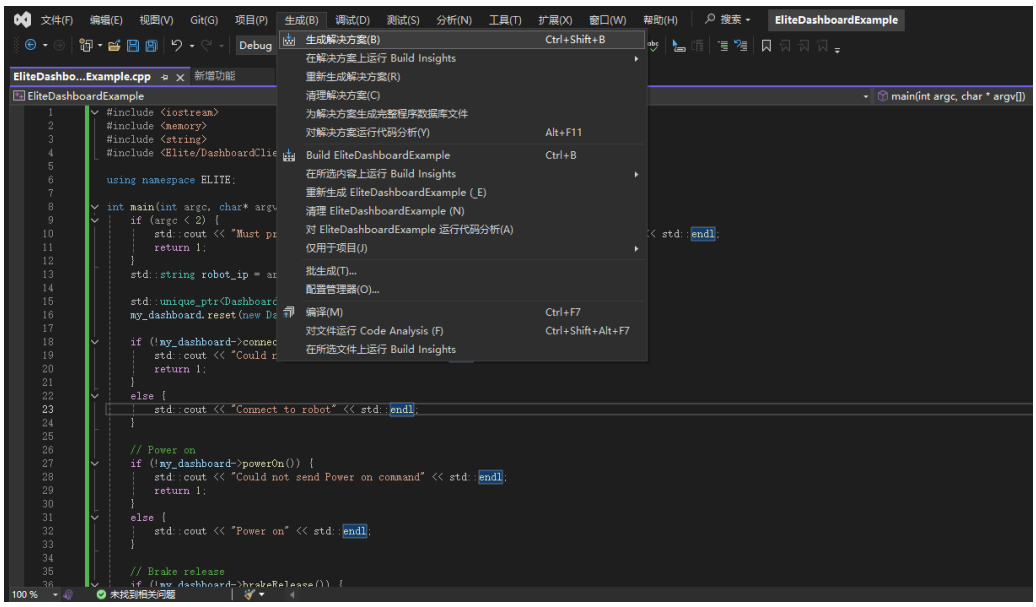


图 2-16: 编译解决方案

17. 编译完成后，解决方案目录下会生成一个 **x64** 文件夹。在该文件夹的 **Debug** 子目录中，

将生成一个.exe 可执行程序。

请将 **elite-cs-sdk\Debug** 目录下的所有文件拷贝到程序所在目录中(即 **x64\Debug**)。

如需运行编译生成的程序，请参考 第 3 章 的说明。

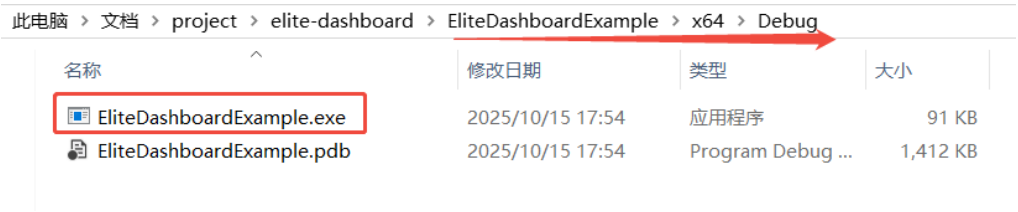


图 2-17: 拷贝文件目录

提示

如果您的编译模式(图中位置)是 Release，则路径中的 Debug 需替换为 Release。

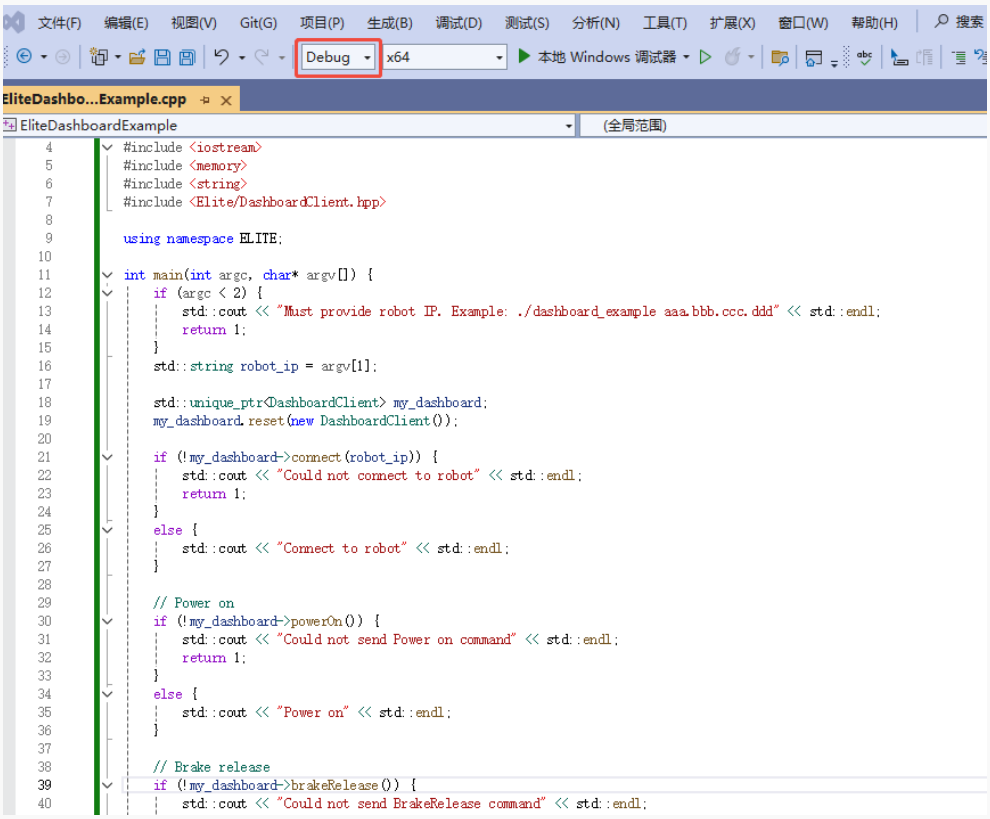


图 2-18: 更换编译模式

第 3 章 机器人配置

3.1 网络配置

3.1.1 CS 系列标准控制柜

如果您使用的是 CS 系列标准控制柜，需要将 FB1 和 FB2 两个网口都连接到局域网中，并设置好 IP 地址。

为什么要连接两个网口?

