ARIEMEDI

精准导航　智引未来

瑞瞳 SDK 开发说明文档

Linux C++版

版本号 V4.0.0

型号规格 RUITONG SDK

艾瑞迈迪医疗科技（北京）有限公司

目录

1. 简介

1.1 产品概述

适用于瑞瞳系列产品（瑞瞳 SE、瑞瞳 MAX）的 SDK 工具包，包含瑞瞳设备的连接与初始化、标志物定位、器械识别跟踪等功能，用户可根据业务需求结合 SDK 灵活的进行应用层开发。

1.2 环境要求

1.2.1 系统要求

Linux 系统，内核 4.15.0/5.15.0

1.2.2 开发环境与编译语言

Visual Studio Code，C++

2. SDK 接口说明

2.1 设备连接接口

相关头文件 ARMDCombinedAPI.h。

2.1.1 connect

原型

```
int connect(std::string hostname/IP);
```

功能描述

连接与初始化设备。

参数

std::string 类型：设备对应的 hostname 或 IP。

返回值

int 类型：连接设备成功或失败状态值，其中-1 对应“连接失败”，0 对应“连接成功”。

2.1.2 disconnect

原型

```
void disconnect();
```

功能描述

断开与设备的连接。

参数

无。

返回值

无。

2.1.3 getConnectionIPs

原型

```
std::vector<std::string> getConnectionIPs();
```

功能描述

获取本地网卡与设备的 IP 地址。

参数

无。

返回值

std::vector<std::string>类型：vector 容器储存了本地网卡与设备的 IP 地址。

2.1.4 printAPIandFirmwareVersion

原型

```
void printAPIandFirmwareVersion();
```

功能描述

打印 SDK 与固件版本号。

参数

无。

返回值

无。

2.2 器械设置接口

相关头文件 ARMDCombinedAPI.h。

2.2.1 generateAROM

原型

```
uint16_t generateAROM(std::string directory, ToolCalibrationData tool);
```

功能描述

根据用户输入的参数，生成器械标定文件（*.arom）。**注意**：该接口函数无法替代器械尖端标定过程，若要确定器械尖端坐标，请使用标定软件。

参数

std::string 类型：器械标定文件所在文件夹 directory；

ToolCalibrationData 类型：器械标定结构体 tool。

返回值

uint16_t 类型：生成器械标定文件（*.arom）成功或失败信息，其枚举类型命名空间为 GenerationStatus。

2.2.2 convert2AROM

原型

uint16_t convert2AROM(std::string directoryOut, std::string directoryIn);

功能描述

将用户输入器械非标准标定文件（*.rom）转换为标准标定文件（*.arom）。

参数

std::string 类型：输出标准标定文件（*.arom）路径 directoryOut；

std::string 类型：输入非标准标定文件（*.rom）路径 directoryIn。

返回值

uint16_t 类型：生成器械标定文件（*.arom）成功或失败信息，其枚举类型命名空间为 GenerationStatus。

2.2.3 loadPassiveToolAROM

原型

void loadPassiveToolAROM (std::string directory);

功能描述

从指定文件夹目录读取所有器械标定文件（*.arom）。

参数

std::string 类型：器械标定文件所在文件夹 directory。

返回值

无。

2.2.4 loadPassiveToolNAROM

原型

void loadPassiveToolNAROM (std::string dirctory);

功能描述

从指定文件夹目录读取所有器械非标准标定文件（*.rom）。**注意**：读取非标准标定文件存在风险，建议用户读取标准标定文件。

参数

std::string 类型：器械非标准标定文件所在文件夹 directory。

返回值

无。

2.2.5 getToolStorage

原型

```
std::vector<std::vector<MarkerPosition>> getToolStorage();
```

功能描述

获取所有器械标定数据。

参数

无。

返回值

std::vector<std::vector<MarkerPosition>>类型：嵌套 vector 容器，其中外层容器储存了不同器械的标定数据，内层容器储存了单个器械的所有标志物的坐标 MarkerPosition。

