AIeveR
Robotics Limited
雲芯機器人有限公司

DepthSight - L 系列 3D 激光线扫相机通信库

参考手册

V3.0.0

修订日期：2025.06

AIeveR Robotics Limited

软件使用许可协议

使用本软件前，客户需同意下述软件使用许可协议 （以下简称“本协议”）的内容。 客户使用或复制 DepthSight - L 系列 3D 激光线扫相机通信库（以下简称“本软件”）的部分或全部功能时，将视为客户已同意本协议中规定的所有内容，且本协议成立。

第 1 条 （使用权的授予）

在客户遵守本协议内容的前提下，云芯未来机器人（深圳）有限公司（以下简称“本公司”）将对客户授予本软件的非独占性使用权。

第 2 条 （禁止事项）

客户不可对本软件实施以下操作或改动。

a. 更改本软件现有功能或向本软件添加新功能。

b. 对本软件实施的所有逆向工程行为，如：反编译、反汇编等。

c. 向第三方二次销售、转让、二次分配、授予、租赁本软件及本公司提供的本软件授权密钥等。但允许客户将本软件与使用本软件编写的应用程序共同进行二次分配。

第 3 条 （著作权等）

与本软件及本软件手册相关的所有知识产权 （如著作权），归本公司所有。

第 4 条 （免责）

客户或第三方因使用本软件而遭受的所有损害，本公司概不负责!

第 5 条 （支持）

本公司将基于本协议，根据客户针对本软件提出的问题，提供技术支持。但并不保证本公司提供的技术支持服务可使客户达成期望目的。

第 6 条 （协议终止）

1. 当客户进行废弃本软件及其复制品等以停止使用本软件时，本协议自动终止。

2. 当客户违反本协议中规定的任一条款时，本公司可单方面解除本协议。同时，客户应立即废弃本软件及其复制品，或将之返还至本公司。

3. 因客户违反本协议，而使本公司蒙受损失时，客户应向本公司赔偿相关损失。

第 7 条 （准据法）

本协议遵从中华人民共和国法律。

前言

使用前请务必阅读本用户手册。

阅读后，请妥善保管，以便日后查阅。

DepthSight - L 系列 3D 激光线扫相机通信库，为客户提供通过应用程序控制 DepthSight - L 系列 3D 激光线扫相机的通信接口。具体使用方法，请向本公司索取示例程序。

本手册的内容是本着准确无误的目标进行编制的。但是如果发现有不清楚、错误或含糊的内容，请联系本公司销售部们。如有缺页或装订错误，本公司予以更换。

联系我们

云芯未来机器人（深圳）有限公司

电子邮件：whxu@aiever-robotics.com

地 址：深圳市福田区 ISPSZHK 深港国际科技园 E 栋 6 楼 608 室

邮 编：518000

感谢选用云芯机器人的机器视觉相关产品

为回报客户，我们将以一流的机器视觉产品、完善的售后服务、高效的技术支持，帮助您建立自己的机器视觉系统。

云芯机器人的更多信息

在我们的网页上可以获得更多关于公司 和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等。您也可以通过邮件（whxu@aiever-robotics.com）咨询关于公司和产品的更多信息。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

电子邮件：whxu@aiever-robotics.com

地 址：深圳市福田区 ISPSZHK 深港国际科技园 E 栋 6 楼 608 室

邮 编：518000

用户手册的用途

用户通过阅读本手册，能够熟悉机器视觉传感产品 DepthSight - L 系列 3D 激光线扫相机通信库的基本使用，能够在实际工程项目中进行开发。

用户手册的使用对象

本用户手册适用于，对软件开发调试操作有一定了解的工程人员。

用户手册的主要内容

本手册由 9 节内容组成。详细介绍了 DepthSight - L 系列 3D 激光线扫相机通信库的运行环境、文件构成、嵌入方式、变量类型、结构体定义、类详细说明、开发例程等。

相关文件

关于 L 系列 3D 激光线扫相机的硬件安装，请参阅随本产品配套的《DepthSight - L 系列 3D 激光线扫相机产品快速使用指南》。关于 DepthSight - L 系列 3D 激光线扫相机的软件调试，请参阅随本产品配套的《DepthSight - L 系列 3D 激光线扫相机软件使用手册》

文档修订版本

编号	版本号	修订日期
	1.0.0	2025.01.13
	2.0.0	2025.03.15
	3.0.0	2025.06.06

目录

1. 运行环境

操作系统	Windows 10 64bit 系统 Windows 11 64bit 系统
CPU	Intel I5 及以上
GPU	带有独显的电脑最佳
内存容量	16GB 以上
硬盘空间	10GB 以上
通信接口	Ethernet 1000BASE

* 确保电脑网卡是千兆以及网口 2048 字节的缓存大小是否达到要求。

1.1. Microsoft C++ Runtime Library

这是执行 dll 所需的运行环境，随 SDK 发送给用户，存放路径为“sdk_c++/env/vc_redist.x64.exe”。

1.2. 防火墙设置

使用 SDK 生成的工程，在运行可执行文件时需要勾选“专用网络”和“公用网络”选项。

1.3. 接口测试电脑配置及环境

测试电脑 1

电脑型号	Dell 天逸 510S-07IRB
操作系统	Windows 11 64bit 系统
CPU	Intel I7-14700 2.10Hz
内存容量	32GB
硬盘空间	1TB
通信接口	Ethernet 1000BASE

测试电脑 2

电脑型号	Dell OptiPlex 3050
操作系统	Windows 10 64bit 系统
CPU	Intel Core (TM) I5-7500T 2.7GHz
内存容量	16GB
硬盘空间	500GB
通信接口	Ethernet 1000BASE

2. 文件构成

Release 版本 SDK:

AIRLink.dll	DLL 本体
AIRLink.lib	DLL 本体的依赖库
Communication.dll	相机通信动态库
Communication.lib	相机通信动态库的依赖库
scanner_sdk.dll	相机底层 sdk 动态库
scanner_sdk.lib	相机底层 sdk 的依赖库
AIScanner.h	定义通信接口的头文件
AICommonTypes.h	定义接口的类型头文件
ErrorStatus.h	定义状态码的头文件
api_global.h	定义外部库的头文件
PostProcessing.h	定义点云算法后处理的头文件
ScannerProperties.h	定义相机型号的头文件
ScannerSetting.h	定义相机工作参数的头文件
AIeveR.h	L 系列相机通信库头文件

Debug 版本的 SDK 库文件有“d”后缀。

3. SDK 使用示例

- **在工程中使用 SDK**

  SDK 包含 include/AIeveR_Head、bin、lib、env 文件夹以及配置文件，直接放入工程目录下即可。注意区分 bin 文件与 lib 文件下的 debug 版本库与 release 版本库。

- **头文件**

  请确保下列头文件在 include 文件夹中:

```
include
├─AIeveR.h
└─AIeveR_Head
    ├─AICommonTypes.h
    ├─AIScanner.h
    ├─AIScannerDeveloper.h
    ├─api_global.h
    ├─ErrorStatus.h
    ├─PostProcessing.h
    ├─ScannerProperties.h
    └─ScannerSetting.h
```

- **示例参数配置文件**：DefaultParameters.txt，可用于批量导入参数配置。
- **手动链接**

  Release 模式下请链接“AIRLink.lib”、 “Communication.lib”、“scanner_sdk.lib”；Debug 模式下请链接“AIRLinkd.lib”、“Communicationd.lib”、“scanner_sdkd.lib”。
- **CMake 链接**

  在 CMakeLists.txt 添加下面的语句，即可链接到 SDK。

```
set(AIRLink_Cpp_DIR ${CMAKE_SOURCE_DIR}/cmake)
find_package(AIRLink_Cpp REQUIRED)
set(AIRLink_Cpp_INCLUDE_DIRS ${CMAKE_SOURCE_DIR}/include)
message(STATUS "AIRLink_Cpp_INCLUDE_DIRS = ${AIRLink_Cpp_INCLUDE_DIRS}")

if(AIRLink_Cpp_FOUND)
    message("AIRLink_Cpp found")
else()
    message(FATAL_ERROR "Cannot find AIRLink_Cpp")
endif()

target_link_libraries(${PROJECT_NAME}
    PRIVATE
    scanner_sdk::communication
    scanner_sdk::scanner_sdk
     scanner_sdk::AIRLink_Cpp
)

target_include_directories(${PROJECT_NAME}
    PRIVATE
    ${AIRLink_Cpp_INCLUDE_DIRS}
)
```

4. 数据结构—变量类型

* 所有的变量名都被包含在命名空间 AleveR 中。

本文使用的基本 C++变量类型定义如下：

uint32_t	无符号的 32 bit 整数
uint8_t	无符号的 8 bit 整数
size_t	无符号 32/64 bit 整数
unsigned int	无符号的 32 bit 整数
int	有符号的 32 bit 整数
char	有符号的 8 bit 字符
float	有符号的 32 bit 浮点数
double	有符号的 64 bit 浮点数
bool	布尔值

std::string	STL 字符串
std::vector	STL 向量容器

5. 数据结构—结构体定义

* 所有的变量名都被包含在命名空间 AIeveR 中。

名称	设置取值范围区间
所属文件	AICommonTypes.h

定义	`template <typename T>`
	`struct Range{`
	`  constexpr Range() : _min(0), _max(0) {}`
	`  constexpr Range(T min, T max) : _min(min), _max(max) {}`
	`  T _min;`
	`  T _max;};`

描述	用于设定某一变量类型的取值区间。

表 1 Range

名称	设置 float 类型的 3D 坐标
所属文件	AICommonTypes.h

定义	`struct AIeveR_Point3F{`
	`  AIeveR_Point3F() :x(0.), y(0.), z(0.){}`
	`  AIeveR_Point3F(float x, float y, float z):x(x), y(y), z(z){}`
	`  float x;`
	`  float y;`
	`  float z;};`

描述	功能类似 opencv 中的 Point3f 类型，用于存放 3D 坐标。

表 2 AIeveR_Point3F

名称	设置 double 类型的 3D 坐标
所属文件	AICommonTypes.h

定义	`struct AIeveR_Point3D{`
	`  AIeveR_Point3D() :x(0.), y(0.), z(0.){}`
	`  AIeveR_Point3D(double x, double y, double z):x(x), y(y), z(z){}`
	`  double x;`
	`  double y;`
	`  double z;};`

描述	功能类似 opencv 中的 Point3d 类型，用于存放 3D 坐标。

表 3 AIeveR_Point3D

名称	自定义 2D 尺寸
所属文件	AICommonTypes.h

定义	struct Size{ constexpr Size() : width(0), height(0) {} constexpr Size(size_t width, size_t height) : width(width), height(height){} bool operator==(const Size& other) { return width == other.width && height == other.height; } size_t width; size_t height;};
描述	自定义/初始化设置图像的宽和高，重载“==”符号，用初始化值判断宽和高是否相等，返回 bool 值。

表 4 Size

名称	自定义图像的 ROI
所属文件	AICommonTypes.h
定义	struct ROI{ constexpr ROI() : upperLeftX(0), upperLeftY(0), width(0), height(0) {} constexpr ROI(unsigned upperLeftX, unsigned upperLeftY, size_t width, size_t height) : upperLeftX(upperLeftX), upperLeftY(upperLeftY), width(width), height(height){} bool operator==(const ROI& other) { return width == other.width && height == other.height; } unsigned upperLeftX; unsigned upperLeftY; size_t width; size_t height;};
描述	自定义/初始化设置图像的 ROI，重载“==”符号，用初始化值判断 ROI 区域的宽和高是否相等，返回 bool 值。

表 5 ROI

名称	设备 IP 分配的方式
所属文件	AICommonTypes.h

定义	enum struct IpAssignmentMethod { Unknown, DHCP = 4, Static };
	inline static std::string ipAssignmentMethodToString(IpAssignmentMethod type){
	switch (type) {
	case IpAssignmentMethod::Static: //静态分配 IP
	return "Static";
	case IpAssignmentMethod::DHCP: //自动分配 IP
	return "DHCP";
	case IpAssignmentMethod::Unknown:
	return "Unknown";
	}
	return "";}

描述	设置设备的 IP 分配方式，默认为 IP4。

表 6 IpAssignmentMethod

名称	CMOS 获取数据格式设置
所属文件	AIScanner.h

定义	struct AIeveR_Data{
	int data_width = 3200;
	uint32_t *pc_ptr_ = nullptr;
	uint32_t pc_ptr_length_ = 0;
	uint8_t *gray_ptr_ = nullptr;
	uint32_t gray_ptr_length_ = 0 ;
	std::vector<unsigned int> frame_cnt_vec;
	std::vector<int> encoder_value_vec;
	};

描述	设置 CMOS 接收数据的格式。变量说明: int data_width = 3200 // 每条线的点云数量 uint32_t *pc_ptr_ = nullptr // 获取到轮廓数据指针 uint32_t pc_ptr_length_ = 0 // 获取到的轮廓数据数量 uint8_t *gray_ptr_ = nullptr // 获取到灰度数据指针 uint32_t gray_ptr_length_ = 0 // 获取到的灰度数据数量 std::vector<unsigned int> frame_cnt_vec // 获取到的帧号值向量 std::vector<int> encoder_value_vec // 获取到的编码器值向量

表 7 AIeveR_Data

名称	错误类型描述
所属文件	ErrorStatus.h

定义	struct ErrorStatus{
	enum ErrorCode {
	AIEVER_STATUS_TIMEOUT_ERROR = -1, ///< 操作超时。
	AIEVER_STATUS_SUCCESS = 0, ///< 操作成功。
	AIEVER_STATUS_CONSTRUCTER_FAIL_NULLPTR_ERROR, ///< 空指针。
	AIEVER_STATUS_INVALID_CALL_ERROR, ///< 调用该函数前必须调用其他函数
	AIEVER_STATUS_UDP_INIT_FAIL_ERROR, ///< udp 通信连接初始化失败。
	AIEVER_STATUS_IP_NOT_MATCH_ERROR, ///< ip 不匹配。
	AIEVER_STATUS_VERIFY_FAIL_ERROR, ///< 参数下发失败。
	AIEVER_STATUS_OUT_OF_RANGE_ERROR, ///< 输入值超出取值范围
	AIEVER_STATUS_SWITCH_SUCCESS, ///< 模式切换成功。