因为 CS 系列标准控制柜采用双处理器的分布式架构：

- FB1 网口对应一个处理器，负责运行机器人的 dashboard 功能；
- FB2 网口对应另一个处理器，负责运行与 SDK 进行通讯的控制软件。

因此需要同时连接两个网口。

对于机器人软件 2.14.3 之后的版本，使用 SDK 的 headless 模式（参见第 5.2 节）可以只连接 FB2 网口。

3.1.2 CS 系列 B1 控制柜

如果您使用的是 CS 系列 B1 控制柜，只需连接一个网口。

3.2 设置远程控制模式

1. 点击示教器右上角的 Elite 图标 > 「设置」。

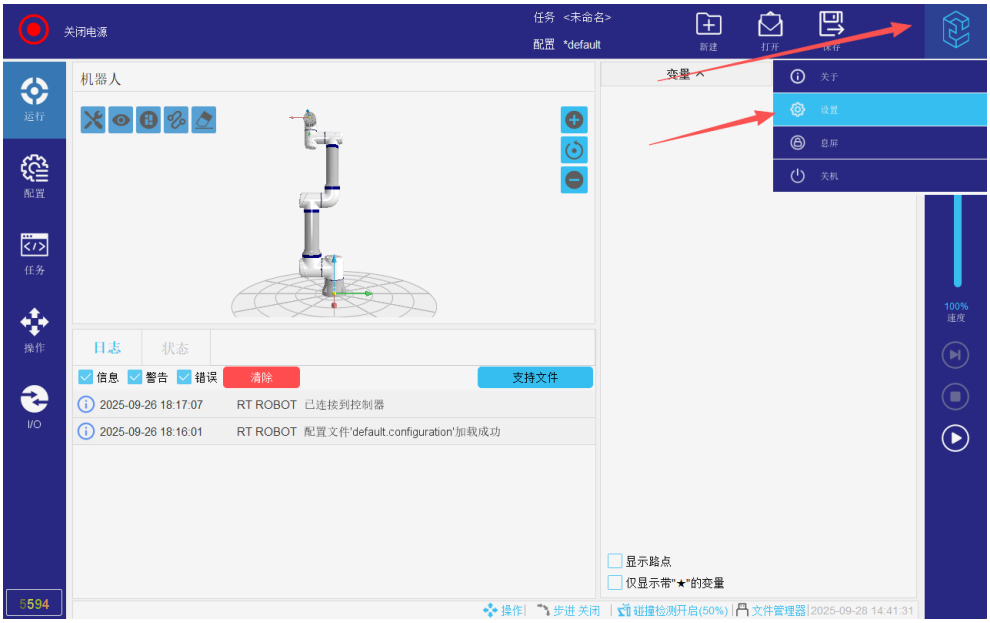


图 3-1: 示教器设置

2. 启用/禁用「远程控制模式」，点击左下角退出：

图 3-2: 远程控制模式设置

如果选择了启用远程控制模式，界面右侧栏会出现「本地控制」选项，请切换到「远程控制」模式。

图 3-3: 远程控制选项

图 3-4: 远程控制界面

提示

CS 系列机器人如果远程控制模式是禁用状态，则会处于” 混合模式”，即既可以接收外部的控制，也可以接收示教器本地的控制，因此在上面步骤中，远程控制模式可以设为禁用。

3.3 测试网络连接

有两种方式可以测试 SDK 是否能与机器人通讯：

1. 使用机器人脚本
2. 使用 bash 指令测试

推荐使用方法 1（机器人脚本）。如果该方法能够成功执行，说明机器人配置基本没有问题，不会出现后续控制运动时的连接问题。

方法 2（Bash 指令）可用于初步排查网络是否畅通，但即使指令测试成功，也可能因路由等原因导致运动控制失败。因此，若使用方法 2 无法正常控制机器人运动，请使用方法 1。

3.3.1 使用机器人脚本（推荐）

1. **获取 IP 地址**
   在要运行 SDK 的操作系统中执行以下指令：
   - Linux：
     ```
     1 ifconfig
     ```
   - Windows：
     ```
     1 ipconfig
     ```

   记录下与机器人相连网口的 IP，记为 Local IP。

   提示

   可能遇到的问题：
   Linux 中执行 **ifconfig** 时可能会出现找不到命令的提示，此时需要安装一下该指令，例如在 Ubuntu 中使用 **sudo apt install net-tools** 指令安装。

2. **在示教器上创建测试脚本**
   - 点击示教器左侧栏的「任务」。

图 3-5: 选择任务

- 点击「占位节点」>「高级」>「脚本」。

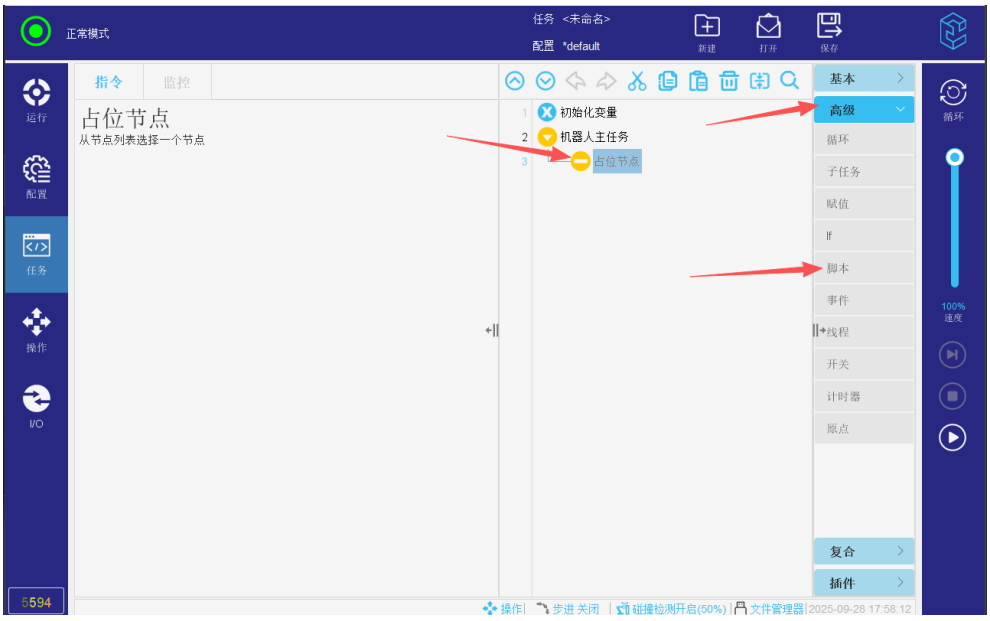


图 3-6: 选择脚本

- 点击左侧下拉框，选择「文件」，点击「编辑」。

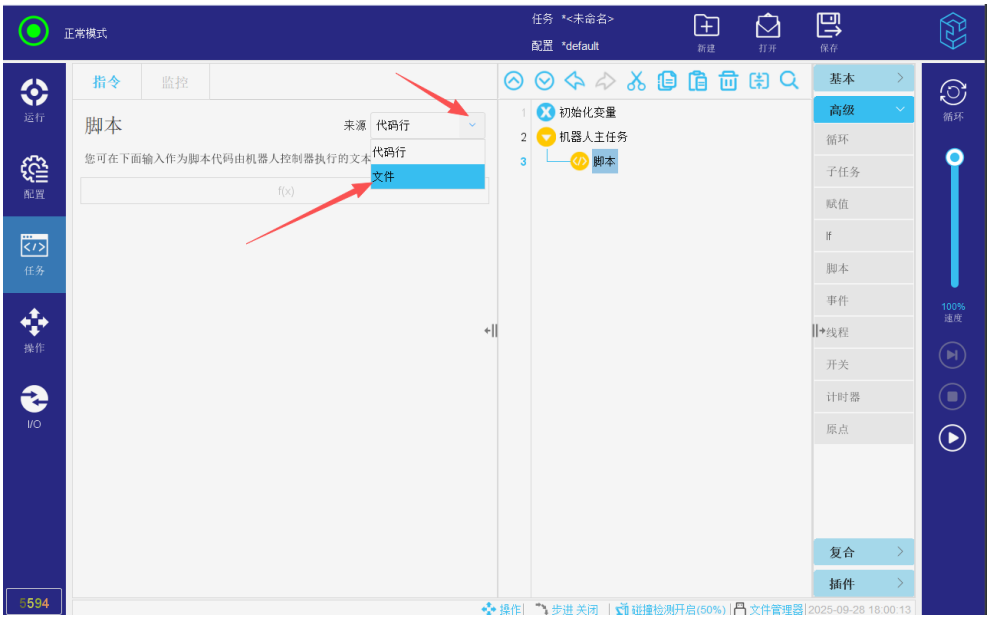


图 3-7: 编辑文件

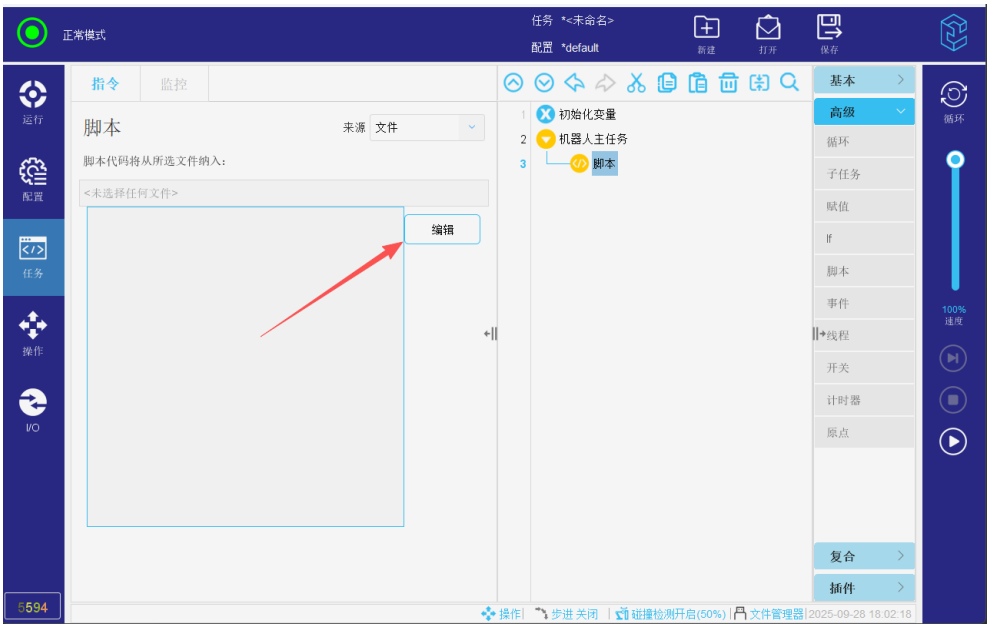


图 3-8: 编辑脚本

- 将以下内容写入文本编辑器中（注意将 Your.Local.IP.Addr 替换为刚才记录的 Local IP 地址）：

```
1 socket_open("Your.Local.IP.Addr", 50001)
2 socket_send_string("Hello SDK\n")
3 sleep(1)
```