2.2.6 getTrackToolsNum

原型

```
int getTrackToolsNum();
```

功能描述

获取待跟踪器械数量。

参数

无。

返回值

int 类型：待跟踪器械数量，其与读入的器械标定文件（*.arom）数量一致。

2.2.7 getToolName

原型

```
std::vector<std::string> getToolName();
```

功能描述

获取所有器械名称。

参数

无。

返回值

std::vector<std::string>类型：vector 容器储存了所有器械的名称。

2.3 数据传输接口

相关头文件 ARMDCombinedAPI.h。

2.3.1 startTracking

原型

```
void startTracking();
```

功能描述

开始跟踪器械。

参数

无。

返回值

无。

2.3.2 stopTracking

原型

```
void stopTracking();
```

功能描述

停止跟踪器械。

参数

无。

返回值

无。

2.3.3 startImaging

原型

```
void startImaging();
```

功能描述

开始传输双目图像数据。

参数

无。

返回值

无。

2.3.4 stopImaging

原型

void stopImaging ();

功能描述

停止传输双目图像数据。

参数

无。

返回值

无。

2.3.5 trackingUpdate

原型

void trackingUpdate();

功能描述

刷新所有器械的跟踪数据与双目图像数据。

参数

无。

返回值

无。

2.3.6 getAllMarkers

原型

std::vector<MarkerData> getAllMarkers();

功能描述

获取当前视场中所有标志物的跟踪数据。

参数

无。

返回值

std::vector<MarkerData>类型：vector 容器储存了当前视场中所有标志物的跟踪数据 MarkerData。

2.3.7 getLeftImagingData

原型

char* getLeftImagingData();

功能描述

获取当前左目相机图像，图像尺寸为 1280*800 像素。

参数

无。

返回值

char*类型：当前左目相机图像数据。

2.3.8 getRightImagingData

原型

```
char* getRightImagingData();
```

功能描述

获取当前右目相机图像，图像尺寸为 1280*800 像素。

参数

无。

返回值

char*类型：当前右目相机图像数据。

2.3.9 getTrackingData

原型

```
std::vector<ToolTrackingData> getTrackingData(std::vector<MarkerPosition> cordi);
```

功能描述

获取当前所有器械的跟踪数据。

参数

std::vector<MarkerPosition>类型：vector 容器储存当前视场中所有标志物的跟踪数据 cordi。

返回值

std::vector<ToolTrackingData>类型：vector 容器储存了当前所有器械的跟踪数据 ToolTrackingData。

2.3.10 getUnMatchedMarkers

原型

```
std::vector<MarkerPosition> getUnMatchedMarkers();
```

功能描述

获取当前所有游离的未匹配标志物跟踪数据。该函数需在 getTrackingData 后运行，才可以获取正确数量。

参数

无。

返回值

std::vector<MarkerPosition>类型：vector 容器储存了当前视场中所有游离标志物的跟踪数据 MarkerPosition。

2.3.11 getSystemTime

原型

std::string getSystemTime();

功能描述

获取当前系统时间。

参数

无。

返回值

std::string 类型：当前系统时间。

2.4 状态监测接口

相关头文件 ARMDCombinedAPI.h。

2.4.1 getGravityVector

原型

std::vector<double> getGravityVector ();

功能描述

获取当前设备重力向量。

参数

无。

返回值

std::vector<double>类型：vector 容器储存了当前设备重力单位向量。

2.4.2 getConnectionStatus

原型

uint16_t getConnectionStatus();