	AIEVER_STATUS_INVALID_DEVICE , ///< 无效设备（上位机与指定设备已断开连接或尚未建立连接） AIEVER_STATUS_NO_SUPPORT_ERROR , ///< 当前不支持该操作。 AIEVER_STATUS_INVALID_INPUT_ERROR, ///< 输入的参数为空或异常。 AIEVER_STATUS_FILE_IO_ERROR , ///< 文件 IO 操作异常。 AIEVER_STATUS_DEVICE_BUSY , ///< 设备忙。 AIEVER_STATUS_SDK_INTERNAL_ERROR, ///< SDK 内部错误。 AIEVER_STATUS_NOT_RUNNING ///< 设备未运行。}; ErrorStatus() = default; ErrorStatus(ErrorCode code, const std::string& message) : errorCode(code), errorDescription(message){} bool isOK() const { return (errorCode == AIEVER_STATUS_SUCCESS || errorCode == AIEVER_STATUS_SWITCH_SUCCESS); } ///判断是否操作成功 ErrorCode errorCode{AIEVER_STATUS_SUCCESS}; std::string errorDescription;};
描述	描述所有系统错误的类型，在 ErrorStatus 中枚举，在调用通信接口时作为判断返回值的规范。 成员变量：errorDescription ///错误描述

表 8 ErrorStatus

名称	点云拼接数据
所属文件	PostProcessing.h

定义	template <typename T>
	struct ProfileStitcherParams{
	std::vector < AIeveR_Point3F > PointVec {};
	unsigned long long ValidStitcherCounts {0}; //返回有效拼接的点云行数
	std::vector<int32_t > FlagValues {}; };

描述	PointVec：需要拼接的点云帧数； FlagValues：编码器值或者帧号； 其中，PointVec 的大小与 FlagValues 的大小需要一致，且两者都需要大于 0。

表 9 ProfileStitcherParams

名称	设备信息（设备搜索用）
所属文件	ScannerProperties.h

定义	struct AIeveR_ScannerInfo{ std::string Scanner_Type; std::string Scanner_Ip; uint16_t Scanner_Port; std::string Scanner_Mac;};
描述	变量说明: Scanner_Type //相机型号 Scanner_Ip //IP 地址 Scanner_Port //通信端口 Scanner_Mac //Mac 地址

表 10 AIeveR_ScannerInfo

名称	相机固件信息
所属文件	ScannerProperties.h

定义	struct AIeveR_ScannerFirewareVersion{
	std::string camera_series;
	std::string camera_type;
	int fireware_major;
	int fireware_minor;
	int fireware_date;
	int fireware_date_version;
	};

描述	变量说明: std::string camera_series //相机系列 std::string camera_type //相机类型 int fireware_major //固件主版本号 int fireware_minor //固件次版本号 int fireware_date //固件日期 int fireware_date_version //固件日期批次,指的是固件日期下的第几个版本

表 11 AIeveR_ScannerFirewareVersion

名称	L 系列相机名称
所属文件	ScannerProperties.h

定义	struct Series_Properties{
	static constexpr const int series_L10400_workdist = 140;
	static constexpr const int series_L11600_workdist = 400;
	static constexpr const int series_L10140_workdist = 1600;
	};

描述	根据工作距离由小到大排列 140mm、400mm、1600mm，对应 L 系列相机的型号分为 L10140、L10400、L11600。 三种型号的相机的点云间距分别为 0.035mm、0.1mm、0.5mm。

表 12 Series_Name

*** ScannerSetting.h 的参数描述类在 AIRLink 类一节中讲解**

6. 状态码定义

状态码	值	定义
AIEVER_STATUS_TIMEOUT_ERROR	-1	操作超时
AIEVER_STATUS_SUCCESS	0	操作成功
AIEVER_STATUS_CONSTRUCTER_FAIL_NULLPTR_ERROR	1	空指针
AIEVER_STATUS_INVALID_CALL_ERROR	2	调用该函数前必须调用其他函数
AIEVER_STATUS_UDP_INIT_FAIL_ERROR	3	udp 通信连接初始化失败
AIEVER_STATUS_IP_NOT_MATCH_ERROR	4	ip 不匹配
AIEVER_STATUS_VERIFY_FAIL_ERROR	5	参数下发失败
AIEVER_STATUS_OUT_OF_RANGE_ERROR	6	输入值超出取值范围
AIEVER_STATUS_SWITCH_SUCCESS	7	模式切换成功
AIEVER_STATUS_INVALID_DEVICE	8	无效设备（上位机与指定设备已断开连接或尚未建立连接）
AIEVER_STATUS_NO_SUPPORT_ERROR	9	当前不支持该操作
AIEVER_STATUS_INVALID_INPUT_ERROR	10	输入的参数为空或异常
AIEVER_STATUS_FILE_IO_ERROR	11	文件 IO 操作异常
AIEVER_STATUS_DEVICE_BUSY	12	设备忙
AIEVER_STATUS_SDK_INTERNAL_ERROR	13	SDK 内部错误
AIEVER_STATUS_NOT_RUNNING	14	设备未运行

表 13 Status_code

7. 参数配置文件

用户可以将自定义配置编写到 txt 格式的配置文件中，使用 loadParameters 函数读取。编写时，将配置项目（键）和配置量（值）在写在同一行，使用空格隔开。同时支持使用 # 号进行注释。

下面的文本是一个完整的配置文件的内容示例。

注意：程序控制相机工作时修改配置文件无效，不建议在相机工作时修改参数。

示例：DefaultParameters.txt

```
FrameRateTriggerMode 0 # 触发模式 0（固定帧率触发）
DataMode 1 # 数据模式 1 ()
LaserMode 1 # 激光模式 1 ()
Gain 0 # 增益 0 ()

# 设置曝光
ShortExposureTime 1000 # 短曝光时间 1000 ms
# MidExposureTime 1001 # 中曝光时间 1001 ms（被注释掉）
```

```
# LongExposureTime 1002 # 长曝光时间 1002 ms (被注释掉)


ExposureMode 0 # 曝光模式 0 ()
#HDR_1K 80
#HDR_2K 100
FrameTime 2000
LaserInten 100
CameraRoiSwitch 1
CameraRoiInit 0
CameraRoiHeight 1080
ClearEncoderCount 1
EncoderIntervalNum 1
EncoderCountMode 1


BatchDataValue 10


InterpolationFillingThr 10
FilterProcessTimeAxisMeanWin 3
FilterProcessTimeAxisMedianWin 3
FilterProcessXAxisMeanWin 3
FilterProcessXAxisMedianWin 7
```

8. AIRLink 类

8.1. AIRLink 类一览

8.1.1. 通信建立与相机控制

类名	AIScanner
所属文件	AIScanner.h
概要	搜索相机设备，连接/断开相机设备、相机基础参数设置、获取基础参数、数据获取
类成员函数	discoverCameras、connect(Info)、connect(IP)、disconnect; setParameterValue、getParameterValue; setHeartbeatInterval (暂未开放)、getCameraInfo、getCameraTemperature; start、stop、loadParameters、getModel、getSerial;

表 14 AIScanner

*** 下表为参数描述类:**

表 15 ScannerSetting

类名	ScannerSetting
所属文件	ScannerSetting.h
概要	DepthSight - L 系列相机的关键参数设置，多个类的集合
类集合	ParameterType（private）、DataMode、FrameRateTriggerMode、 WorkTriggerMode 、WorkExternalTriggerType、WorkExternalTriggerDelay、

	WorkExternalTriggerCaptureNums、WorkExternalTriggerCaptureRounds、ContourCorrectionSwitch、ContourCorrectionAngleRefer(暂未开放)、ContourCorrectionAngleMeasure(暂未开放)、ContourCorrectionHeightRefer(暂未开放)、ContourCorrectionHeightMeasure(暂未开放)、ExposureMode(仅支持单曝光)、ShortExposureTime、MidExposureTime(暂未开放)、LongExposureTime(暂未开放)、FrameTime、Gain、LaserInten、LaserMode、HDR_1K(暂未开放)、HDR_2K(暂未开放)、CameraRoiSwitch、CameraRoiInit、CameraRoiHeight、EncoderCountMode、EncoderIntervalNum、EncoderTriggerDelay、ClearEncoderCount、BatchDataCallBackSwitch、BatchDataValue、LaserIntensityThr、LineExtractMode(Max)、InterpolationFillingThr、FilterProcessTimeAxisMeanWin、FilterProcessTimeAxisMeadianWin、FilterProcessXAxisMeanWin、FilterProcessXAxisMeadianWin、HDRLevel;

8.1.2. 数据获取

类名	AIScanner
所属文件	AIScanner.h
概要	该类包含获取图像、获取点云批数据、回调注册、阻塞获取数据、获取数据长度等。
类成员	getSingleProfile、getSingleGray、getTheSameBatchData、setBatchDataHandler、getBatchDataThroughBlocking、getDataWidth、setContourCorrectionAngleHeight、registerDisconnectEventCallback、registerRoundStatusCallback;

表 16 AIScanner

8.1.3. 数据处理

类名	PostProcessing
所属文件	PostProcessing.h
概要	算法后处理类，包含对原始点云数据的解码、设置点云拼接参数、设置后处理算法（补间、平滑等）的参数等。
类成员	PostProcessing、DecodeProfilesXYZ、DecodeProfilesZ、setMoveDirection、getMoveDirection、resetProfileStitcher、ProfileStitch、setDistInterval

表 17 PostProcessing

8.1.4. 数据保存

类名	AIScanner
所属文件	AIScanner.h
概要	该类包含保存不同格式的点云数据的函数。
类成员	savePointCloudToPly(注意重载)、saveDataToCSV、saveDataToTIFF、saveGrayToCSV、saveGrayToTIFF

表 18 AIScanner

如上一节所示，PLY 格式保存三维点（可以选择是否去除无效值），CSV 格式按多行（帧）保存解析后的 z 值，TIFF 格式按多行（帧）保存未解析的 32 位整数 Z 值。

8.2. AIRLink 类参考

8.2.1. 通信建立与相机控制

类名::函数名	AIScanner::discoverCameras
所属文件	AIScanner.h
定义	`static std::vector<AIeveR_ScannerInfo> discoverCameras(unsigned int timeoutMs = 1000);`
描述	搜寻在线的设备。 **入参：** timeoutMs 搜寻超时时间，单位(ms)。当 timeoutMs 被设置为 0 时，会一直阻塞直到发现可用设备为止。 **返回值：** 已找到的在线设备信息集合

表 19 discoverCameras

类名::函数名	AIScanner::connect
所属文件	AIScanner.h
定义	`ErrorStatus connect(const AIeveR_ScannerInfo&info, unsigned int timeoutMs = 5000);`
描述	传入一个相机 info ，尝试连接该相机。 **入参：** info 相机信息； timeoutMs 连接超时时间，单位(ms)；当 timeoutMs 被设置为 0 时，该方法会一直阻塞直到成功连接相机为止。 **返回值：** ErrorStatus 错误状态。

表 20 connect(Info)

类名::函数名	AIScanner::connect
所属文件	AIScanner.h
定义	`ErrorStatus connect(const std::string& ip, unsigned int timeoutMs = 5000);`
描述	*注意与上一个同名函数区分入参。 传入一个相机 IP，尝试连接该相机。 **入参：** ip 相机 IP； timeoutMs 连接超时时间，单位(ms)；当 timeoutMs 被设置为 0 时，该方法会一直阻塞直到成功连接相机为止。 **返回值：** ErrorStatus 错误状态。

表 21 connect(IP)

类名::函数名	AIScanner::disconnect
所属文件	AIScanner.h

定义	`void disconnect();`
描述	无参，用于断开连接，并释放资源。 **注意：在程序结束之前必须调用 disconnect 接口，否则占用的资源无法释放，可能导致重启程序之后出现无法连接相机等问题。**

表 22 disconnect

类名::函数名	AIScanner::setParameterValue
所属文件	AIScanner.h
定义	`ErrorStatus getParameterValue(const std::string& parameterName, std::vector<int>& value);`
描述	设置参数（需要参考 SDK 使用手册或者命名空间 Scanner_Setting 中的不同参数定义类型，有不同的范围值） **入参：** parameterName 待设置的参数名； value 待设置的值。 **返回值：** ErrorStatus 错误状态。

支持项目	parameterName	value
数据模式 DataMode	PC PC_GRAY	0 1
触发模式 FrameRateFrameRateTriggerMode	FixedFrameRateTrigger //固定帧率触发 EncoderTrigger //编码器触发	0 1
曝光模式（仅支持 SingleExposure） ExposureMode	SingleExposure //单曝光 DualExposure //双曝光 HDR_1 //HDR1 HDR_2 //HDR2	0 1 2 3
触发工作模式 WorkTriggerMode	InternalTrigger //内部（软）触发开始工作 ExternalTrigger //外部触发开始工作	0 1
外部触发工作模式 WorkExternalTriggerType	SingleTerminalFixedType //单端子固定数目 SingleTerminalControlType //单端子控制 DoubleTerminalControlType //双端子控制	0 1 2
外部触发相机开始/结束工作延时 WorkExternalTriggerDelay	WorkExternalTriggerDelay //单位：us	
外部触发采集的固定数目 WorkExternalTriggerCaptureNums	WorkExternalTriggerCaptureNums	
外部触发采集轮数 WorkExternalTriggerCaptureRounds	WorkExternalTriggerCaptureRounds	
轮廓校正开关 ContourCorrectionSwitch	Bool //布尔值	0 1
轮廓校正角度参考值（该设置暂	ContourCorrectionAngleRefer	

	未开放） ContourCorrectionAngleRefer	//precision：1×10^{-5}，单位：degree	
	轮廓校正角度测量值（该设置暂未开放） ContourCorrectionAngleMeasure	ContourCorrectionAngleMeasure //precision：1×10^{-5}，单位：degree	
	轮廓校正高度参考值（该设置暂未开放） ContourCorrectionHeightRefer	ContourCorrectionHeightRefer //precision：1×10^{-5}，单位：mm	
	轮廓校正高度测量值（该设置暂未开放） ContourCorrectionHeightMeasure	ContourCorrectionHeightMeasure //precision：1×10^{-5}，单位：mm	
	短曝光时间 ShortExposureTime	ShortExposureTime //单位：us	(25, 90000)
	中曝光时间（该设置暂未开放） MidExposureTime	MidExposureTime //单位：us	(25, 90000)
	长曝光时间（该设置暂未开放） LongExposureTime	LongExposureTime //单位：us	(25, 90000)
	帧时间 FrameTime	FrameTime //用户设置，单位：us	
	相机增益模式 Gain	GainLevel_1 //Gain x1 GainLevel_2 //Gain x1.33 GainLevel_3 //Gain x2 GainLevel_4 //Gain x4	0 1 2 3
	激光强度 LaserInten	LaserInten	(0, 100)
	激光模式 LaserMode	Flashing //频闪 AlwaysOn //常量	0 1
	HDR 1K（该设置暂未开放） HDR_1K	HDR_1K	(0, 200)
	HDR 2K HDR_2K	HDR_2K	(0, 200)
	相机 ROI 开关 CameraRoiSwitch	CameraRoiSwitch //布尔值	1/0
	相机 ROI 起始行数 CameraRoiInit	CameraRoiInit	(0, 1072)
	相机 ROI 高度 CameraRoiHeight	CameraRoiHeight	(8, 1080)
	编码器计数模式 EncoderCountMode	mode_1_phase_1_inc //a pdge mode_2_phase_1_inc //a&b pdge mode_2_phase_2_inc //a&b pdge mode_2_phase_4_inc //a&b pdge	0 1 2 3