- 点击「保存」，并设置文件名。

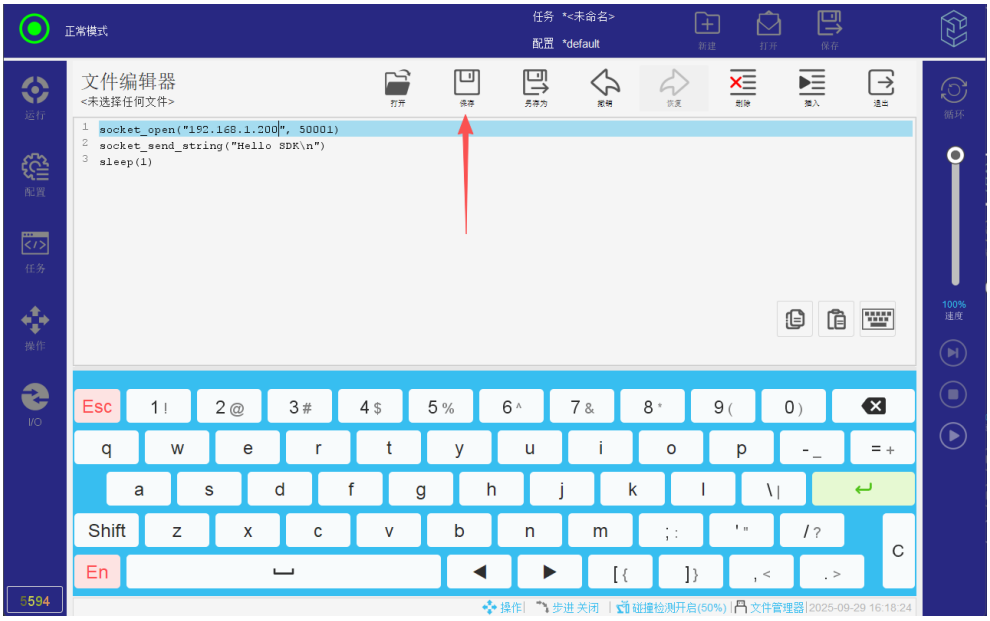


图 3-9: 保存脚本

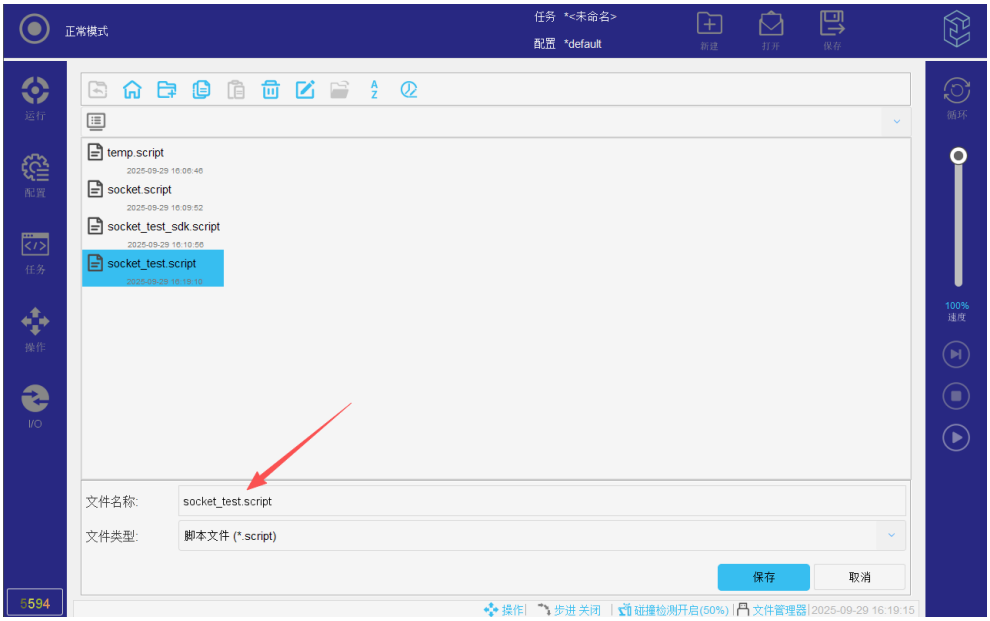


图 3-10: 设置脚本名称

3. 创建 TCP 测试服务器

在要运行 SDK 的操作系统中创建一个端口为 50001 的 TCP 服务。

将以下 python 脚本内容复制并保存为 sdk_connection_test.py 文件，然后执行 **python3 sdk_connection_test.py** 运行。

```
1 import socket
2
3 def start_tcp_server(host='localhost', port=50001):
4     server_socket = socket.socket(socket.AF_INET, socket.
  SOCK_STREAM)
```

```
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    try:
        # 设置accept超时（1秒）
        server_socket.settimeout(1.0)

        server_socket.bind((host, port))
        server_socket.listen(5)
        print(f"TCP服务器启动在 {host}:{port}，等待客户端连接...")

        while True:
            try:
                # 接受客户端连接（现在会超时，让出控制权）
                client_socket, client_address = server_socket.accept()
                print(f"客户端 {client_address} 已连接")

                try:
                    while True:
                        data = client_socket.recv(1024)
                        if not data:
                            print(f"客户端 {client_address} 已断开连接")
                            break

                        received_string = data.decode('utf-8')
                        print(f"来自 {client_address} 的数据: {received_string}")

                except ConnectionResetError:
                    print(f"客户端 {client_address} 异常断开连接")
                except Exception as e:
                    print(f"处理客户端 {client_address} 时发生错误: {e}")
                finally:
                    client_socket.close()

            except socket.timeout:
                # 超时是正常的，继续循环
                continue

    except KeyboardInterrupt:
        print("\n服务器被用户中断")
    except Exception as e:
```

```
        print(f"服务器发生错误: {e}")
    finally:
        server_socket.close()
        print("服务器已关闭")

if __name__ == "__main__":
    start_tcp_server()
```

4. **运行测试**

   点击示教器上的任务运行按钮，如果一切正常，会看到 TCP 服务器接收到”Hello SDK”的一行字符串。

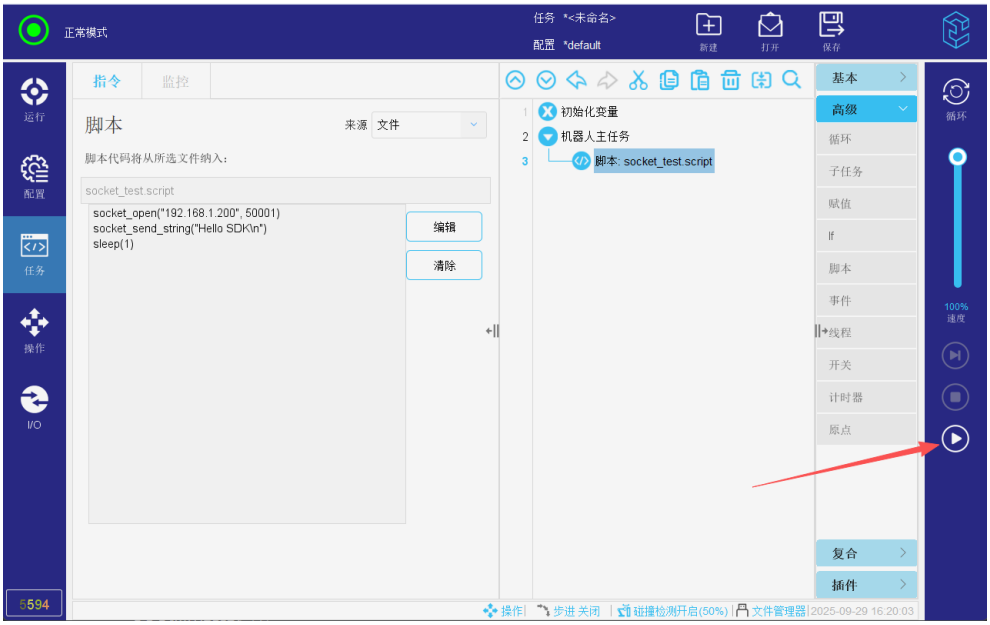


图 3-11: 运行测试任务

可能遇到的问题:

1. 如果在创建 TCP 服务时失败，可能是系统中其他程序占用了 50001 端口。如果需要更换端口，机器人脚本中的端口也要相应修改。

2. 示教器提示：”socket_send_string”：未连接服务端”socket_0”。出现这个异常的原因是，机器人无法连接 SDK 的 TCP 服务端口。可能的原因包括：
   - 如果使用了虚拟机，需要将虚拟机的网卡设置为桥接模式，并桥接到与机器人相连的网卡上，然后设置好 IP。
   - 如果您使用的是 CS 系列标准控制柜，可能没有连接 FB2 网口。
   - 局域网设置可能关闭了连接，建议将机器人与运行 SDK 的电脑直接连接。如果您使用的是 CS 系列标准控制柜，可以使用交换机将运行 SDK 的电脑、控制柜 FB1、控制柜 FB2 连接在一起。
   - IP 设置不正确，运行 SDK 的电脑 IP 与控制柜网口 IP 不在同一网段。

- 电脑开启了 VPN，建议关闭。
- 服务器没有启动成功，系统中某些应用占用了 50001 端口，请尝试更换端口（注意服务器端口和机器人脚本中的 **socket_open** 指令端口需要同时修改）。
- 服务器被关闭，可能在接收到“Hello SDK”字符串后您关闭了服务器，但此时机器人仍在循环执行脚本。
- 如上述情况均确认无误但仍无法连接，请重启控制柜再重试。

3.3.2 使用 bash 指令

1. 获取 IP 地址

在要运行 SDK 的操作系统中输入以下指令：

- Linux：

```
ifconfig
```

- Windows：

```
ipconfig
```

记录下与机器人相连网口的 IP，记为 Local IP。

提示

可能遇到的问题：
Linux 中执行 **ifconfig** 时可能会出现找不到命令的提示，此时需要安装一下该指令，例如在 Ubuntu 中使用 **sudo apt install net-tools** 指令安装。

2. 测试机器人到 SDK 的连接

在要运行 SDK 的操作系统中运行以下指令，使用 ssh 登录到机器人的操作系统里（注意替换 **aaa.bbb.ccc.ddd** 为机器人 IP，如果您使用的是 CS 系列标准控制柜，则为 FB2 的 IP），密码为 **elibot**：

```
ssh root@aaa.bbb.ccc.ddd
```

登录成功后输入以下指令，检查机器人是否能连接到 SDK（注意替换 **aaa.bbb.ccc.ddd** 为刚才记录的 Local IP）。

```
ping aaa.bbb.ccc.ddd
```

如果连接成功，会有类似以下的输出：

```
PING 192.168.1.110 (192.168.1.110): 56 data bytes
64 bytes from 192.168.1.110: seq=0 ttl=64 time=0.373 ms
64 bytes from 192.168.1.110: seq=1 ttl=64 time=0.409 ms
64 bytes from 192.168.1.110: seq=2 ttl=64 time=0.397 ms
64 bytes from 192.168.1.110: seq=3 ttl=64 time=0.432 ms
```

如果无法连接，需要检查网络设置，例如：IP 地址是否冲突，SDK 所在系统与机器人是否在同一网段，所在的局域网环境等。

3. 测试 SDK 到机器人的连接

输入 **exit** 退出登录，返回至要运行 SDK 的操作系统中，执行以下指令，简单检查 SDK 是否能连接上机器人（注意替换 **aaa.bbb.ccc.ddd** 为机器人 IP，如果是 CS 系列标准控制柜，需要确保能够接通 FB1 和 FB2）。

```
ping aaa.bbb.ccc.ddd
```

第 4 章 SDK 使用

4.1 执行先前的编译结果

在第 2 章中我们编译了一个简单的代码，现在我们来运行那个程序。为了更好地展示运行效果，建议先关闭机器人本体的电源。

图 4-1: 关闭电源

4.1.1 Linux

在前面的步骤中，**dashboard_example.cpp** 已经被编译为 **dashboard_example** 可执行程序，使用以下指令运行这个程序（**Your.Robot.IP.Addr** 需替换为实际的机器人 IP 地址）：

```
./dashboard_example Your.Robot.IP.Addr
```