功能描述

获取当前设备连接状态。

参数

无。

返回值

uint16_t 类型：当前设备连接状态，其枚举类型命名空间为 ConnectionStatus。

2.4.3 getTransmissionStatus

原型

```
uint16_t getTransmissionStatus();
```

功能描述

获取当前设备传输数据类型。

参数

无。

返回值

uint16_t 类型：当前设备传输数据类型，其枚举类型命名空间为 TransmissionStatus。

2.4.4 getSystemAlert

原型

```
uint16_t getSystemAlert();
```

功能描述

获取当前设备警告信息。

参数

无。

返回值

uint16_t 类型：当前设备温度、震动警告信息，其枚举类型命名空间为 DeviceAlert。

3. SDK 重要数据结构

3.1 器械标定信息数据结构 ToolCalibrationData

```
struct ToolCalibrationData
{
    std::string name = “”;                  //器械名称
    bool type = false;                      //器械类型
    int minNumMarker = 3;                   //最少匹配标志物数量
    int planeNum = 1;                       //平面数量
    std::vector<std::vector<MarkerPosition>> markers;//每个平面标志物数量
    double pin[3] = {0.0, 0.0, 0.0};        //尖端坐标
    double calbError = -1;                  //标定误差，-1 表示器械未标定
```

```
    double maxFRE = 1;                  //最大匹配误差
    double maxAngle = 60;               //最大匹配角度
};
```

3.2 跟踪信息数据结构 ToolTrackingData

```
struct ToolTrackingData
{
    std::string name = "";              //名称
    Transformation transform;           //位姿信息
    int matchStatus = 0;                //匹配状态：1（跟踪），0（丢失）
    int matchedPlane = 0;               //匹配平面编号
    int matchedMarkersNum = 0;          //平面内匹配标志物数量
    std::vector<MarkerPosition> markers;  //标志物坐标
    std::string timespec = "";          //当前时间
}
```

3.3 标志物位置数据结构 MarkerPosition

```
struct MarkerPosition
{
    double P[4];                        //标志物坐标
    int leftExposure = 0;               //标志物亮度
    int rightExposure = 0;              //标志物亮度

};
```

3.4 跟踪位姿信息数据结构 Transform

```
struct Transformation
{
    uint16_t status;            //匹配状态，其枚举类型命名空间为 TransformationStatus
    double matrix[4][4];        //姿态矩阵
    double qw, qx, qy, qz;      //姿态四元数
    double tx, ty, tz;          //器械尖端坐标
    double error;               //匹配误差
};
```

4. SDK 重要枚举类型的命名空间

4.1 器械标定文件生成状态 GenerationStatus

```
namespace GenerationStatus
{
    enum value
    {
        OK = 0x0000,                    //生成成功
        MinMatchedNumLess = 0x0001,     //生成失败，最少匹配标志物数量过少
        MinMatchedNumMore = 0x0002,     //生成失败，最少匹配标志物数量过多
```

```
        PlaneNumLess = 0x0003,                //生成失败，平面数量过少
        PlaneNumMore = 0x0004,                //生成失败，平面数量过多
        PlaneNumNotMatching = 0x0005,         //生成失败，平面数量不匹配
        MarkerNumLess = 0x0006,               //生成失败，平面内标志物数量过少
        MarkerNumMore = 0x0007,               //生成失败，平面内标志物数量过多
        MarkerNumLimitExceed = 0x0008,        //生成失败，器械标志物数量过多
        DistanceTooSmall = 0x0009,            //生成失败，标志物间距过小
        NameError = 0x000A,                   //生成失败，文件名包含中文与符号
        PathNotExist = 0x000B,                //生成失败，文件存储路径不存在
        TypeError = 0x000C,                   //生成失败，文件类型错误
    };
}
```

4.2 设备传输数据类型 TransmissionStatus

```
namespace TransmissionStatus
{
    enum value
    {
        NoneData = 0x0000,                    //无数据传输
        TrackingData = 0x0001,                //跟踪数据传输
        ImagingData = 0x0002,                 //图像据传输
        AllData = 0x0003,                     //跟踪+图像数据传输
    };
}
```