	编码器触发间隔 EncoderIntervalNum	EncoderIntervalNum //用户设置	详情见表格
	编码器计数清零 ClearEncoderCount	ClearEncoderCount //布尔值	1/0
	批处理开关 BatchDataCallBackSwitch	BatchDataCallBackSwitch //布尔值	1/0
	批处理行数 BatchDataValue	BatchDataValue //建议值：10	详情见表格
	中心线提取阈值 LaserIntensityThr	LaserIntensityThr	(0, 255)
	线提取模式 LineExtractMode	Max //目前只支持 MAX 模式 Near Far Continue Remove	0 1 2 3 4
	补间阈值 InterpolationFillingThr	InterpolationFillingThr	(0, 2048)
	时间轴平滑均值 FilterProcessTimeAxisMeanWin	_value_1 _value_3 _value_5 _value_9 _value_17	1 3 5 9 17
	时间轴平滑中值 FilterProcessTimeAxisMeadianWin	_value_1 _value_3 _value_5 _value_7 _value_9	1 3 5 7 9
	X 轴平滑均值 FilterProcessXAxisMeanWin	_value_1 _value_3 _value_5 _value_9 _value_17 _value_33	1 3 5 9 17 33
	X 轴平滑中值 FilterProcessXAxisMeadianWin	_value_1 _value_3 _value_5 _value_7 _value_9	1 3 5 7 9

表 23 setParameterValue

类名::函数名	AIScanner::getParameterValue
所属文件	AIScanner.h
定义	`ErrorStatus getParameterValue(const std::string& parameterName, std::vector<int>& value);`

描述

获取参数（需要参考命名空间 ScannerSetting 中的不同参数定义类型，调用不同的接口）

入参： parameterName 待设置的参数名；

value 待设置的值。

返回值： ErrorStatus 错误状态。

默认参数文件为 DefaultParameters.txt，放置于用户 bin 目录下，将下表中所需的参数配置。

支持项目	parameterName	value
数据模式 DataMode	PC PC_GRAY	0 1
触发模式 FrameRateFrameRateTriggerMode	FixedFrameRateTrigger //连续触发 EncoderTrigger //外部触发（例如编码器触发）	0 1
触发工作模式 WorkTriggerMode	InternalTrigger //内部触发开始工作 ExternalTrigger //外部触发开始工作	0 1
曝光模式（仅支持 SingleExposure、HDR_2） ExposureMode	SingleExposure //单曝光 DualExposure //双曝光 HDR_1 //HDR1 HDR_2 //HDR2	0 1 2 3
外部触发工作模式 WorkExternalTriggerType	SingleTerminalFixedType //单端子固定数目 SingleTerminalControlType //单端子控制 DoubleTerminalControlType //双端子控制	0 1 2
外部触发相机开始/结束工作延时 WorkExternalTriggerDelay	WorkExternalTriggerDelay //单位：us	
外部触发采集的固定数目 WorkExternalTriggerCaptureNums	WorkExternalTriggerCaptureNums	
外部触发采集轮数 WorkExternalTriggerCaptureRounds	WorkExternalTriggerCaptureRounds	
轮廓校正开关 ContourCorrectionSwitch	Bool //布尔值	0 1
轮廓校正角度参考值(该设置暂未开放) ContourCorrectionAngleRefer	ContourCorrectionAngleRefer //precision：1×10^{-5}，单位：degree	
轮廓校正角度测量值(该设置暂未开放) ContourCorrectionAngleMeasure	ContourCorrectionAngleMeasure //precision：1×10^{-5}，单位：degree	
轮廓校正高度参考值(该设置暂未开放) ContourCorrectionHeightRefer	ContourCorrectionHeightRefer //precision：1×10^{-5}，单位：mm	
短曝光时间 ShortExposureTime	ShortExposureTime //单位：us	(25, 90000)
中曝光时间（该设置暂未开放） MidExposureTime	MidExposureTime //单位：us	(25, 90000)

	长曝光时间（该设置暂未开放） LongExposureTime	LongExposureTime //单位：us	(25, 90000)
	帧时间 FrameTime	FrameTime //用户设置，单位：us	
	相机增益模式 Gain	GainLevel_1 //Gain x1 GainLevel_2 //Gain x1.33 GainLevel_3 //Gain x2 GainLevel_4 //Gain x4	0 1 2 3
	激光强度 LaserInten	LaserInten	(0, 100)
	激光模式 LaserMode	Flashing //频闪 AlwaysOn //常量	0 1
	HDR 1K（该设置暂未开放） HDR_1K	HDR_1K	(0, 200)
	HDR 2K（该设置暂未开放） HDR_2K	HDR_2K	(0, 200)
	相机 ROI 开关 CameraRoiSwitch	CameraRoiSwitch //布尔值	1/0
	相机 ROI 起始行数 CameraRoiInit	CameraRoiInit	(0, 1072)
	相机 ROI 高度 CameraRoiHeight	CameraRoiHeight	(8, 1080)
	编码器计数模式 EncoderCountMode	mode_1_phase_1_inc //a pdge mode_2_phase_1_inc //a&b pdge mode_2_phase_2_inc //a&b pdge mode_2_phase_4_inc //a&b pdge	0 1 2 3
	编码器触发间隔 EncoderIntervalNum	EncoderIntervalNum //用户设置	详情见表格
	编码器计数清零 ClearEncoderCount	ClearEncoderCount //布尔值	1/0
	批处理开关 BatchDataCallBackSwitch	BatchDataCallBackSwitch //布尔值	1/0
	批处理行数 BatchDataValue	BatchDataValue //建议值：10	详情见表格
	中心线提取阈值 LaserIntensityThr	LaserIntensityThr	(0, 255)
	线提取模式 LineExtractMode	Max //目前只支持 MAX 模式 Near Far Continue Remove	0 1 2 3 4
	补间阈值	InterpolationFillingThr	(0, 2048)

	InterpolationFillingThr		
	时间轴平滑均值 FilterProcessTimeAxisMeanWin	_value_1 _value_3 _value_5 _value_9 _value_17	1 3 5 9 17
	时间轴平滑中值 FilterProcessTimeAxisMeadianWin	_value_1 _value_3 _value_5 _value_7 _value_9	1 3 5 7 9
	X 轴平滑均值 FilterProcessXAxisMeanWin	_value_1 _value_3 _value_5 _value_9 _value_17 _value_33	1 3 5 9 17 33
	X 轴平滑中值 FilterProcessXAxisMeadianWin	_value_1 _value_3 _value_5 _value_7 _value_9	1 3 5 7 9

表 24 getParameterValue

类名::函数名	AIScanner::setHeartbeatInterval
所属文件	AIScanner.h
定义	`ErrorStatus setHeartbeatInterval(unsigned int timeIntervalMs);`
描述	设置上位机和设备之间的心跳（暂未开放） **入参：** timeIntervalMs 心跳间隔，单位(ms)。 **返回值：** ErrorStatus 错误状态。

表 25 setHeartbeatInterval

类名::函数名	AIScanner::getCameraInfo
所属文件	AIScanner.h
定义	`ErrorStatus getCameraInfo(AIeveR_ScannerInfo& info) const;`
描述	获取设备信息 **入/出参：** info 设备信息。 **返回值：** ErrorStatus 错误状态。

表 26 getCameraInfo

类名::函数名	AIScanner::getCameraTemperature
所属文件	AIScanner.h
定义	`ErrorStatus getCameraTemperature(float& temperature);`

描述	获取设备温度 **入参：** temperature 设备温度。 **返回值：** ErrorStatus 错误状态。

表 27 getCameraTemperature

类名::函数名	AIScanner::start
所属文件	AIScanner.h
定义	`ErrorStatus start();`
描述	无参，启动采集数据。（开始相机工作）

表 28 start

类名::函数名	AIScanner::stop
所属文件	AIScanner.h
定义	`ErrorStatus stop();`
描述	无参，停止采集数据，此时无法进行向相机下发参数等操作。（停止相机工作,效果同 disconnect）

表 29 stop

类名::函数名	AIScanner::loadParameters
所属文件	AIScanner.h
定义	`ErrorStatus loadParameters(const std::string& filePath);`
描述	相机参数配置(目前只支持 txt 文件)。特别说明：每次连接相机的时候不是必须下发默认参数了，用户根据需求配置所需参数。相机的会沿用上次断电时的参数，如果是第一次连接，则是出厂默认设置。 **入参：** filePath 保存相机参数的文件地址 **返回值：** ErrorStatus 错误状态。

表 30 loadParameters

类名	ParameterType（private）
所属文件	ScannerSetting.h

定义	`enum class ParameterType {`
	`_Int, ///< Integer type.`
	`_Float, ///< Double type.`
	`_Bool, ///< Boolean type.`
	`_Enum, ///< Enumeration type.`
	`_Roi, ///< %ROI type. 有关详细信息，请参阅@ref ROI。`
	`_Range, ///< %Range type. 有关详细信息，请参阅@ref Range。`
	`_FloatArray, ///< Vector of double types.`
	`};`

描述	枚举类，设置基础类型

表 31 ParameterType（private）

类名	DataMode
所属文件	ScannerSetting.h

定义	class DataMode{
	public:
	static constexpr const char* name = "DataMode";
	static constexpr const char* description = "description DataMode. ";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	PC, //0
	PC_GRAY //1
	};

描述	获取相机数据模式定义，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::DataMode::name, 0);（0/1 对应不同模式）

表 32 DataMode

类名	FrameRateTriggerMode
所属文件	ScannerSetting.h

定义	class TriggerMode{
	public:
	static constexpr const char* name = "FrameRateTriggerMode";
	static constexpr const char* description = "description FrameRateTriggerMode. ";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	FixedFrameRateTrigger, //固定频率触发：0
	EncoderTrigger //编码器触发：1
	};
	};

描述	设置相机触发模式，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::FrameRateTriggerMode::name, 0);（0/1 对应不同模式）

表 33 FrameRateTriggerMode

类名	WorkTriggerMode
所属文件	ScannerSetting.h

定义	class WorkTriggerMode
	{ public:
	static constexpr const char* name = "WorkTriggerMode";
	static constexpr const char* description = "description WorkTriggerMode. ";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	InternalTrigger, //内部触发开始工作
	ExternalTrigger //外部触发开始工作
	};
	};

描述	触发相机开始工作的模式描述，选择不同的触发模式，相机的触发条件和方式有所不同。本系列相机设有内部触发和外部触发两种模式。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: WorkTriggerMode::name, 0);（0/1 对应不同触发模式）

表 34 WorkTriggerMode

类名	WorkExternalTriggerType
所属文件	ScannerSetting.h

定义	class WorkExternalTriggerType
	{ public:
	static constexpr const char* name = "WorkExternalTriggerType";
	static constexpr const char* description = "description WorkExternalTriggerType. ";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	SingleTerminalFixedType, //单端子固定数目
	SingleTerminalControlType, //单端子控制
	DoubleTerminalControlType //双端子控制
	};
	};

描述	外部触发相机开始工作的类型描述，选择不同的触发模式，相机的触发条件和方式有所不同。本系列相机设有（单端子固定数目/单端子控制/双端子控制）三种模式。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: WorkExternalTriggerType::name, 0);（0/1/2 对应不同端子模式）

表 35 WorkExternalTriggerType

类名	WorkExternalTriggerDelay
所属文件	ScannerSetting.h
定义	class WorkExternalTriggerDelay { public: static constexpr const char* name = "WorkExternalTriggerDelay"; static constexpr const char* description = "description

	WorkExternalTriggerDelay";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr const char* unit = "μs";
	};

描述	设置外部触发相机开始工作和结束工作的延时长短。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::WorkExternalTriggerDelay::name, 100);

表 36 WorkExternalTriggerDelay

类名	WorkExternalTriggerCaptureNums
所属文件	ScannerSetting.h

定义	class WorkExternalTriggerCaptureNums
	{ public:
	static constexpr const char* name = "WorkExternalTriggerCaptureNums";
	static constexpr const char* description = "description WorkExternalTriggerCaptureNums";
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	设置外部触发相机开始工作采集的固定数目, 仅 External_Trigger_Type_==单端子固定数目时有效，默认 10 帧。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: WorkExternalTriggerCaptureNums::name, 2000);

表 37 WorkExternalTriggerCaptureNums

类名	WorkExternalTriggerCaptureRounds
所属文件	ScannerSetting.h

定义	class WorkExternalTriggerCaptureRounds
	{public:
	static constexpr const char* name = "WorkExternalTriggerCaptureRounds";
	static constexpr const char* description = "description WorkExternalTriggerCaptureRounds";
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	设置外部触发相机开始工作采集的轮数。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: WorkExternalTriggerCaptureRounds::name, 10);

表 38 WorkExternalTriggerCaptureRounds

类名	ContourCorrectionSwitch
所属文件	ScannerSetting.h
定义	class ContourCorrectionSwitch {public: static constexpr const char* name = "ContourCorrectionSwitch";

	static constexpr const char* description = "description ContourCorrectionSwitch";
	static constexpr ParameterType type = ParameterType::_Bool;
	};

描述	轮廓校正开关。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: ContourCorrectionSwitch::name, true);

表 39 ContourCorrectionSwitch

类名	ContourCorrectionAngleRefer
所属文件	ScannerSetting.h

定义	class ContourCorrectionAngleRefer
	{public:
	static constexpr const char* name = "ContourCorrectionAngleRefer";
	static constexpr const char* description = "description ContourCorrectionAngleRefer";
	static constexpr float precision = 0.00001; //input value = real_value/0.00001
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	轮廓校正角度参考值。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: ContourCorrectionAngleRefer::name, 2.3);

表 40 ContourCorrectionAngleRefer

类名	ContourCorrectionAngleMeasure
所属文件	ScannerSetting.h

定义	class ContourCorrectionAngleMeasure
	{public:
	static constexpr const char* name = "ContourCorrectionAngleMeasure";
	static constexpr const char* description = "description ContourCorrectionAngleMeasure";
	static constexpr float precision = 0.00001; //input value = real_value/0.00001
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	轮廓校正角度测量值。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: ContourCorrectionAngleMeasure::name, 2.3);