如果执行成功，您会看到机器人电源已打开并释放抱闸。

4.1.2 Windows

1. 在前面的步骤中，已经编译好了程序，在 Visual Studio 软件中，右键点击解决方案资源管理器中的项目名称，选择「属性」。

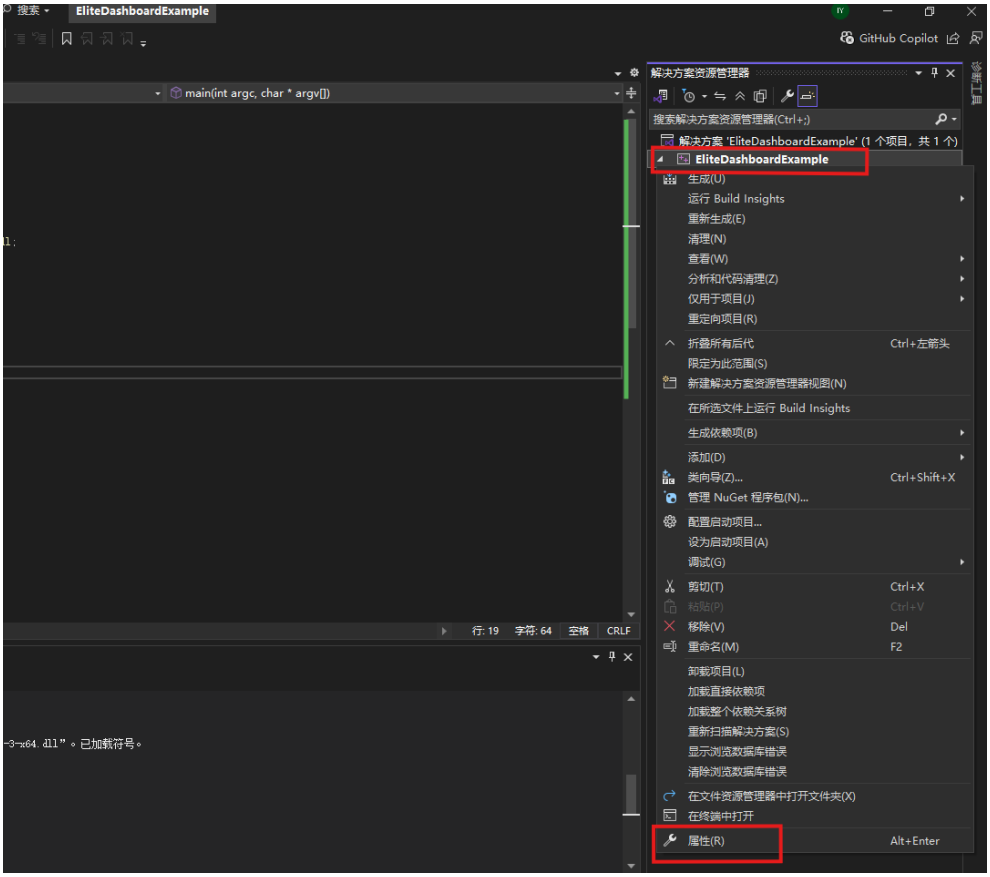


图 4-2: Visual Studio 项目属性

2. 选择「配置属性」>「调试」>「命令参数」，设置参数为机器人 IP 地址，点击「确定」。

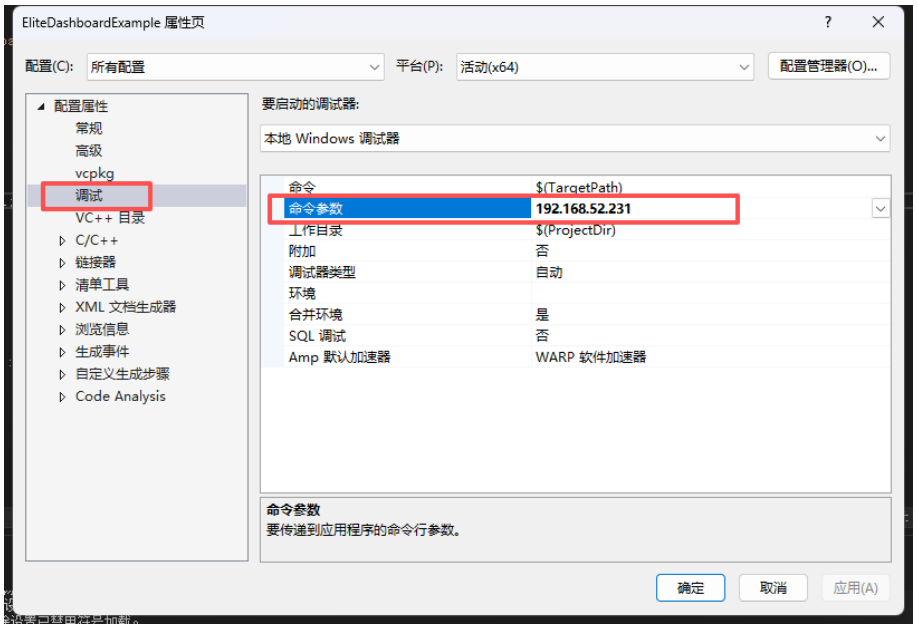


图 4-3: 设置命令参数

3. 点击「本地 Windows 调试器」运行此程序。

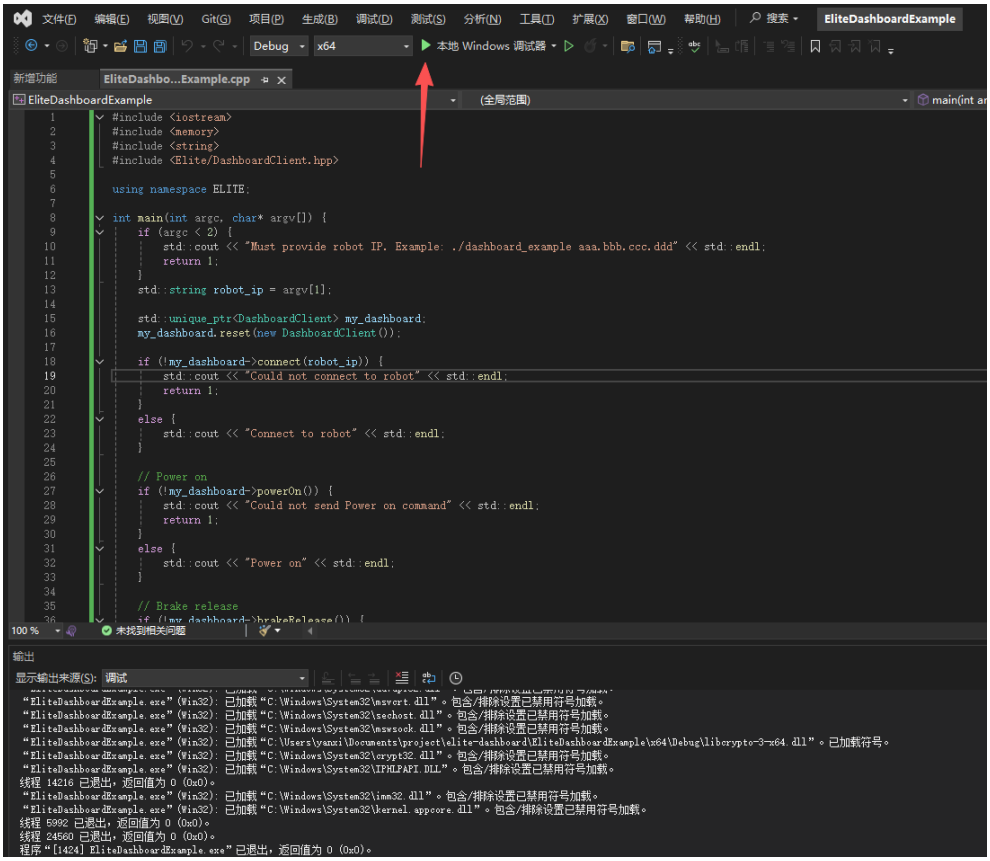


图 4-4: 运行调试程序

4. 程序成功运行后，机器人将会上电并释放抱闸。

4.1.3 简单说明

艾利特 CS 系列机器人提供了一个 Dashboard 接口，该接口提供了大量的界面操作功能，例如上下电、加载任务等。SDK 对这些功能进行了封装。刚才运行的程序就是调用了 Dashboard 接口的上电和释放抱闸两个功能。