4.3 设备连接状态 ConnectionStatus

```
namespace ConnectionStatus
{
    enum value
    {
        Connected = 0x0000,                   //连接成功
        DisConnected = 0x0001,                //断开成功
        DisConnectedFailed = 0x0002,          //断开失败
        PortConflicts = 0x0003,               //连接失败，连接端口冲突
        ResloveError = 0x0004,                //连接失败，IP解析错误
        InitialUDPError = 0x0005,             //连接失败，初始化错误
        Interruption = 0x0006,                //连接中断
        InvalidAddr = 0x0007,                 //连接失败，IP地址无效
        DeviceInexist = 0x0008,               //连接失败，设备不存在
        UnknownError = 0x0009,                //连接失败，未知错误
    };
}
```

4.4 设备警告信息 DeviceAlert

```
namespace DeviceAlert
{
```

```
        enum value
        {
            Normal = 0x0000,              //无警告
            Vibration = 0x0001,           //震动警告，剧烈震动已影响设备精度，建议返厂
            TempTooHigh = 0x0002,     //高温警告，高温已影响设备精度，请为设备降温
            TempTooLow = 0x0003,     //低温警告，低温已影响设备精度，请提升环境温度
        };
    }
```

4.5 器械跟踪状态 TransformationStatus

```
    namespace TransformationStatus
    {
        enum values
        {
            Enabled = 0x0000,                 //器械跟踪成功
            PartiallyOutOfVolume = 0x0001,//器械跟踪成功，但部分标志物超出视场范围
            OutOfVolume = 0x0002,           //器械跟踪成功，但全部标志物超出视场范围
            PartiallyMatched = 0x0003,       //器械跟踪成功，但部分标志物未识别到
            TooFewMarkers = 0x0004,         //器械跟踪失败，匹配标志物数量低于设定值
            BadTransformFit = 0x0005,       //器械跟踪失败，器械跟踪误差超过设定值
            ToolMissing = 0x0009,             //器械跟踪失败，视场中不存在器械的标志物
        };
    }
```

5. Demo 使用指南

测试该 Demo 的 Linux 系统为 Ubuntu 18.04/22.04，开发环境为 Visual Studio Code。

5.1 连接准备

将接入瑞瞳的网卡 IPv4 连接方式设置为 Link-Local Only。查看接入瑞瞳设备背面标记的 hostname，例如 RT-SE000001.local。

5.2 文件目录

include 文件夹包含 SDK 头文件（*.h），lib 文件夹包含 SDK 库文件（libARMDsterotracking.so 或 libARMDsterotrackingd.so），tool 文件夹包含器械标定文件（*.arom），调用 SDK 的示例代码为 main.cpp。

5.3 工程配置

Visual Studio Code 项目里在 tasks.json 文件中添加库及头文件名称与路径，其中 tasks.json 文件详细如下：

```
{
```

```
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++ build active file",
            "command": "/usr/bin/g++",
            "args": [
                "-g",
                "${workspaceFolder}/main.cpp",
                "-o",
                "${workspaceFolder}/SDK-Demo",
                "-I${workspaceFolder}/include",（SDK 的头文件目录）
                "-L${workspaceFolder}/lib/Debug",（SDK 的库目录）
                "-lARMDsterotrackingd"（SDK 的库名称）
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "detail": "Task generated by Debugger."
        }
    ],
    "version": "2.0.0"
}
```

5.4 调用流程

Step1：连接设备并初始化，调用 connect(hostname)

设置设备的 hostname，并连接。

Step2：读入器械标定文件，调用 configureTools ()

读入 tool 路径下所有器械标定文件（*.arom）。

Step3：显示器械跟踪信息，调用 printTrackingData ()

器械跟踪信息包括名称、标志物坐标、当前时间、配准误差、姿态矩阵和尖端点坐标。

Step4：关闭系统，调用 closeSystem ()

停止跟踪，断开连接。

6. 常见问题

Q：启动系统时显示连接失败

A：检查连接时输入的 hostname 是否与设备背面标记的一致。