表 41 ContourCorrectionAngleMeasure

类名	ContourCorrectionHeightRefer
所属文件	ScannerSetting.h

定义	class ContourCorrectionHeightRefer
	{public:
	static constexpr const char* name = "ContourCorrectionHeightRefer";
	static constexpr const char* description = "description ContourCorrectionHeightRefer";
	static constexpr float precision = 0.00001; //input value = real_value/0.00001
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	轮廓校正高度参考值。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: ContourCorrectionHeightRefer::name, 2.3);

表 42 ContourCorrectionHeightRefer

类名	ContourCorrectionHeightMeasure
所属文件	ScannerSetting.h

定义	class ContourCorrectionHeightMeasure
	{public:
	static constexpr const char* name = "ContourCorrectionHeightMeasure";
	static constexpr const char* description = "description ContourCorrectionHeightMeasure";
	static constexpr float precision = 0.00001; //input value = real_value/0.00001
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	轮廓校正高度测量值。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: ContourCorrectionHeightRefer::name, 2.3);

表 43 ContourCorrectionHeightMeasure

类名	ExposureMode
所属文件	ScannerSetting.h

定义	class ExposureMode{
	public:
	static constexpr const char* name = "ExposureMode";
	static constexpr const char* description = "description ExposureMode. ";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	SingleExposure, //单曝光: 0
	DualExposure, //双曝光: 1
	HDR_1, //HDR1: 2
	HDR_2, //HDR2: 3
	};};

描述	该设置目前版本仅支持 SingleExposure 设置相机曝光模式，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::ExposureMode::name, 0); （0/1/2/3 对应不同模式）

表 44 ExposureMode

类名	ShortExposureTime
所属文件	ScannerSetting.h

定义	`class ShortExposureTime{`
	`public:`
	`static constexpr const char* name = "ShortExposureTime";`
	`static constexpr const char* description ="description ShortExposureTime";`
	`static constexpr ParameterType type = ParameterType::_Int;`
	`static constexpr Range<int> range() { return { 25, 90000 }; }`
	`static constexpr const char* unit = "μs";`
	`};`

描述	设置相机短曝光模式，设置范围（25, 90000），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 800);

表 45 ShortExposureTime

类名	MidExposureTime
所属文件	ScannerSetting.h

定义	`class MidExposureTime{`
	`public:`
	`static constexpr const char* name = "MidExposureTime";`
	`static constexpr const char* description = "description MidExposureTime";`
	`static constexpr ParameterType type = ParameterType::_Int;`
	`static constexpr Range<int> range() { return { 25, 90000 }; }`
	`static constexpr const char* unit = "μs";`
	`};`

描述	该设置暂未开放 设置相机中曝光模式，设置范围（25, 90000），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::MidExposureTime::name, 800);

表 46 MidExposureTime

类名	LongExposureTime
所属文件	ScannerSetting.h

定义	class LongExposureTime{
	public:
	static constexpr const char* name = "LongExposureTime";
	static constexpr const char* description = "description LongExposureTime";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 25, 90000 }; }
	static constexpr const char* unit = "μs";
	};

描述	该设置暂未开放 设置相机长曝光模式，设置范围（25, 90000），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::LongExposureTime::name, 800);

表 47 LongExposureTime

类名	FrameTime
所属文件	ScannerSetting.h

定义	class FrameTime{
	public:
	static constexpr const char* name = "FrameTime";
	static constexpr const char* description = "description FrameTime";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr const char* unit = "μs";
	};

描述	设置相机曝光时长，选择数据格式只有通过该类进行设置(帧时间大与短曝光时间+10us) 设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::FrameTime::name, 2000);

表 38 FrameTime

类名	Gain
所属文件	ScannerSetting.h

定义	class Gain{
	public:
	static constexpr const char* name = "Gain";
	static constexpr const char* description = "description Gain";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	GainLevel_1, //Gain x1
	GainLevel_2, //Gain x1.33
	GainLevel_3, //Gain x2

	GainLevel_4, //Gain x4 }; };
描述	设置相机模拟增益，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::Gain::name, 0);

表 48 Gain

类名	LaserInten
所属文件	ScannerSetting.h

定义	class LaserInten{
	public:
	static constexpr const char* name = "LaserInten";
	static constexpr const char* description = "description LaserInten";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 0, 100 }; }
	};

描述	设置相机激光强度，设置范围（0, 100），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 0);

表 49 LaserInten

类名	LaserMode
所属文件	ScannerSetting.h

定义	class LaserMode{
	public:
	static constexpr const char* name = "LaserMode";
	static constexpr const char* description = "description LaserMode";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	Flashing, //频闪
	AlwaysOn //常量
	};
	};

描述	设置相机激光工作模式，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserMode::name, 0);

表 50 LaserMode

类名	HDR_1K
所属文件	ScannerSetting.h

定义	class HDR_1K{
	public:
	static constexpr const char* name = "HDR_1K";
	static constexpr const char* description = "description HDR_1K < HDR_2K";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 0, 200 }; }
	};

描述	该设置暂未开放 设置相机 HDR_1K 输出模式，设置范围（0, 200），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::HDR_1K::name, 0);

表 51 HDR_1K

类名	HDR_2K
所属文件	ScannerSetting.h

定义	class HDR_2K{
	public:
	static constexpr const char* name = "HDR_1K";
	static constexpr const char* description = "description HDR_1K < HDR_2K";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 0, 200 }; }
	};

描述	该设置暂未开放 设置相机 HDR_2K 输出模式，设置范围（0, 200），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::HDR_2K::name, 0);

表 52 HDR_2K

类名	CameraRoiSwitch
所属文件	ScannerSetting.h

定义	class CameraRoiSwitch{
	public:
	static constexpr const char* name = "CameraRoiSwitch";
	static constexpr const char* description = "description CameraRoiSwitch";
	static constexpr ParameterType type = ParameterType::_Bool;
	};

描述	设置相机 ROI 开关，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::CameraRoiSwitch::name, true);（true 或者 false）

表 53 CameraRoiSwitch

类名	CameraRoiInit
所属文件	ScannerSetting.h

定义	class CameraRoiInit{
	public:
	static constexpr const char* name = "CameraRoiInit";
	static constexpr const char* description = "description CameraRoiInit";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 0, 1072 }; }
	};

描述	设置相机 ROI 初始化，设置范围（0, 1072），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::CameraRoiInit::name, 0);注意： 注意：累加步长为 8，该值必须是 8 的整数倍。 SDK 中 ROI 的计数方向：原点位于图像左上角，自上向下；而上位机软件中的 ROI 的计数方向：原点位于图像左中心，自上向下。

表 54 CameraRoiInit

类名	CameraRoiHeight
所属文件	ScannerSetting.h

定义	class CameraRoiHeight{
	public:
	static constexpr const char* name = "CameraRoiHeight";
	static constexpr const char* description = "description CameraRoiHeight";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 8, 1080 }; }
	};

描述	设置相机 ROI 高度初始化，设置范围（8, 1080），选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::CameraRoiHeight::name, 8); 注意：累加步长为 8，该值必须是 8 的整数倍。 SDK 中 ROI 的计数方向：原点位于图像左上角，自上向下；而上位机软件中的 ROI 的计数方向：原点位于图像左中心，自上向下。

表 55 CameraRoiHeight

类名	EncoderCountMode
所属文件	ScannerSetting.h
定义	class EncoderCountMode{ public: static constexpr const char* name = "EncoderCountMode"; static constexpr const char* description = "description EncoderCountMode"; static constexpr ParameterType type =

	ParameterType::_Enum;
	enum struct Value {
	mode_1_phase_1_inc, //a pdge
	mode_2_phase_1_inc, //a&b pdge
	mode_2_phase_2_inc, //a&b pdge
	mode_2_phase_4_inc, //a&b pdge
	};
	};

描述	设置电机编码器模式，选择数据格式只有通过该类进行设置，设置示例: auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::EncoderCountMode::name, 0); 编码器模式分为 4 种: 1 相 1 递增：编码器 A 相每次上升沿（A+）时递增，不可递减; 2 相 1 递增：编码器 A 相信号比 B 相提前 90 度且 A+时递增；编码器 A 相信号比 B 相滞后 90 度且 A-时递减。 2 相 2 递增：编码器 A 相信号比 B 相提前 90 度，A+或 A-均递增；编码器 A 相信号比 B 相滞后 90 度，A+或 A-均递减。 2 相 4 递增：编码器 A 相信号比 B 相提前 90 度，A+，A-，B+，B-均递增；编码器 A 相信号比 B 相滞后 90 度，A+，A-，B+，B-均递减。

表 56 EncoderCountMode

类名	EncoderIntervalNum
所属文件	ScannerSetting.h

定义	class EncoderIntervalNum{
	public:
	static constexpr const char* name = "EncoderIntervalNum";
	static constexpr const char* description = "description EncoderIntervalNum";
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	设置电机编码器触发相机的间隔，选择数据格式只有通过该类进行设置（如设定为 N,则每 N 次脉冲触发一次相机采集）。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::EncoderIntervalNum::name, 10);

表 57 EncoderIntervalNum

类名	EncoderTriggerDelay
所属文件	ScannerSetting.h

定义	class EncoderTriggerDelay {
	public:
	static constexpr const char* name = " EncoderTriggerDelay ";
	static constexpr const char* description = " description EncoderTriggerDelay";
	static constexpr ParameterType type = ParameterType::_Int;

	static constexpr const char* unit = "μs"; };
描述	设置电机编码器触发相机的延时，选择数据格式只有通过该类进行设置（如设定为 N,则每次编码器触发信号 N 微秒之后采集图像）。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting:: EncoderTriggerDelay::name, 10);

表 58 EncoderTriggerDelay

类名	ClearEncoderCount
所属文件	ScannerSetting.h

定义	class ClearEncoderCount{
	public:
	static constexpr const char* name = "ClearEncoderCount";
	static constexpr const char* description = "description ClearEncoderCount";
	static constexpr ParameterType type = ParameterType::_Bool;
	};

描述	设置是否清除电机编码器计数，选择数据格式只有通过该类进行设置，设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::ClearEncoderCount::name, true);

表 59 ClearEncoderCount

类名	BatchDataCallBackSwitch
所属文件	ScannerSetting.h

定义	class BatchDataCallBackSwitch{
	public:
	static constexpr const char* name = "BatchDataCallBackSwitch";
	static constexpr const char* description = "description BatchDataCallBackSwitch";
	static constexpr ParameterType type = ParameterType::_Bool;
	};

描述	数据回调批处理开关，选择数据格式只有通过该类进行设置，该设置用于区分批处理模式与单帧模式。布尔值 1 代表打开批处理，布尔值 0 代表关闭批处理，使用单帧。设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name, false);

表 60 BatchDataCallBackSwitch

类名	BatchDataValue

所属文件	ScannerSetting.h

定义	class BatchDataValue{
	public:
	static constexpr const char* name = "BatchDataValue";
	static constexpr const char* description = "description BatchDataValue";
	static constexpr ParameterType type = ParameterType::_Int;
	};

描述	设置数据回调批处理值，选择数据格式只有通过该类进行设置。例如设置连续模式，使用回调函数来获取点云数据，每次从缓存区获取的点云数据大小便是 BatchDataValue。 设置示例 auto error_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataValue::name,100);

表 61 BatchDataValue

类名	LaserIntensityThr
所属文件	ScannerSetting.h

定义	class LaserIntensityThr{
	public:
	static constexpr const char* name = "LaserIntensityThr";
	static constexpr const char* description = "description LaserIntensityThr";
	static constexpr ParameterType type = ParameterType::_Int;
	static constexpr Range<int> range() { return { 0, 255 }; }
	};

描述	线提取阈值。

表 62 LaserIntensityThr

类名	LineExtractMode
所属文件	ScannerSetting.h

定义	class LineExtractMode{
	public:
	static constexpr const char* name = "LineExtractMode";
	static constexpr const char* description = "description LineExtractMode";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	Max, //目前只支持 MAX 模式
	Near,
	Far,
	Continue,
	Remove
	};
	};

描述	线提取模式。

表 63 LineExtractMode

类名	InterpolationFillingThr
所属文件	ScannerSetting.h
定义	`class InterpolationFillingThr { public: static constexpr const char* name = "InterpolationFillingThr"; static constexpr const char* description = "description InterpolationFillingThr"; static constexpr ParameterType type = ParameterType::_Int; static constexpr Range<int> range() { return { 0, 2048 }; } };`
描述	补间阈值描述。

表 64 InterpolationFillingThr

类名	FilterProcessTimeAxisMeanWin
所属文件	ScannerSetting.h
定义	`class FilterProcessTimeAxisMeanWin{ public: static constexpr const char* name = "FilterProcessTimeAxisMeanWin"; static constexpr const char* description = "description FilterProcessTimeAxisMeanWin"; static constexpr ParameterType type = ParameterType::_Enum; enum struct Value { _value_1 = 1, _value_3 = 3, _value_5 = 5, _value_9 = 9, _value_17 = 17 }; };`
描述	时间轴平滑均值描述。

表 65 FilterProcessTimeAxisMeanWin

类名	FilterProcessTimeAxisMeadianWin
所属文件	ScannerSetting.h

定义	class FilterProcessTimeAxisMeadianWin{
	public:
	static constexpr const char* name = "FilterProcessTimeAxisMeadianWin";
	static constexpr const char* description = "description FilterProcessTimeAxisMeadianWin";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	_value_1 = 1,
	_value_3 = 3,
	_value_5 = 5,
	_value_7 = 7,
	_value_9 = 9,
	};
	};

描述	时间轴平滑中值描述。

表 66 FilterProcessTimeAxisMeadianWin

类名	FilterProcessXAxisMeanWin
所属文件	ScannerSetting.h

定义	class FilterProcessXAxisMeanWin{
	public:
	static constexpr const char* name = "FilterProcessXAxisMeanWin";
	static constexpr const char* description = "description FilterProcessXAxisMeanWin";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	_value_1 = 1,
	_value_3 = 3,
	_value_5 = 5,
	_value_9 = 9,
	_value_17 = 17,
	_value_33 = 33
	};
	};

描述	X 轴平滑均值描述。

表 67 FilterProcessXAxisMeanWin

类名	FilterProcessXAxisMeadianWin
所属文件	ScannerSetting.h
定义	class FilterProcessXAxisMeadianWin { public: static constexpr const char* name = "FilterProcessXAxisMeadianWin";

	static constexpr const char* description = "description FilterProcessXAxisMeadianWin";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	_value_1 = 1,
	_value_3 = 3,
	_value_5 = 5,
	_value_7 = 7,
	_value_9 = 9,
	};
	};