4.2 使用 SDK 控制机器人运动

4.2.1 Linux

参考第 2.1.4 小节，创建 **move_example.cpp** 文件，将以下代码复制到文件中。

```
#include <Elite/DataType.hpp>
#include <Elite/EliteDriver.hpp>
#include <Elite/Log.hpp>
#include <Elite/DashboardClient.hpp>
#include <Elite/RtsiIOInterface.hpp>

#include <future>
#include <iostream>
#include <memory>
#include <thread>

using namespace ELITE;

class TrajectoryControl {
   private:
    std::unique_ptr<EliteDriver> driver_;

    std::unique_ptr<DashboardClient> dashboard_;
    EliteDriverConfig config_;

   public:
    TrajectoryControl(const EliteDriverConfig& config) {
        config_ = config;
        driver_ = std::make_unique<EliteDriver>(config);
        dashboard_ = std::make_unique<DashboardClient>();

        ELITE_LOG_INFO("Connecting to the dashboard");
        if (!dashboard_->connect(config.robot_ip)) {
            ELITE_LOG_FATAL("Failed to connect to the dashboard.");
            throw std::runtime_error("Failed to connect to the dashboard.");
        }
        ELITE_LOG_INFO("Successfully connected to the dashboard");
    }

    ~TrajectoryControl() {
        if (dashboard_) {
```

```
37              dashboard_->disconnect();
38          }
39          driver_->stopControl();
40      }
41
42      bool startControl() {
43          ELITE_LOG_INFO("Start powering on...");
44          if (!dashboard_->powerOn()) {
45              ELITE_LOG_FATAL("Power-on failed");
46              return false;
47          }
48          ELITE_LOG_INFO("Power-on succeeded");
49
50          ELITE_LOG_INFO("Start releasing brake...");
51          if (!dashboard_->brakeRelease()) {
52              ELITE_LOG_FATAL("Brake release failed");
53              return false;
54          }
55          ELITE_LOG_INFO("Brake released");
56
57          if (config_.headless_mode) {
58              if (!driver_->isRobotConnected()) {
59                  if (!driver_->sendExternalControlScript()) {
60                      ELITE_LOG_FATAL("Fail to send external control
    script");
61                      return false;
62                  }
63              }
64          } else {
65              if (!dashboard_->playProgram()) {
66                  ELITE_LOG_FATAL("Fail to play program");
67                  return false;
68              }
69          }
70
71          ELITE_LOG_INFO("Wait external control script run...");
72          while (!driver_->isRobotConnected()) {
73              std::this_thread::sleep_for(std::chrono::milliseconds(10));
74          }
75          ELITE_LOG_INFO("External control script is running");
76          return true;
77      }
78
79      bool moveTrajectory(const std::vector<vector6d_t>& target_points,
    float point_time, float blend_radius, bool is_cartesian) {
```

```
80          std::promise<TrajectoryMotionResult> move_done_promise;
81          driver_->setTrajectoryResultCallback([&](TrajectoryMotionResult
    result) { move_done_promise.set_value(result); });
82
83          ELITE_LOG_INFO("Trajectory motion start");
84          if(!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::START, target_points.size(), 200)) {
85              ELITE_LOG_ERROR("Failed to start trajectory motion");
86              return false;
87          }
88
89          for (const auto& joints : target_points) {
90              if (!driver_->writeTrajectoryPoint(joints, point_time,
    blend_radius, is_cartesian)) {
91                  ELITE_LOG_ERROR("Failed to write trajectory point");
92                  return false;
93              }
94              // Send NOOP command to avoid timeout.
95              if(!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::NOOP, 0, 200)) {
96                  ELITE_LOG_ERROR("Failed to send NOOP command");
97                  return false;
98              }
99          }
100
101         std::future<TrajectoryMotionResult> move_done_future =
    move_done_promise.get_future();
102         while (move_done_future.wait_for(std::chrono::milliseconds(50))
    != std::future_status::ready) {
103             // Wait for the trajectory motion to complete, and send NOOP
     command to avoid timeout.
104             if(!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::NOOP, 0, 200)) {
105                 ELITE_LOG_ERROR("Failed to send NOOP command");
106                 return false;
107             }
108         }
109         auto result = move_done_future.get();
110         ELITE_LOG_INFO("Trajectory motion completed with result: %d",
    result);
111
112         if(!driver_->writeIdle(0)) {
113             ELITE_LOG_ERROR("Failed to write idle command");
114             return false;
115         }
```

```
116
117         return result == TrajectoryMotionResult::SUCCESS;
118     }
119
120     bool moveTo(const vector6d_t& point, float time, bool is_cartesian)
    {
121         return moveTrajectory({point}, time, 0, is_cartesian);
122     }
123 };
124
125 int main(int argc, const char** argv) {
126     EliteDriverConfig config;
127     if (argc == 2) {
128         config.robot_ip = argv[1];
129     } else if (argc == 3) {
130         config.robot_ip = argv[1];
131         config.local_ip = argv[2];
132     } else {
133         std::cout << "Must provide robot IP. Example: ./
    trajectory_example aaa.bbb.ccc.ddd <eee.fff.ggg.hhh>" << std::endl;
134         return 1;
135     }
136     config.headless_mode = true;
137     config.script_file_path = "external_control.script";
138     std::unique_ptr<TrajectoryControl> trajectory_control = std::
    make_unique<TrajectoryControl>(config);
139     std::unique_ptr<RtsiIOInterface> rtsi_client = std::make_unique<
    RtsiIOInterface>("robot_position_recipe.txt", "", 250);
140
141     ELITE_LOG_INFO("Connecting to the RTSI");
142     if (!rtsi_client->connect(config.robot_ip)) {
143         ELITE_LOG_FATAL("Fail to connect or config to the RTSI.");
144         throw std::runtime_error("Fail to connect or config to the RTSI"
    );
145     }
146     ELITE_LOG_INFO("Successfully connected to the RTSI");
147
148     ELITE_LOG_INFO("Starting trajectory control...");
149     if(!trajectory_control->startControl()) {
150         ELITE_LOG_FATAL("Failed to start trajectory control.");
151         return 1;
152     }
153     ELITE_LOG_INFO("Trajectory control started");
154
155     vector6d_t actual_joints = rtsi_client->getActualJointPositions();
```

```
    actual_joints[3] = -1.57;

    ELITE_LOG_INFO("Moving joints to target: [%lf, %lf, %lf, %lf, %lf, %lf]",
                   actual_joints[0], actual_joints[1], actual_joints[2], actual_joints[3], actual_joints[4], actual_joints[5]);
    if(!trajectory_control->moveTo(actual_joints, 3, false)) {
        ELITE_LOG_FATAL("Failed to move joints to target.");
        return 1;
    }
    ELITE_LOG_INFO("Joints moved to target");


    vector6d_t actual_pose = rtsi_client->getAcutalTCPPose();
    std::vector<vector6d_t> trajectory;

    actual_pose[2] -= 0.2;
    trajectory.push_back(actual_pose);


    actual_pose[1] -= 0.2;
    trajectory.push_back(actual_pose);

    actual_pose[1] += 0.2;
    actual_pose[2] += 0.2;
    trajectory.push_back(actual_pose);

    ELITE_LOG_INFO("Moving joints to target");
    if(!trajectory_control->moveTrajectory(trajectory, 3, 0, true)) {
        ELITE_LOG_FATAL("Failed to move trajectory.");
        return 1;
    }
    ELITE_LOG_INFO("Joints moved to target");

    return 0;
}
```

编译步骤：

1. 使用以下指令编译此代码（如果您的编译器不支持 C++17，需要将命令中的 C++ 标准改为 C++14）：

```
g++ move_example.cpp -o move_example -lelite-cs-series-sdk --std=c++17
```

2. 编译完成后，会生成名为 **move_example** 的可执行文件。

配置文件准备：

1. 创建 **robot_position_recipe.txt** 文件，将以下内容复制到文件中：

```
actual_joint_positions
actual_TCP_pose
```

2. 获取 **external_control.script** 文件

- 如果您使用系统的包管理工具安装 SDK(如 Ubuntu 的 apt),通常可在**/usr/share/Elite/** 路径下找到该文件。
- 如果使用预编译包安装或编译安装 SDK，通常可以在**/usr/local/share/Elite/**路径下找到该文件。

将该文件拷贝到 **move_example** 所在的目录下。

运行程序：

1. 执行 **ifconfig** 指令，获取本机 IP。

2. 执行以下指令来运行程序（注意将 Your.Robot.IP.Addr 替换为您的机器人 IP 地址，Your.Local.IP.Addr 替换为本机 IP 地址，并确保机器人周围环境安全）：

```
./move_example Your.Robot.IP.Addr Your.Local.IP.Addr
```

运行成功后，机器人将会以当前位姿为起点，运行一个三角形轨迹。

可能出现的异常：

1. **机器人没有运动**
   示教器提示：”socket_read_binary_integer”：未连接服务端”reverse_socket”。出现这个异常的原因是机器人无法连接 SDK 的 TCP 服务端口。可参考 第 3.3.1 小节最后的**可能遇到的问题**部分的描述进行排查。

2. **机器人运动结束**
   示教器提示:”socket_read_binary_integer”:未连接服务端”script_command_socket”。此现象是程序在退出时出现了意外问题，可以尝试升级 SDK。此现象通常不需要过多关注。

3. **机器人运动过程中中断**
   机器人示教器日志栏提示:”Socket timed out waiting for command on reverse_socket. The script will exit now.”，或者提示：”socket_read_binary_integer”：未连接服务端”xxx_socket”。此现象是触发了 SDK 控制机器人的超时机制。

可能的原因包括：

- 运行 SDK 的电脑性能不足以支撑与控制柜的数据传输速度。
- 网络传输出现异常，例如网线松脱。

4. **端口被占用**

   如果程序抛出端口被占用的异常，使用以下指令查看是哪个端口被占用，然后关闭对应进程：

```
netstat -ano |grep 50001
netstat -ano |grep 50002
netstat -ano |grep 50003
netstat -ano |grep 50004
```