描述	X 轴平滑中值描述。

表 68 FilterProcessXAxisMeadianWin

类名	HDRLevel
所属文件	ScannerSetting.h

定义	class HDRLevel {
	public:
	static constexpr const char* name = "HDRLevel";
	static constexpr const char* description = "description HDRLevel";
	static constexpr ParameterType type = ParameterType::_Enum;
	enum struct Value {
	_level_1 = 0,
	_level_2 = 1,
	_level_3 = 2,
	};
	};

描述	HDR 等级设置。（曝光模式需要切换到 HDR 2K）

表 69 HDRLevel

8.2.2. 数据获取

类名::函数名	AIScanner::getSingleProfile
所属文件	AIScanner.h
定义	ErrorStatus getSingleProfile(uint32_t * pc_ptr_, unsigned int &pc_ptr_length_, unsigned int &frame_cnt_, unsigned int &encoder_value_, unsigned int wait_time_milliseconds = 100);
描述	获取一条点云轮廓数据 **入/出参：** pc_ptr_ 用来存储轮廓数据内存空间指针 pc_ptr_length_ 输出实际拷贝到 pc_ptr 内存空间的数据长度

	frame_cnt_vec 获取到的轮廓数据的帧号 encoder_value_ 获取到的轮廓数据的编码器值（-2147483647 ~ 2147483648） wait_time_milliseconds 获取数据超时时间，单位(ms)。当 timeoutMs 被设置为 0 时，该方法会一直阻塞直到获取一条最新数据为止。 **返回值：** ErrorStatus 错误状态。

表 69 getSingleProfile

类名::函数名	AIScanner::getSingleGray
所属文件	AIScanner.h

定义	`ErrorStatus getSingleGray(uint8_t * gray_ptr_,`
	`unsigned int &gray_ptr_length_,`
	`unsigned int &frame_cnt_,`
	`int &encoder_value_,`
	`unsigned int wait_time_milliseconds);`

描述	获取最新一组灰度值 **入/出参：** gray_ptr_ 用来存储灰度数据的内存空间指针（需要用户自己申请和管理） gray_ptr_length_ 输出实际拷贝到 gray_ptr 内存空间的数据长度 frame_cnt_ 获取到的灰度数据的帧号 encoder_value_ 获取到的轮廓数据的编码器值（-2147483647 ~ 2147483648） wait_time_milliseconds 获取数据超时时间，单位(ms)。当 timeoutMs 被设置为 0 时，该方法会一直阻塞直到获取一条最新数据为止。 **返回值：** ErrorStatus 错误状态。

表 70 getSingleGray

类名::函数名	AIScanner::getTheSameBatchData
所属文件	AIScanner.h

定义	`ErrorStatus getTheSameBatchData(uint32_t * pc_ptr_,`
	`uint8_t * gray_ptr_,`
	`unsigned int &frame_cnt_,`
	`int &encoder_value_,`
	`unsigned int wait_time_milliseconds);`

描述	获取最新一帧的数据，包含 Z 值+灰度。（该函数只在 DataMode 为 PC_GRAY 模式下生效） **入/出参：** pc_ptr_,用来存储轮廓数据的内存空间的指针（需要用户自己申请和管理） gray_ptr_ 用来灰度数据的内存空间的指针（需要用户自己申请和管理） frame_cnt_ 获取到的数据的帧号 encoder_value_ 获取到的轮廓数据的编码器值（-2147483647 ~ 2147483648） wait_time_milliseconds 获取数据超时时间，单位(ms)。当 timeoutMs 被设置为 0 时，该方法会一直阻塞直到获取一条最新数据为止。 **返回值：** ErrorStatus 错误状态。

	AIEVER_STATUS_SUCCESS：失败 AIEVER_STATUS_SUCCESS ： 成功

表 71 getTheSameBatchData

类名::函数名	AIScanner::setBatchDataHandler
所属文件	AIScanner.h
定义	`ErrorStatus setBatchDataHandler(AIeveR_BatchDataCallBack callBackFuntion,const int & count);`
描述	注册回调函数。注意：在丢帧的情况，回调函数获取到的数据行数下会小于设置回调时的期望行数。 **入参：** callBackFuntion 回调函数 count 每次返回的点云轮廓线的数量 **返回值：** ErrorStatus 错误状态。

表 72 setBatchDataHandler

类名::函数名	AIScanner::getBatchDataThroughBlocking
所属文件	AIScanner.h
定义	`ErrorStatus getBatchDataThroughBlocking(AIeveR_Data &data );`
描述	阻塞的方式获取批数据 **入/出参：** data 批数据输出 **返回值：** ErrorStatus 错误状态。

表 73 getBatchDataThroughBlocking

类名::函数名	AIScanner::getDataWidth
所属文件	AIScanner.h
定义	`ErrorStatus getDataWidth(int & dataWidth);`
描述	获取当前相机型号的数据宽度 **入/出参：** dataWidth 数据宽度输出 **返回值：** ErrorStatus 错误状态。

表 74 getDataWidth

类名::函数名	AIScanner:: setContourCorrectionAngleHeight
所属文件	AIScanner.h
定义	`ErrorStatus setContourCorrectionAngleHeight(const double& angleRefer, const double& angleMeasure, const double& heightRefer, const double& heightMeasure);`
描述	设置轮廓校正的高度/角度的参考值和测量值 **入参：** angleRefer 角度参考值 angleMeasure 角度测量值 ssheightRefer 高度参考值 heightMeasure 高度测量值 **返回值：** ErrorStatus 错误状态。

表 75 setContourCorrectionAngleHeight

类名::函数名	AIScanner:: registerDisconnectEventCallback
所属文件	AIScanner.h
定义	`ErrorStatus registerDisconnectEventCallback(const AIeveR_DisconnectEventCallBack& callback_func);`
描述	注册相机掉线事件 **入参：** callback_func 回调函数 **返回值：** ErrorStatus 错误状态。

表 76 registerDisconnectEventCallback

类名::函数名	AIScanner:: registerRoundStatusCallback
所属文件	AIScanner.h
定义	`ErrorStatus registerRoundStatusCallback(const AIeveR_RoundStatusCallBack& callback_func);`
描述	注册单轮采集状态回调事件 **入参：** callback_func 回调函数 回调函数入参含义: 1 ： 当前轮次开始 2 ： 当前轮次结束 **返回值：** ErrorStatus 错误状态。

表 77 registerRoundStatusCallback

8.2.3. 数据处理

类名::函数名	PostProcessing::PostProcessing
所属文件	PostProcessing.h
定义	`PostProcessing::PostProcessing(int work_distance);`
描述	数据后处理构造函数 **入参：** work_distance：相机工作距离（如 L10140 为 140mm，填入 140）。

表 76 PostProcessing

类名::函数名	PostProcessing::DecodeProfilesZ
所属文件	PostProcessing.h
定义	`ErrorStatus DecodeProfilesZ(const uint32_t* data_vec_ptr, Size_t data_vec_len, std::vector<float>& z_vec, size_t point_cloud_len);`
描述	3200 点 Z 值解码(uint32_t z -> float z，不可用于后续的拼接操作)，需要注意出参 point_cloud_vec 的数据结构 AIeveR_Point3F，如果需要用 OpenCV 处理数据请转换为对应数据类型。 另外，本软件提供了点云的拼接：profilestitch 1、直接对未解析的整数高度值进行拼接，得到 n * 3200 的点序列（或称为高度图

	像），（可选）将该整数点序列解析为浮点数点序列，可以保存为的 TIFF 文件（通常是整数）或 CSV 文件（通常是浮点数）。 2、先对每帧（如 3200 点）整数高度值进行解析，得到单帧的三维轮廓，再结合运动方向和编码器值/帧号进行点云拼接，得到完整的三维点云。后续可以保存为 PLY 格式的点云文件 **入参：** data_vec_ptr 需要解析的点云数据 data_vec_len 需要解析的点云数量 **出参：** point_cloud_ptr 输出解析好的点云数据 point_cloud_len 输出解析好的点云数量 **返回值：** ErrorStatus 错误状态。

表 77 DecodeProfilesZ

类名::函数名	PostProcessing::DecodeProfilesXYZ
所属文件	PostProcessing.h
定义	`ErrorStatus DecodeProfilesXYZ(const uint32_t* data_vec_ptr, size_t data_vec_len, std::vector<AIeveR_Point3F>& point_cloud_vec, size_t point_cloud_len);`
描述	3200 点 Z 值解码(uint32_t z -> float xyz，将 Z 值解析成三维数据，会自动补 xy，可用于后续的拼接操作)，自动补充 X 与 Y 的值。 - X 轴数据使用固定位置的 3200 个等间距 x 值进行填充，间距由相机型号确定（L11600: 0.5mm; L10400: 0.1mm; L10140: 0.035mm），且确保所有点的中心为原点（以 L10400 为例， 最小值为-159.95，最大值为 159.95）。 - Y 轴数据使用 0 填充，这是因为所有点位于激光平面上。 需要注意出参 point_cloud_vec 的数据结构 AIeveR_Point3F，如果需要用 OpenCV 处理数据请转换为对应数据类型。 另外，本软件提供了点云的拼接：profilestitch 1、直接对未解析的整数高度值进行拼接，得到 n * 3200 的点序列（或称为高度图像），（可选）将该整数点序列解析为浮点数点序列，可以保存为的 TIFF 文件（通常是整数）或 CSV 文件（通常是浮点数）。 2、先对每帧（如 3200 点）整数高度值进行解析，得到单帧的三维轮廓，再结合运动方向和编码器值/帧号进行点云拼接，得到完整的三维点云。后续可以保存为 PLY 格式的点云文件 **入参：** data_vec_ptr 需要解析的点云数据 data_vec_len 需要解析的点云数量 **出参：** point_cloud_ptr 输出解析好的点云数据 point_cloud_len 输出解析好的点云数量 **返回值：** ErrorStatus 错误状态。

表 78 DecodeProfilesXYZ

类名::函数名	PostProcessing:: setDistInterval
所属文件	PostProcessing.h

定义	`ErrorStatus setDistInterval (const double& dist,const bool& isEncoder);`
描述	解析 z 值，并自动补充 x, y 轴数据，构造位于激光平面坐标系的三维数序列，便于后续沿某一运动方向进行轮廓拼接（使用 AIeveR_Point3F 结构，可以方便转为 OpenCV 的 cv::Point3f 类型）。 x 轴数据使用固定位置的 3200 个等间距 x 值进行填充，间距由相机型号确定（L11600: 0.5mm; L10400: 0.1mm; L10140: 0.035mm），且确保所有点的中心为原点（以 L10400 为例， 最小值为-159.95，最大值为 159.95）。 y 轴数据使用 0 填充，这是因为所有点位于激光平面上（如 1 所示）。 **入参：** dist 期望设置的轮廓距离值 isEncoder 是否使用编码器拼接时的距离值。true:使用编码器拼接时的间距值。false:使用帧号拼接时的间距值。 **返回值：** ErrorStatus 错误状态。

表 79 setDistInterval

类名::函数名	PostProcessing::setMoveDirection
所属文件	PostProcessing.h
定义	`ErrorStatus setMoveDirection(AIeveR_Point3D vec3d );`
描述	设置移动方向（自动归一化）。与上表同名函数区分入参 **入参：** vec3d XYZ 三个方向量。 **返回值：** ErrorStatus 错误状态。

表 80 setMoveDirection

类名::函数名	PostProcessing::getMoveDirection
所属文件	PostProcessing.h
定义	`ErrorStatus getMoveDirection(AIeveR_Point3D vec3d );`
描述	获取移动方向。 **出参：** vec3d XYZ 三个方向量。 **返回值：** ErrorStatus 错误状态。

表 81 getMoveDirection

类名::函数名	PostProcessing::resetProfileStitcher
所属文件	PostProcessing.h
定义	`ErrorStatus resetProfileStitcher();`
描述	当进行带有方向和运动距离的点云拼接时，算法以第一帧为起点，固定第一帧的 Y 坐标为 0，X 坐标是 3200（仅依赖于相机型号），使用 resetProfileStitcher（手动重置拼接器），将使得拼接器重新以当前一帧开始拼接，忽略之前累计的运动距离。 **出参：** vec3d XYZ 三个方向量。 **返回值：** ErrorStatus 错误状态。

表 82 resetProfileStitcher

类名::函数名	PostProcessing::ProfileStitch
所属文件	PostProcessing.h
定义	`ErrorStatus ProfileStitch(ProfileStitcherParams& params,const bool& useEncoder);`
描述	使用数据帧号或者编码器进行拼接。 1、当进行带有方向和运动距离的点云拼接时，使用编码器的脉冲触发信号作为拼接条件进行拼接（仅在开启编码器拼接的情况下生效）。 2、当进行带有方向和运动距离的点云拼接时，使用每一帧的帧号作为拼接条件进行拼接（仅在不开启编码器拼接的情况下生效）。 另，本软件提供两种方式的拼接: 1、直接对未解析的整数高度值进行拼接，得到 n * 3200 的点序列（或称为高度图像），（可选）将该整数点序列解析为浮点数点序列，可以保存为的 TIFF 文件（通常是整数）或 CSV 文件（通常是浮点数）。 2、先对每帧（如 3200 点）整数高度值进行解析，得到单帧的三维轮廓，再结合运动方向和编码器值/帧号进行点云拼接,得到完整的三维点云,可以保存为 PLY 格式的点云文件。 **入参：** params 需要拼接的轮廓数据。 useEncoder 是否使用编码器拼接。 **返回值：** ErrorStatus 错误状态。

表 83 ProfileStitch

8.2.4. 数据保存

类名::函数名	AIScanner::savePointCloudToPly
所属文件	AIScanner.h
定义	`static ErrorStatus savePointCloudToPly(const std::vector<AIeveR_Point3F>& pcData, const std::vector<uint8_t>& grayData, const std::string& filePath , PlyFileFormat format = PlyFileFormat::BINARY , bool filterInvalidZ = true);`
描述	将获取的点云数据保存至 ply 文件 **入参：** pcData 需要保存的点云数据 grayData 需要保存的灰度图像数据 filePath 保存的文件路径 format PLY 文件的数据类型 filterInvalidZ 是否过滤无效 Z 值 **返回值：** ErrorStatus 错误状态。

表 84 savePointCloudToPly

类名::函数名	AIScanner::savePointCloudToPly(重载)
所属文件	AIScanner.h
定义	`static ErrorStatus savePointCloudToPly(const std::vector<AIeveR_Point3F>&pcData,`

	const std::vector<uint8_t>&grayData, const std::vector<int>& encoderData, const std::vector<unsigned int>& frameData, const std::string& filePath, PlyFileFormat format = PlyFileFormat::BINARY, bool filterInvalidZ = true);
描述	点云数据保存至 ply 文件（可保存编码器值和帧号） **入参：** pcData 需要保存的点云数据 grayData 需要保存的灰度图像数据 encoderData 点云对应的编码器值 frameData 点云对应的帧号值 filePath 保存的文件路径 format PLY 文件的数据类型 filterInvalidZ 是否过滤无效 Z 值 **返回值：** ErrorStatus 错误状态。

表 85 savePointCloudToPly

类名::函数名	AIScanner::saveDataToCSV
所属文件	AIScanner.h
定义	static ErrorStatus saveDataToCSV(const unsigned int* data,int dataSize, int dataWidth,const std::string filePath);
描述	将获取的点云数据保存至 csv 文件 先解析数据，再将数据保存至 csv 文件。每一行为一帧数据，第一列为帧索引，均从 1 开始编号。 以整数数据为 {0 , 0x007FFFFF , 0, 0x00800000}，每轮廓 2 个点为例,如果写入表头, csv 文件内容如下: profile, z_1, z_2 1, 0.00000, 1023.99988 2, 0.00000, -1024.00000 ..... **入参：** data 未解码的 32 位整数 z 值数据 dataSize 数据大小 dataWidth 数据宽度 filePath 保存的文件路径 **返回值：** ErrorStatus 错误状态。

表 86 saveDataToCSV

类名::函数名	AIScanner:: saveGrayToCSV
所属文件	AIScanner.h
定义	static ErrorStatus saveGrayToCSV(const std::vector<uint8_t>& grayData, int dataWidth, const std::string& filePath);
描述	获点云数据保存至 csv 文件。注意：数据宽度是一帧的数据宽度，可以通过 getDataWidth()方法获取。 **入参：** grayData 灰度数据 ss

	dataWidth 数据宽度 filePath 保存的文件路径 **返回值：** ErrorStatus 错误状态。

表 87 saveGrayToCSV

类名::函数名	AIScanner::saveDataToTIFF
所属文件	AIScanner.h
定义	`static ErrorStatus saveDataToTIFF(const unsigned int *data, int dataSize, int dataWidth, const std::string filePath);`
描述	将未解析的整数 z 值序列作为高度图像写入 TIFF 文件。注意：数据宽度是一帧的数据宽度，可以通过 getDataWidth()方法获取。 **入参：** pcData 需要保存的点云数据 dataSize 数据大小 dataWidth 数据宽度 filePath 保存的文件路径 **返回值：** ErrorStatus 错误状态。

表 88 saveDataToTIFF

类名::函数名	AIScanner::saveGrayToTIFF
所属文件	AIScanner.h
定义	`static ErrorStatus saveGrayToTIFF(const std::vector<uint8_t>& grayData, int dataWidth, const std::string& filePath);`
描述	灰度数据保存至 TIFF 文件 **入参：** grayData 灰度数据 dataWidth 数据宽度 filePath 保存的文件路径 **返回值：** ErrorStatus 错误状态。

表 89 saveGrayToTIFF

类名::函数名	AIScanner::getModel
所属文件	AIScanner.h
定义	`ErrorStatus getModel(std::string &model);`
描述	获取当前设备的型号 **入参：** model 设备型号 **返回值：** ErrorStatus 错误状态。

表 90 getModel

类名::函数名	AIScanner::getSerial
所属文件	AIScanner.h
定义	`ErrorStatus getSerial(std::string &serial);`

描述	获取当前设备的序列号 **入参：** serial 设备序列号 **返回值：** ErrorStatus 错误状态。

表 91 getSerial

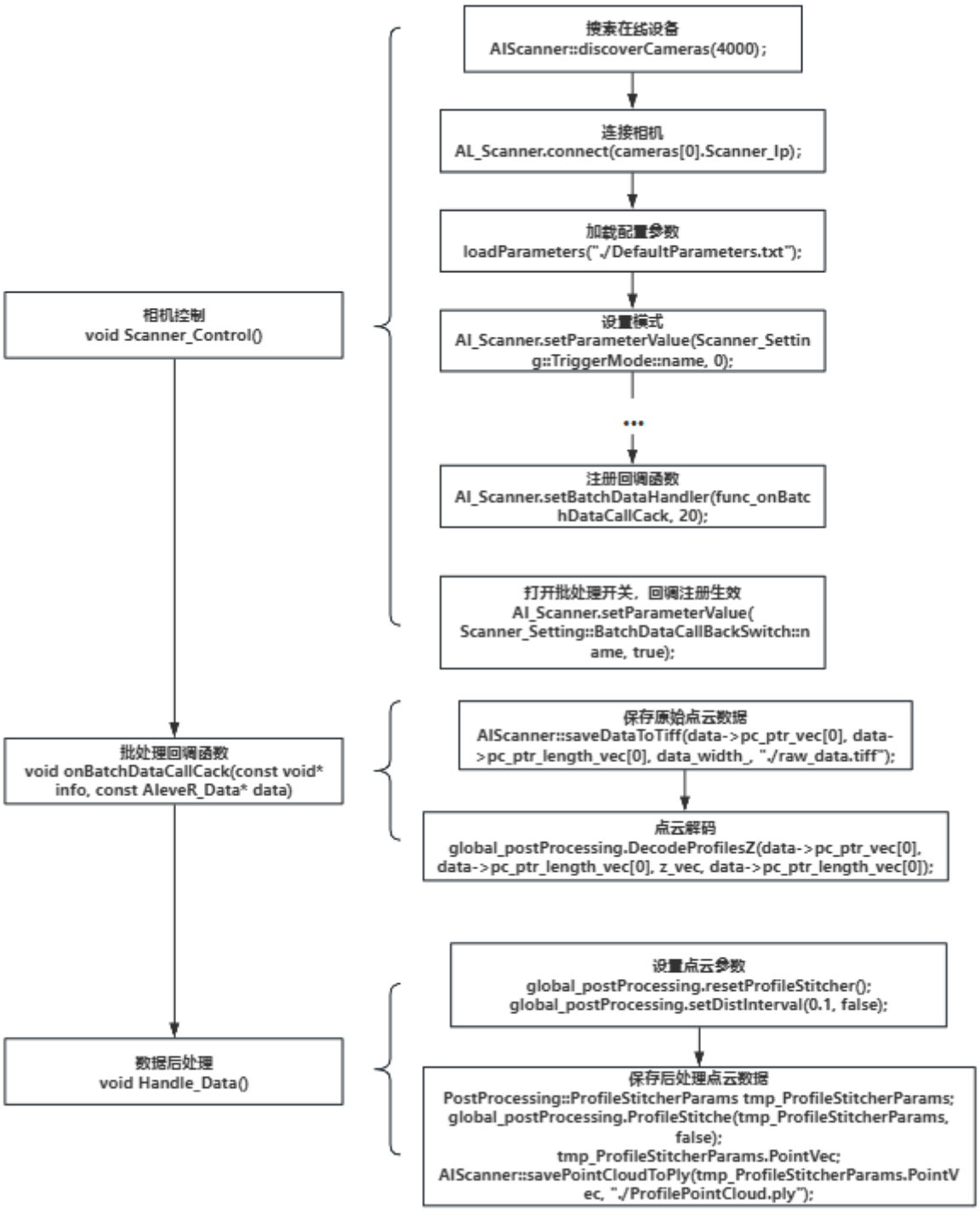
9. 接口函数调用示例

！注意相机控制最后一栏的区别！

9.1. （软触发+固定帧率触发）回调方式获取数据

基本流程图如下：

***由于每次获取到的当前帧都是最新的一帧，所以回调方式与单帧方式请勿同时使用，并且在获取单帧时，请同时获取并对齐帧号进行使用。**



示例代码：

```
#pragma once
#include "../include/AIeveR.h"
#include <iostream>
#include <fstream>
#include <thread>
#include "../include/progressbar.h"
using namespace AIeveR::Device;
using namespace AIeveR::Algorithm;

//全局变量声明
```

```
int callBackCount = 0;
int needCallbackCount = 100;
static std::vector<uint8_t> ALL_GRAY_VEC;
static std::vector<AIeveR_Point3F> ALL_PC_VEC;
static std::vector < int32_t> FRAME_VEC;
static std::vector < int32_t> ENCODER_VEC;
//实例化后处理类，这里 PostProcessing 类的构造函数入参需要区分不同的相机型号工作距离，此处使用的是 L10400 型号的相机。
//不同的相机型号参考使用手册，或者 Series_Name 类里不同型号的声明。
PostProcessing global_postProcessing(Series_Properties::series_L10140_workdist);
//批处理回调函数

//批处理回调函数
void onBatchDataCallCack(const void* info, const AIeveR_Data* data)
{
    //如果达到需要回调的次数，则不执行后续的代码
    if (callBackCount == needCallbackCount)
        {
            return;
        }
    //统计回调的次数
    callBackCount++;
    //每条线的点的数量 例如( L10400 : data_width = 3200)
    int data_width_ = data->data_width;


        //当前批次数据的行数
        int lineNums = data->encoder_value_vec.size();

        //如果你想保存当前批次原始 Z 值到 Tiff 文件
        //AIScanner::saveDataToTIFF(data->pc_ptr_vec[0], data->pc_ptr_length_vec[0], data_width_, "./raw_data.tif
f");
        //如果你想保存当前批次原始 Z 值到 CSV 文件
        //AIScanner::saveDataToCSV(data->pc_ptr_vec[0], data->pc_ptr_length_vec[0], data_width_, "./raw_data.csv"
);

        //手动申请下面轮廓数据解析需要用的内存空间
        std::vector<AIeveR_Point3F> PC_3200_VEC;

        //获取到的所有点的数量（ = lineNums * data_width_）
        int pointNum = data->pc_ptr_length_;
        PC_3200_VEC.reserve(data->pc_ptr_length_);

        //从相机获取到的整型数据解析为轮廓数据 (uint z -> float z)
        //这里解析出来的数据只有 z 值不能用于后续的拼接。
        std::vector<float> z_vec; //（用于存储解析后 Z 值的容器）
        global_postProcessing.DecodeProfilesZ(data->pc_ptr_, data->pc_ptr_length_, z_vec, data->pc_ptr_length_);
```

```
64.
65.         //从相机获取到的整型数据解析为轮廓数据 (uint z -> float xyz)
66.         //这里解析出来的是三维数据，可用于后续的拼接操作。
67.         global_postProcessing.DecodeProfilesXYZ(data->pc_ptr_, data->pc_ptr_length_, PC_3200_VEC, data->pc_ptr_le
    ngth_);
68.
69.         //这里将解析后所有的轮廓数据装入容器 ALL_PC_VEC。
70.         for (auto pc : PC_3200_VEC)
71.         {
72.             ALL_PC_VEC.push_back(pc);
73.         }
74.         //将每行数据对应的编码器值装入容器 ENCODER_VEC
75.         ENCODER_VEC.insert(ENCODER_VEC.end(), data->encoder_value_vec.begin(), data->encoder_value_vec.end());
76.
77.         //将每行数据对应的帧号值装入容器 FRAME_VEC
78.         FRAME_VEC.insert(FRAME_VEC.end(), data->frame_cnt_vec.begin(), data->frame_cnt_vec.end());
79.
80.         //灰度数据保存
81.         if (data->gray_ptr_length_ > 0)
82.         {
83.             ALL_GRAY_VEC.insert(ALL_GRAY_VEC.end(), data->gray_ptr_, data->gray_ptr_ + data->gray_ptr_length_);
84.         }
85.
86.         return;
87.
88.
89. }
90.
91. //数据后处理
92. void Free_Handle_Data()
93. {
94.         //重置后处理类的状态
95.         global_postProcessing.resetProfileStitcher();
96.
97.         //设置每帧之间的距离,这里我们设置为 0.1 毫米，false 代表设定的是帧号之间的距离值。
98.         global_postProcessing.setDistInterval(0.035, false);
99.
100.         //构造用于拼接的结构体
101.         PostProcessing::ProfileStitcherParams tmp_ProfileStitcherParams;
102.
103.         //传入需要拼接的点云数据
104.         tmp_ProfileStitcherParams.PointVec = ALL_PC_VEC;
105.
106.         //传入点云数据对应的帧号值
107.         tmp_ProfileStitcherParams.FlagValues = FRAME_VEC;
108.
109.         //开始拼接，并指定为帧号值拼接的方式。
```

```
110.         global_postProcessing.ProfileStitch(tmp_ProfileStitcherParams, false);
111.
112.         //得到拼接后的轮廓数据
113.         tmp_ProfileStitcherParams.PointVec;
114.
115.         //如果你想保存拼接后的点云
116.         AIScanner::savePointCloudToPly(tmp_ProfileStitcherParams.PointVec,ALL_GRAY_VEC, "./ProfilePointCloud_Free
     Callback.ply");
117. }
118.
119. //相机控制，回调的方式获取批数据（固定频率触发）
120. void Free_Scanner_Control()
121. {
122.         //实例化相机控制类
123.         AIScanner AI_Scanner;
124.
125.         //搜索在线相机
126.         std::vector<AIeveR_ScannerInfo> cameras = AIScanner::discoverCameras(1000 * 2);
127.
128.         //如果有搜索到相机则尝试连接
129.         if (cameras.size() > 0)
130.         {
131.             //假设连接第一个
132.             ErrorStatus connect_status = AI_Scanner.connect(cameras[0]);
133.
134.             //如果成功连接，则开始控制相机
135.             if (connect_status.isOK())
136.             {
137.                 //配置相机的参数，这里通过读取 txt 文件统一读入并下发。（txt 文件的详细说明请查阅使用手册）
138.                 ErrorStatus load_status = AI_Scanner.loadParameters("./DefaultParameters.txt");
139.
140.                 //成功配置相机默认参数
141.                 if (load_status.isOK())
142.                 {
143.                     //while(1){//
144.
145.                     //设置为内部触发
146.                     ErrorStatus set_status =
147.                         AI_Scanner.setParameterValue(Scanner_Setting::WorkTriggerMode::name, 0);
148.                     //切换到固定频率触发模式
149.                     set_status = AI_Scanner.setParameterValue(Scanner_Setting::FrameRateTriggerMode::name, 0);
150.                     /*
151.                     对参数设置状态（set_status）的处理
152.                     ...
153.                     */
154.
155.                     //注册回调函数，并指定批处理行数为 20。
```

```
156.                    set_status = AI_Scanner.setBatchDataHandler(onBatchDataCallCack, 20);
157.                    /*
158.                    对参数设置状态（set_status）的处理
159.                    ...
160.                    */
161.
162.                    //手动修改批处理行数，此时批处理行数会变成 10，上面注册的回调函数每次收到的批处理行数也会变成 10.
163.                    //（如果你在注册了回调函数之后想手动修改批处理行数）
164.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataValue::name, 100);
165.                    /*
166.                    对参数设置状态（set_status）的处理
167.                    ...
168.                    */
169.
170.                    //开启批处理，这里需要开启后，注册的回调函数才会生效。
171.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name, tru
        e);
172.                    /*
173.                    对参数设置状态（set_status）的处理
174.                    ...
175.                    */
176.
177.                    //如需手动修改相机的某些参数，例如这里修改了相机的短曝光,设置为 800μs。
178.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
179.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 800);
180.                    /*
181.                    对参数设置状态（set_status）的处理
182.                    ...
183.                    */
184.
185.                    //如需手动修改相机的某些参数，例如这里修改了激光的强度,设置为 80。
186.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
187.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 80);
188.                    /*
189.                    对参数设置状态（set_status）的处理
190.                    ...
191.                    */
192.
193.                    //相机开始采集
194.                    ErrorStatus start_status = AI_Scanner.start();
195.                    if (start_status.isOK())
196.                    {
197.                        /*
198.                        用户可以自定义条件停止采集
199.                        */
200.                        while (1)
201.                        {
```

```
                            //触发了 n 次回调之后停止
                            if (callBackCount == needCallbackCount)
                            {
                                break;
                            }
                            _sleep(10);
                        }

                        //停止采集
                        AI_Scanner.stop();
                        AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name, false);
                        //处理获取到的数据
                        Free_Handle_Data();
                    }
            }
                //断开连接并释放资源
                AI_Scanner.disconnect();
            }
        }
}
```