5. **奇异位姿报错**

   示教器提示：目标位姿为奇异位姿，机器人不可达。出现此现象时，使用示教器手动调整机器人位姿到一个合理的范围内。

4.2.2 Windows

参考第 2.2.3 小节，创建一个 MoveExample 新工程。

代码配置：

1. 将以下代码复制到 VS 项目中：

```
#include <Elite/DataType.hpp>
#include <Elite/EliteDriver.hpp>
#include <Elite/Log.hpp>
#include <Elite/DashboardClient.hpp>
#include <Elite/RtsiIOInterface.hpp>

#include <future>
#include <iostream>
#include <memory>
#include <thread>

using namespace ELITE;

class TrajectoryControl {
private:
    std::unique_ptr<EliteDriver> driver_;

    std::unique_ptr<DashboardClient> dashboard_;
    EliteDriverConfig config_;

public:
    TrajectoryControl(const EliteDriverConfig& config) {
        config_ = config;
        driver_ = std::make_unique<EliteDriver>(config);
        dashboard_ = std::make_unique<DashboardClient>();

        ELITE_LOG_INFO("Connecting to the dashboard");
        if (!dashboard_->connect(config.robot_ip)) {
            ELITE_LOG_FATAL("Failed to connect to the dashboard.");
            throw std::runtime_error("Failed to connect to the dashboard.");
        }
        ELITE_LOG_INFO("Successfully connected to the dashboard");
    }

    ~TrajectoryControl() {
        if (dashboard_) {
```

```
            dashboard_->disconnect();
        }
        driver_->stopControl();
    }

    bool startControl() {
        ELITE_LOG_INFO("Start powering on...");
        if (!dashboard_->powerOn()) {
            ELITE_LOG_FATAL("Power-on failed");
            return false;
        }
        ELITE_LOG_INFO("Power-on succeeded");

        ELITE_LOG_INFO("Start releasing brake...");
        if (!dashboard_->brakeRelease()) {
            ELITE_LOG_FATAL("Brake release failed");
            return false;
        }
        ELITE_LOG_INFO("Brake released");

        if (config_.headless_mode) {
            if (!driver_->isRobotConnected()) {
                if (!driver_->sendExternalControlScript()) {
                    ELITE_LOG_FATAL("Fail to send external control
    script");
                    return false;
                }
            }
        }
        else {
            if (!dashboard_->playProgram()) {
                ELITE_LOG_FATAL("Fail to play program");
                return false;
            }
        }

        ELITE_LOG_INFO("Wait external control script run...");
        while (!driver_->isRobotConnected()) {
            std::this_thread::sleep_for(std::chrono::milliseconds(10));
        }
        ELITE_LOG_INFO("External control script is running");
        return true;
    }

    bool moveTrajectory(const std::vector<vector6d_t>& target_points,
```

```
    float point_time, float blend_radius, bool is_cartesian) {
81      std::promise<TrajectoryMotionResult> move_done_promise;
82      driver_->setTrajectoryResultCallback([&](TrajectoryMotionResult
    result) { move_done_promise.set_value(result); });
83
84      ELITE_LOG_INFO("Trajectory motion start");
85      if (!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::START, target_points.size(), 200)) {
86          ELITE_LOG_ERROR("Failed to start trajectory motion");
87          return false;
88      }
89
90      for (const auto& joints : target_points) {
91          if (!driver_->writeTrajectoryPoint(joints, point_time,
    blend_radius, is_cartesian)) {
92              ELITE_LOG_ERROR("Failed to write trajectory point");
93              return false;
94          }
95          // Send NOOP command to avoid timeout.
96          if (!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::NOOP, 0, 200)) {
97              ELITE_LOG_ERROR("Failed to send NOOP command");
98              return false;
99          }
100     }
101
102     std::future<TrajectoryMotionResult> move_done_future =
    move_done_promise.get_future();
103     while (move_done_future.wait_for(std::chrono::milliseconds(50))
    != std::future_status::ready) {
104         // Wait for the trajectory motion to complete, and send NOOP
     command to avoid timeout.
105         if (!driver_->writeTrajectoryControlAction(ELITE::
    TrajectoryControlAction::NOOP, 0, 200)) {
106             ELITE_LOG_ERROR("Failed to send NOOP command");
107             return false;
108         }
109     }
110     auto result = move_done_future.get();
111     ELITE_LOG_INFO("Trajectory motion completed with result: %d",
    result);
112
113     if (!driver_->writeIdle(0)) {
114         ELITE_LOG_ERROR("Failed to write idle command");
115         return false;
```

```
116         }
117
118         return result == TrajectoryMotionResult::SUCCESS;
119     }
120
121     bool moveTo(const vector6d_t& point, float time, bool is_cartesian)
    {
122         return moveTrajectory({ point }, time, 0, is_cartesian);
123     }
124 };
125
126 int main(int argc, const char** argv) {
127     EliteDriverConfig config;
128     if (argc == 2) {
129         config.robot_ip = argv[1];
130     }
131     else if (argc == 3) {
132         config.robot_ip = argv[1];
133         config.local_ip = argv[2];
134     }
135     else {
136         std::cout << "Must provide robot IP. Example: ./
    trajectory_example aaa.bbb.ccc.ddd <eee.fff.ggg.hhh>" << std::endl;
137         return 1;
138     }
139     config.headless_mode = true;
140     config.script_file_path = "external_control.script";
141     std::unique_ptr<TrajectoryControl> trajectory_control = std::
    make_unique<TrajectoryControl>(config);
142     std::unique_ptr<RtsiIOInterface> rtsi_client = std::make_unique<
    RtsiIOInterface>("robot_position_recipe.txt", "", 250);
143
144     ELITE_LOG_INFO("Connecting to the RTSI");
145     if (!rtsi_client->connect(config.robot_ip)) {
146         ELITE_LOG_FATAL("Fail to connect or config to the RTSI.");
147         throw std::runtime_error("Fail to connect or config to the RTSI"
    );
148     }
149     ELITE_LOG_INFO("Successfully connected to the RTSI");
150
151     ELITE_LOG_INFO("Starting trajectory control...");
152     if (!trajectory_control->startControl()) {
153         ELITE_LOG_FATAL("Failed to start trajectory control.");
154         return 1;
155     }
```

```
156     ELITE_LOG_INFO("Trajectory control started");
157 
158     vector6d_t actual_joints = rtsi_client->getActualJointPositions();
159     actual_joints[3] = -1.57;
160 
161     ELITE_LOG_INFO("Moving joints to target: [%lf, %lf, %lf, %lf, %lf, %
    lf]",
162         actual_joints[0], actual_joints[1], actual_joints[2],
    actual_joints[3], actual_joints[4], actual_joints[5]);
163     if (!trajectory_control->moveTo(actual_joints, 3, false)) {
164         ELITE_LOG_FATAL("Failed to move joints to target.");
165         return 1;
166     }
167     ELITE_LOG_INFO("Joints moved to target");
168 
169 
170     vector6d_t actual_pose = rtsi_client->getAcutalTCPPose();
171     std::vector<vector6d_t> trajectory;
172 
173     actual_pose[2] -= 0.2;
174     trajectory.push_back(actual_pose);
175 
176 
177     actual_pose[1] -= 0.2;
178     trajectory.push_back(actual_pose);
179 
180     actual_pose[1] += 0.2;
181     actual_pose[2] += 0.2;
182     trajectory.push_back(actual_pose);
183 
184     ELITE_LOG_INFO("Moving joints to target");
185     if (!trajectory_control->moveTrajectory(trajectory, 3, 0, true)) {
186         ELITE_LOG_FATAL("Failed to move trajectory.");
187         return 1;
188     }
189     ELITE_LOG_INFO("Joints moved to target");
190 
191     return 0;
192 }
```

2. 参考第 2.2.3 小节编译项目，并拷贝相关库文件。

配置文件准备：

1. 将 **external_control.script** 文件拷贝到编译结果 MoveExample.exe 所在的目录

下。**external_control.script** 文件在源代码的 **source/resources** 路径下。如果使用预编译包，此文件在 **resources** 目录下。

2. 在编译结果 MoveExample.exe 目录下，创建 **robot_position_recipe.txt** 文件，将以下内容复制到文件中：

```
actual_joint_positions
actual_TCP_pose
```

运行程序：

在 Windows 终端中执行以下指令运行程序（注意将 Your.Robot.IP.Addr 替换为您的机器人 IP 地址，Your.Local.IP.Addr 替换为本机 IP 地址，并确保机器人周围环境安全）：

```
.\MoveExample.exe Your.Robot.IP.Addr Your.Local.IP.Addr
```

运行成功后，机器人将会以当前位姿为起点，运行一个三角形轨迹。

可能出现的异常：

1. **机器人没有运动**
   示教器提示：”socket_read_binary_integer”：未连接服务端”reverse_socket”。出现这个异常的原因是机器人无法连接 SDK 的 TCP 服务端口。可参考 第 3.3.1 小节最后的**可能遇到的问题**部分的描述进行排查。

2. **机器人运动结束**
   示教器提示：”socket_read_binary_integer”：未连接服务端”script_command_socket”。此现象是程序在退出时出现了意外问题，可以尝试升级 SDK。此现象通常不需要过多关注。