9.2. （软触发+固定帧率触发）循环获取最新帧数据

基本流程图如下:

***由于每次获取到的当前帧都是最新的一帧，所以回调方式与单帧方式请勿同时使用，并且在获取单帧时，请同时获取并对齐帧号进行使用。**

示例代码：

```
#pragma once
#include "../include/AIeveR.h"
#include <iostream>
#include <fstream>
#include <thread>
#include "../include/progressbar.h"
#include "../include/pgbar/pgbar.hpp"
#include <opencv2/opencv.hpp>
using namespace AIeveR::Device;
using namespace AIeveR::Algorithm;

```

```
12.
13. //声明全局变量
14. //
15.
16. //数据指针
17. uint32_t* pc_3200_ptr = nullptr;
18.
19. //用户期望获取的数据帧数
20. unsigned int user_need_lines = 0;
21.
22. //每行数据的点数
23. int dataWidth = 0;
24.
25. //用于存储所有三维数据的容器
26. std::vector<AIeveR_Point3F> all_pc_vec;
27.
28.
29. //用于存放所有数据的对应帧号的容器
30. std::vector<int32_t> all_frame_vec;
31.
32. void Loop_Handle_Data(PostProcessing& post_processing)
33. {
34.         //将原始的 Z 值保存到 csv 和 tiff
35.         AIScanner::saveDataToCSV(pc_3200_ptr, user_need_lines * dataWidth, dataWidth, "./free_loop_single.csv
    ");
36.         AIScanner::saveDataToTIFF(pc_3200_ptr, user_need_lines * dataWidth, dataWidth, "./free_loop_single.ti
    ff");
37.
38.         //从相机获取到的整型数据解析为轮廓数据 (uint z -> float xyz)
39.         std::vector<AIeveR_Point3F> PC_3200_VEC;
40.         PC_3200_VEC.reserve(user_need_lines * dataWidth);
41.
42.         //对所有帧数据的 Z 值进行解析
43.         post_processing.DecodeProfilesXYZ(pc_3200_ptr, user_need_lines * dataWidth, PC_3200_VEC, user_need_li
    nes * dataWidth);
44.
45.         //如果你想保存未进行拼接的三维数据
46.         std::vector<uint8_t> all_gray_vec;    //此时没有灰度数据，我们传入一个空的灰度容器。
47.         AIScanner::savePointCloudToPly(PC_3200_VEC, all_gray_vec, "./NotStitcherProfilePointCloud_LoopSingle.
    ply");
48.
49.         //重置后处理类的状态
50.         post_processing.resetProfileStitcher();
51.
52.         //设置每帧之间的距离,这里我们设置为 0.1 毫米，false 代表设定的是帧号之间的距离值。
53.         post_processing.setDistInterval(0.035, false);
54.
```

```
        //构造用于拼接的结构体
        PostProcessing::ProfileStitcherParams tmp_ProfileStitcherParams;        //根据帧号拼接

        //传入需要拼接的点云数据
        tmp_ProfileStitcherParams.PointVec = PC_3200_VEC;

        //传入点云数据对应的帧号值
        tmp_ProfileStitcherParams.FlagValues = all_frame_vec;

        //开始拼接，并指定为帧号值拼接的方式。
        post_processing.ProfileStitch(tmp_ProfileStitcherParams, false);

        //得到拼接后的轮廓数据
        tmp_ProfileStitcherParams.PointVec;

        //如果你想保存拼接后的点云
        std::vector<uint8_t> all_gray_vec_2;  //此时没有灰度数据，我们传入一个空的灰度容器。
        AIScanner::savePointCloudToPly(tmp_ProfileStitcherParams.PointVec, all_gray_vec_2, "./ProfilePointCloud_LoopSingle.ply");

        //释放内存空间
        delete[]pc_3200_ptr;
        pc_3200_ptr = nullptr;
}


//固定频率触发（循环获取最新帧的形式）
void Loop_Scanner_Control()
{

        //实例化相机控制类
        AIScanner AI_Scanner;

        //搜索在线相机
        std::vector<AIeveR_ScannerInfo> cameras = AIScanner::discoverCameras(1000 * 2);

        //如果有搜索到相机则尝试连接
        if (cameras.size() > 0)
        {
            //假设连接第一个
            ErrorStatus connect_status = AI_Scanner.connect(cameras[0]);

            //如果成功连接，则开始控制相机
            if (connect_status.isOK())
            {
                //配置相机的参数，这里通过读取 txt 文件统一读入并下发。（txt 文件的详细说明请查阅使用手册）
                ErrorStatus load_status = AI_Scanner.loadParameters("./DefaultParameters.txt");
```

```
101.
102.                //成功配置相机默认参数
103.                if (load_status.isOK())
104.                {
105.                    //设置为内部触发
106.                    ErrorStatus set_status =
107.                    AI_Scanner.setParameterValue(Scanner_Setting::WorkTriggerMode::name, 0);
108.                    //切换到固定频率触发模式
109.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::TriggerMode::name, 0);
110.                    /*
111.                    对参数设置状态（set_status）的处理
112.                    ...
113.                    */
114.
115.                    //如需手动修改相机的某些参数，例如这里修改了相机的短曝光,设置为 800μs。
116.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
117.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 50);

118.                    /*
119.                    对参数设置状态（set_status）的处理
120.                    ...
121.                    */
122.
123.                    //如需手动修改相机的某些参数，例如这里修改了激光的强度,设置为 80。
124.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
125.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 80);
126.                    /*
127.                    对参数设置状态（set_status）的处理
128.                    ...
129.                    */
130.
131.                    //相机开始采集
132.                    ErrorStatus start_status = AI_Scanner.start();
133.                    if (start_status.isOK())
134.                    {
135.                        //获取数据宽度
136.                        AI_Scanner.getDataWidth(dataWidth);
137.
138.                        //实例化后处理类，这里 PostProcessing 类的构造函数入参需要区分不同的相机型号.
139.                        //不同的相机型号参考使用手册，或者 Series_Name 类里不同型号的声明。
140.                        PostProcessing post_processing(cameras[0].Working_Distance);        //
141.
142.                        //用于计数和保存当前帧号的临时变量
143.                        int getCount = 0;
144.                        unsigned int tmpFrame = 0;
145.
146.                        //假设你需要获取到 2000 帧的数据
```

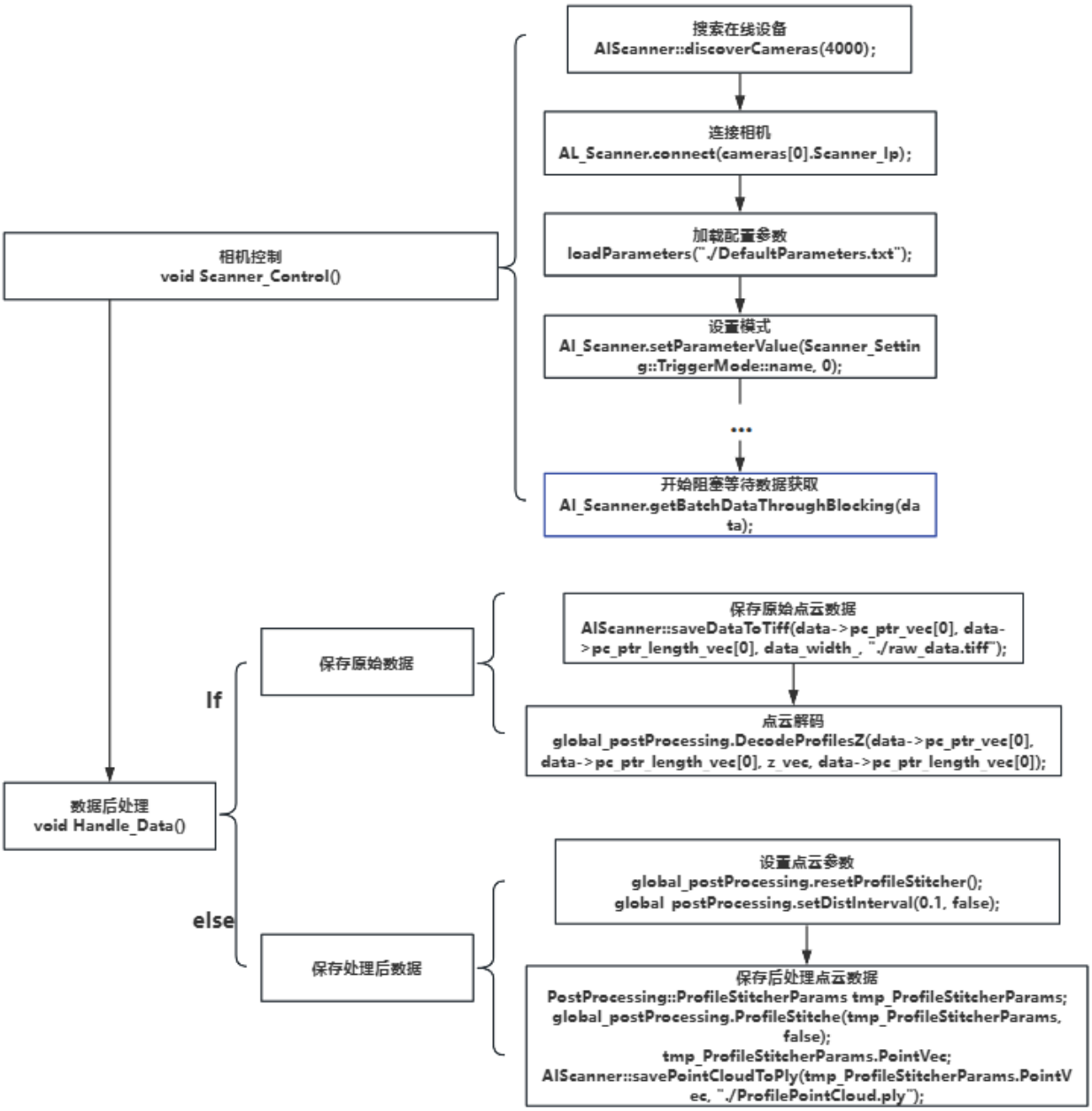
```
147.                        user_need_lines = 2000;
148.                        //申请内存空间
149.                        pc_3200_ptr = new uint32_t[dataWidth * user_need_lines];
150.                        /*
151.                        用户可以自定义条件停止采集
152.                        */
153.
154.                        while (1)
155.                        {
156.                            //数据存储内存初始化
157.                            unsigned int pc3200_length{ 0 };
158.                            unsigned int frame{ 0 };
159.                             int encoder{ 0 };
160.
161.                            // 申请临时内存
162.                            uint32_t*  ptr_vec{ new uint32_t[dataWidth] };
163.
164.                            //开始尝试获取当前最新帧（如果获取过快有可能连续几次循环获取到的都是同一帧数据）
165.                            auto err = AI_Scanner.getSingleProfile(ptr_vec, pc3200_length, frame, encoder, 1
    00);
166.
167.                            //成功获取到当前最新帧的数据并且不是上次获取到的帧号
168.                            if (err.isOK() && tmpFrame != frame)
169.                            {
170.                                //更新当前获取到的帧号
171.                                tmpFrame = frame;
172.
173.                                //装入帧号值容器
174.                                all_frame_vec.push_back(frame);
175.
176.                                //内存拷贝
177.                                memcpy(pc_3200_ptr + (getCount * dataWidth), ptr_vec, sizeof(uint32_t) * dat
    aWidth);
178.
179.                                //释放临时内存空间
180.                                delete[]ptr_vec;
181.                                ptr_vec = nullptr;
182.
183.                                //更新获取到的帧号计数
184.                                getCount++;
185.
186.                                //获取到期望的数据量后退出
187.                                if (getCount == user_need_lines){
188.                                    std::cout << "采集完毕...\n";
189.                                    break;
190.                                }
191.                            }
```

```
                        }
                        //停止采集
                        AI_Scanner.stop();
                        //数据后处理
                        Loop_Handle_Data(post_processing);
                    }
                }
                //断开连接并释放资源
                AI_Scanner.disconnect();
            }
        }
}
```