3. **机器人运动过程中中断**
   机器人示教器日志栏提示：”Socket timed out waiting for command on reverse_socket. The script will exit now.”，或者提示：”socket_read_binary_integer”：未连接服务端”xxx_socket”。此现象是触发了 SDK 控制机器人的超时机制。
   可能的原因包括：
   - 运行 SDK 的电脑不足以支撑与控制柜的数据传输速度。
   - 网络传输出现异常，例如网线松脱。

4. **程序异常弹窗**
   如果出现弹窗报告异常，可能有以下原因：

- **MoveExample.exe** 所在目录下没有正确拷贝 **robot_position_recipe.txt**、**external_control.script** 两个文件。
- 端口被占用，使用以下指令查看端口是否被占用（注意是 TCP 的端口）：

```
netstat -ano | findstr :50001
netstat -ano | findstr :50002
netstat -ano | findstr :50003
netstat -ano | findstr :50004
```

如果端口被占用，需要结束占用的进程。

5. **奇异位姿报错**

示教器提示：目标位姿为奇异位姿，机器人不可达。出现此现象时，使用示教器手动调整机器人位姿到一个合理的范围内。

第 5 章 SDK 示例

本章主要提供 SDK 主要功能的使用示例及其说明。

5.1 SDK 示例

访问SDK 示例下载链接获取 SDK 主要功能的使用示例。

提示

1. 以下示例的编译与运行均依赖 boost 库的 program_options。
2. 如无法访问 github 网站，请访问gitee 镜像仓库获取 SDK 使用示例。

5.1.1 connect_robot_test.cpp

```
./connect\_robot\_test <--robot-ip=ip> [--local-ip=""]
```

功能： 测试机器人与 SDK 是否可以连接。如果连接正常，将输出”Success, robot connected to PC”。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）

示例：
```
./connect\_robot\_test --robot-ip=192.168.1.200 --local-ip
=192.168.1.100
```

5.1.2 dashboard_example.cpp

```
./dashboard\_example <--robot-ip=ip>
```

功能： 连接机器人的 dashboard 端口后进行以下操作：
验证 dashboard 响应，打开/关闭机器人本体的电源、释放抱闸，加载"test" 任务，运行任务，暂停任务，停止任务，输出任务是否被保存。

参数： **–robot-ip arg**：机器人 IP

示例：
```
./dashboard\_example --robot-ip=192.168.1.200
```

5.1.3 freedrive_example.cpp

```
./freedrive\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode=
true]
```

功能： 让机器人进入拖动模式。此示例需手动打开机器人本体的电源并释放抱闸。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./freedrive\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.4 primary_example.cpp

```
./primary\_example <--robot-ip=ip>
```

功能： 从机器人的主端口（30001）获取机器人的运动学参数并打印。向机器人发送一个会触发运行时异常的脚本，捕获异常并打印。向机器人发送一个能正常运行的脚本，脚本功能是在机器人示教器「日志」打印"hello world"。

参数： **–robot-ip arg**：机器人 IP

示例：
```
./primary\_example --robot-ip=192.168.1.200
```

5.1.5 rtsi_example.cpp

```
./rtsi\_example <--robot-ip=ip>
```

功能： 连接机器人的 RTSI 服务，获取控制器版本信息并打印，设置标准数字 IO 0 为低电平，然后设置为高电平，并输出两次设置花费的时间。

参数： **–robot-ip arg**：机器人 IP

示例：
```
./rtsi\_example --robot-ip=192.168.1.200
```

5.1.6 servoj_cartesian_example.cpp

```
./servoj\_cartesian\_example <--robot-ip=ip> [--local-ip=""] [--use-headless
-mode=true]
```

功能： 打开机器人本体的电源，释放抱闸，使用机器人的伺服透传功能让机器人沿 Z 轴方向向下做约 10cm 的直线运动后继续向上做约 9cm 的直线运动。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./servoj\_cartesian\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.7 servoj_example.cpp

```
./servoj\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode=true]
```

功能： 给机器人上电，释放抱闸，使用机器人的伺服透传功能让机器人的第六关节 J6 旋转到 3 rad 后再旋转到 -3 rad。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./servoj\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.8 servoj_plan_example.cpp

```
./servoj\_plan\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode
=true]
```

功能： 类似servoj_example.cpp，但旋转时增加了梯形波的速度规划。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./servoj\_plan\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.9 speedj_example.cpp

```
./speedj\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode=true]
```

功能： 打开机器人本体的电源，释放抱闸，使用机器人本体的 speedj 功能让末端关节以 -0.1 rad/s 的速度旋转约 5 秒，再以 0.1 rad/s 的速度旋转约 5 秒。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./speedj\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.10 speedl_example.cpp

```
./speedl\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode=true]
```

功能： 打开机器人本体的电源，释放抱闸，使用机器人本体的 speedl 功能让机器人沿 Z 轴方向以 -0.02 rad/s 的速度运动约 5 秒，再以 0.02 rad/s 的速度运动约 5 秒。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./speedl\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.11 trajectory_example.cpp

```
./trajectory\_example <--robot-ip=ip> [--local-ip=""] [--use-headless-mode=
true]
```

功能： 打开机器人本体的电源，释放抱闸，让机器人本体以当前位姿为起点，运行一个三角形轨迹。

参数： **–robot-ip arg**：机器人 IP
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./trajectory\_example --robot-ip=192.168.1.200 --local-ip
=192.168.1.100 --use-headless-mode
```

5.1.12 serial_example.cpp

```
./serial_example  <--robot-ip=ip> <--ssh-pw=pw> [--local-ip=""] [--use-
headless-mode=true]
```

功能： 打开机器人本体的电源，释放抱闸，使用机器人工具 RS485 发送“hello world”字符串，并接收 12 字节的数据（超时时间为 5 秒）。

参数： **–robot-ip arg**：机器人 IP
–ssh-pw：机器人操作系统 root 用户的登录密码
–local-ip arg：运行 SDK 电脑的 IP（可选）
–use-headless-mode=true：是否使用 headless 模式（见第 5.2 节）。

示例：
```
./serial\_example --robot-ip=192.168.1.200 --ssh-pw=123456 --
local-ip=192.168.1.100 --use-headless-mode
```

提示

如需获取机器人操作系统 root 用户的登录密码，请联系 ELITE ROBOTS。

5.2 运行示例常见问题

1. **headless 模式**

headless 模式指无示教器模式。在这种模式下，程序直接将 EliteScript 脚本发送给机器人执行，不再需要运行”External Control” 这个 ELITECOs 插件。

2. **报警/报错说明**
   - 如果示教器出现报错：”socket_read_binary_integer：未连接服务器”，可参考 第 3.3.1 小节最后的**可能遇到的问题**部分的描述进行排查。
   - 示教器出现奇异点报警，可能的原因是运动开始时的位姿无法满足接下来要走的轨迹，请手动将机器人移动到一个合适的位姿。

第 6 章 API 手册与用户指南

6.1 用户指南

如需详细了解艾利特机器人 SDK 的使用，请查阅 用户指南。

6.2 API 手册

如需详细了解 API 的说明，请查阅 API 手册。

ELITE ROBOTS
艾利特机器人

明天比今天更简单一点

联系我们

商务合作：market@elibot.cn
技术咨询：technical@elibot.cn

- **苏州公司（生产基地）**
  苏州市工业园区长阳街 259 号中新钟园工业坊 4 栋
  +86-400-189-9358
- **北京公司**
  北京市经济技术开发区荣华南路 10 号院 5 号楼 611 室
- **上海公司（研创中心）**
  上海市浦东新区张江人工智能岛川和路 55 弄 20 号楼 3 层
- **深圳公司**
  深圳市宝安区航空路泰华梧桐岛科技创新园 1A 栋 202 室
- **美国公司**
  10521 Research Dr., Ste. 104, 37932, Knoxville, TN (USA)
- **德国公司**
  Münchener Str. 53, 85290, Geisenfeld, Bavaria (Germany)
- **日本公司**
  TOSHIN Hirokoji Honmachi Bldg., 1F, 2-4-3 Sakae, Naka-ku, 460-0008, Nagoya (Japan)
- **墨西哥公司**
  Calzada del pedregal 523, fraccionamiento el pedregal

关注公众号了解更多