9.3. （软触发+编码器触发）阻塞方式获取批数据

基本流程图如下：

***由于每次获取到的当前帧都是最新的一帧，所以回调方式与单帧方式请勿同时使用，并且在获取单帧时，请同时获取并对齐帧号进行使用。**



示例代码:

```
#pragma once
#include "../include/AIeveR.h"
#include <iostream>
#include <fstream>
#include <thread>
#include "../include/progressbar.h"
using namespace AIeveR::Device;
using namespace AIeveR::Algorithm;

void Block_Handle_Data(const AIScanner &scanner, const AIeveR_Data &data)
{
        //每条线的点的数量 例如( L10400 : data_width = 3200)
        int data_width_ = data.data_width;

            //当前批次数据的行数
```

```
            int lineNums = data.encoder_value_vec.size();

            //如果你想保存当前批次原始 Z 值到 Tiff 文件
            AIScanner::saveDataToTIFF(data.pc_ptr_, data.pc_ptr_length_, data_width_, "./raw_data_BlckingCallback.tiff");
            //如果你想保存当前批次原始 Z 值到 CSV 文件
            AIScanner::saveDataToCSV(data.pc_ptr_, data.pc_ptr_length_, data_width_, "./raw_data_BlckingCallback.csv");

            //获取到的所有点的数量（ = lineNums * data_width_）
            int pointNum = data.pc_ptr_length_;

            //手动申请下面轮廓数据解析需要用的内存空间
            std::vector<AIeveR_Point3F> PC_3200_VEC;
            PC_3200_VEC.reserve(data.pc_ptr_length_);

            std::vector<uint8_t> GRAY_VEC;
            GRAY_VEC.resize(data.gray_ptr_length_);
            memcpy(GRAY_VEC.data(), data.gray_ptr_, data.gray_ptr_length_ * sizeof(uint8_t));

            //获取当前的相机工作距离
            AIeveR_ScannerInfo scanner_info;
            scanner.getCameraInfo(scanner_info);
            int work_dist =  scanner_info.Working_Distance;

            //实例化后处理类，这里 PostProcessing 类的构造函数入参需要区分不同的相机型号.
            //不同的相机型号参考使用手册，或者 Series_Name 类里不同型号的声明。
            PostProcessing post_processing(work_dist);

            //重置后处理类的状态
            post_processing.resetProfileStitcher();

            //设置每个编码器脉冲的距离,这里我们设置为 0.004 毫米，true 代表设置的时编码器脉冲的间距值。
            post_processing.setDistInterval(0.004, true);

            //从相机获取到的整型数据解析为轮廓数据 (uint z -> float xyz)
            //这里解析出来的是三维数据，可用于后续的拼接操作。
            post_processing.DecodeProfilesXYZ(data.pc_ptr_, data.pc_ptr_length_, PC_3200_VEC, data.pc_ptr_length_);

            //如果你想将未拼接的三维数据保存到 ply 文件
            AIScanner::savePointCloudToPly(PC_3200_VEC, GRAY_VEC,"./NotStitcherPointCloud_BlckingCallback.ply");

            //构造用于拼接的结构体
            PostProcessing::ProfileStitcherParams tmp_ProfileStitcherParams;

```

```
            //传入需要拼接的点云数据
            tmp_ProfileStitcherParams.PointVec = PC_3200_VEC;

            //传入点云数据对应的编码器值
            tmp_ProfileStitcherParams.FlagValues = data.encoder_value_vec;

            //开始拼接，并指定为以编码器值拼接的方式开始拼接。
            post_processing.ProfileStitch(tmp_ProfileStitcherParams, true);

            //得到拼接后的轮廓数据
            tmp_ProfileStitcherParams.PointVec;

            //如果你想保存拼接后的点云
            AIScanner::savePointCloudToPly(tmp_ProfileStitcherParams.PointVec, GRAY_VEC,"./ProfilePointCloud.
    ply");
}

//相机控制，阻塞的方式获取批数据（编码器触发）
void Block_Scanner_Control()
{

        //实例化相机控制类
        AIScanner AI_Scanner;

        //搜索在线相机
        std::vector<AIeveR_ScannerInfo> cameras = AIScanner::discoverCameras(1000 * 2);

        //如果有搜索到相机则尝试连接
        if (cameras.size() > 0)
        {
            //假设连接第一个
            ErrorStatus connect_status = AI_Scanner.connect(cameras[0]);

            //如果成功连接，则开始控制相机
            if (connect_status.isOK())
            {
                //配置相机的参数，这里通过读取 txt 文件统一读入并下发。（txt 文件的详细说明请查阅使用手册）
                ErrorStatus load_status = AI_Scanner.loadParameters("./DefaultParameters.txt");

                //成功配置相机默认参数
                if (load_status.isOK())
                {
                    //设置为内部触发
                    ErrorStatus set_status =
                    AI_Scanner.setParameterValue(Scanner_Setting::WorkTriggerMode::name, 0);
                    //切换到编码器触发模式
```

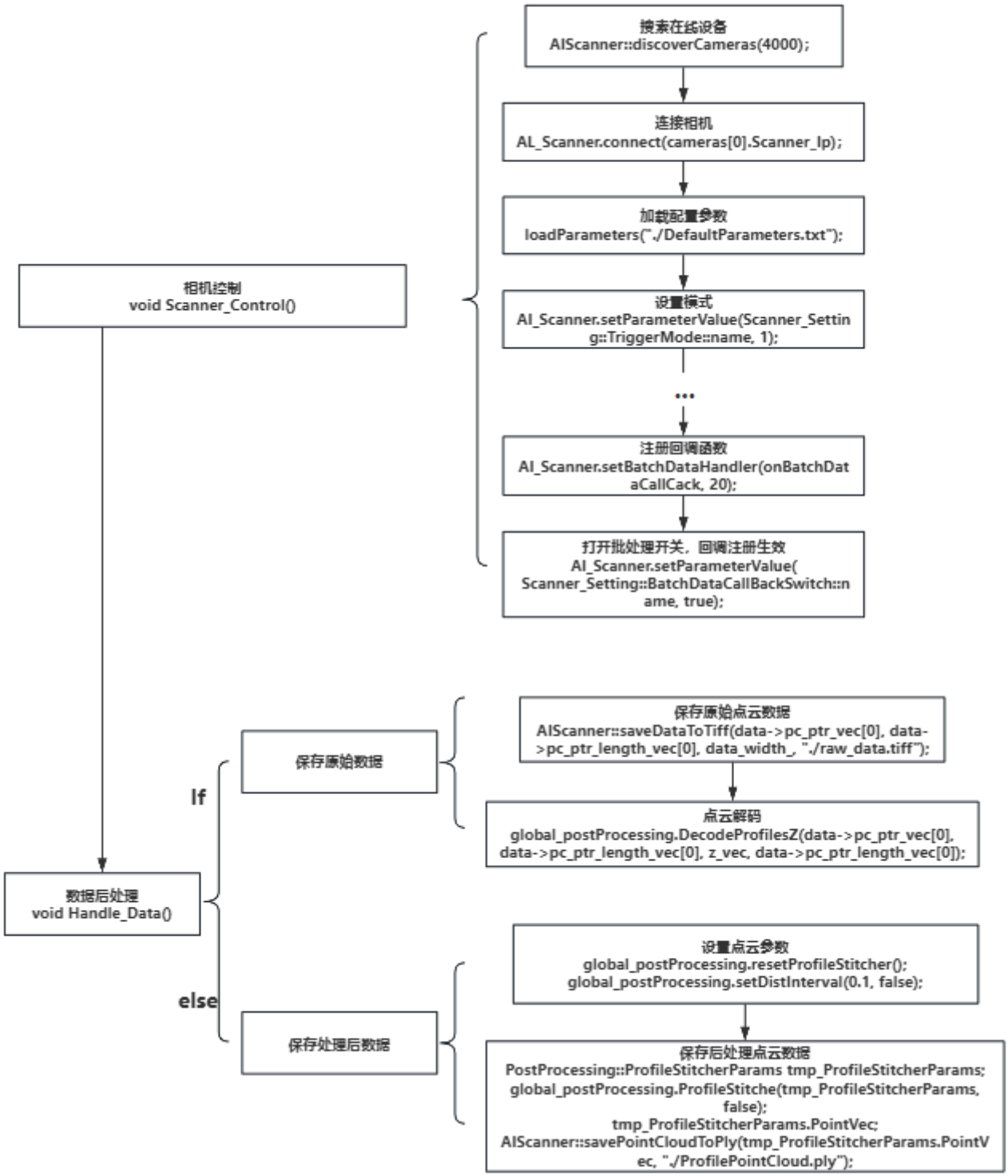
```
104.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::FrameRateTriggerMode::name, 1
    );
105.
106.                    //实际需要的数据行数
107.                    int needLines = 2000;
108.
109.                    //手动修改批处理行数，此时批处理行数会变成 2000。
110.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataValue::name, needLin
    es);
111.                    /*
112.                    对参数设置状态（set_status）的处理
113.                    ...
114.                    */
115.
116.                    //开启批处理
117.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name
    , true);
118.                    /*
119.                    对参数设置状态（set_status）的处理
120.                    ...
121.                    */
122.
123.                    //如需手动修改相机的某些参数，例如这里修改了相机的短曝光,设置为 800μs。
124.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
125.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 50);

126.                    /*
127.                    对参数设置状态（set_status）的处理
128.                    ...
129.                    */
130.
131.                    //如需手动修改相机的某些参数，例如这里修改了激光的强度,设置为 80。
132.                    //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
133.                    set_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 80);
134.                    /*
135.                    对参数设置状态（set_status）的处理
136.                    ...
137.                    */
138.
139.                    //相机开始采集
140.                    ErrorStatus start_status = AI_Scanner.start();
141.                    if (start_status.isOK())
142.                    {
143.
144.                        //实例化存储数据的结构体
145.                        AIeveR_Data data;
146.
```

```
                        //数据宽度
                        int dataWidth = 0;
                        AI_Scanner.getDataWidth(dataWidth);

                        //需要用申请存储点云和灰度的内存空间
                        data.pc_ptr_ = new uint32_t[needLines * dataWidth];
                        data.gray_ptr_ = new uint8_t[needLines * dataWidth];

                        //开始阻塞等待数据获取
                        ErrorStatus api_status  = AI_Scanner.getBatchDataThroughBlocking(data);
                        std::cout << "采集完毕...\n";
                        //停止采集
                        AI_Scanner.stop();

                        //如果能正常获取到数据，则处理获取到的数据(此处获取到数据的总行数 = BatchDataValue)
                        if (api_status.isOK())
                        {
                            Block_Handle_Data(AI_Scanner, data);
                        }
                    }
                }
                //断开连接并释放资源
                AI_Scanner.disconnect();
            }
        }
}
```

9.4. （软触发+编码器触发）回调方式获取批数据

基本流程图如下:

***由于每次获取到的当前帧都是最新的一帧，所以回调方式与单帧方式请勿同时使用，并且在获取单帧时，请同时获取并对齐帧号进行使用。**



示例代码：

```
#pragma once
#include "../include/AIeveR.h"
#include <iostream>
#include <fstream>
#include <thread>
#include "../include/progressbar.h"
using namespace AIeveR::Device;
using namespace AIeveR::Algorithm;

```

```
//全局变量声明
int callBackCount_ = 0;
int needCallbackCount_ = 100;
static std::vector<uint8_t> ALL_GRAY_VEC_;
static std::vector<AIeveR_Point3F> ALL_PC_VEC_;
static std::vector < uint32_t> FRAME_VEC_;
static std::vector < int32_t> ENCODER_VEC_;
int camera_data_width = 0;
//实例化后处理类，这里 PostProcessing 类的构造函数入参需要区分不同的相机型号工作距离，此处使用的是 L10400 型号的相机。
//不同的相机型号参考使用手册，或者 Series_Properties 类里不同型号的声明。
PostProcessing global_postProcessing_(Series_Properties::series_L10140_workdist);

// 批处理回调函数
void Encoder_onBatchDataCallCack(const void *info, const AIeveR_Data *data)
{
    //如果达到需要回调的次数，则不执行后续的代码
    if (callBackCount_ == needCallbackCount_)
        {
            return;
        }
        // 每条线的点的数量 例如( L10400 : data_width = 3200)
        int data_width_ = data->data_width;

        // 当前批次数据的行数
        int lineNums = data->encoder_value_vec.size();

        // 如果你想保存当前批次原始 Z 值到 Tiff 文件
        AIScanner::saveDataToTIFF(data->pc_ptr_, data->pc_ptr_length_, data_width_,
                                  "./raw_data_EncoderCallback.tiff");
        // 如果你想保存当前批次原始 Z 值到 CSV 文件
        AIScanner::saveDataToCSV(data->pc_ptr_, data->pc_ptr_length_, data_width_,
                                 "./raw_data_EncoderCallback.csv");

        // 手动申请下面轮廓数据解析需要用的内存空间
        std::vector<AIeveR_Point3F> PC_3200_VEC;

        // 获取到的所有点的数量（ = lineNums * data_width_）
        int pointNum = data->pc_ptr_length_;
        PC_3200_VEC.reserve(data->pc_ptr_length_);

        // 从相机获取到的整型数据解析为轮廓数据 (uint z -> float z)
        // 这里解析出来的数据只有 z 值不能用于后续的拼接。
        std::vector<float> z_vec; // （用于存储解析后 Z 值的容器）
        global_postProcessing_.DecodeProfilesZ(data->pc_ptr_, data->pc_ptr_length_,
                                               z_vec, data->pc_ptr_length_);

        // 从相机获取到的整型数据解析为轮廓数据 (uint z -> float xyz)
```

```
58.         // 这里解析出来的是三维数据，可用于后续的拼接操作。
59.         global_postProcessing_.DecodeProfilesXYZ(data->pc_ptr_, data->pc_ptr_length_,
60.                                                  PC_3200_VEC, data->pc_ptr_length_);
61.
62.         // 这里将解析后所有的轮廓数据装入容器 ALL_PC_VEC。
63.         ALL_PC_VEC_.insert(ALL_PC_VEC_.end(), PC_3200_VEC.begin(), PC_3200_VEC.end());
64.
65.
66.         // 将每行数据对应的编码器值装入容器 ENCODER_VEC
67.         ENCODER_VEC_.insert(ENCODER_VEC_.end(), data->encoder_value_vec.begin(),
68.                             data->encoder_value_vec.end());
69.
70.
71.         // 将每行数据对应的帧号值装入容器 FRAME_VEC
72.         FRAME_VEC_.insert(FRAME_VEC_.end(), data->frame_cnt_vec.begin(),
73.                           data->frame_cnt_vec.end());
74.
75.         //灰度数据保存
76.         if (data->gray_ptr_length_ > 0)
77.         {
78.             ALL_GRAY_VEC_.insert(ALL_GRAY_VEC_.end(), data->gray_ptr_,data->gray_ptr_+ data->gray_ptr_length_);
79.         }
80.
81.         // 统计回调的次数
82.         callBackCount_++;
83.
84.         return;
85.
86. }
87.
88. // 数据后处理
89. void Encoder_Handle_Data()
90. {
91.     // 重置后处理类的状态
92.     global_postProcessing_.resetProfileStitcher();
93.
94.     // 设置每个编码器脉冲之间的距离,这里我们设置为 0.004 毫米，true
95.     // 代表设定的是编码器脉冲之间的距离值。
96.     global_postProcessing_.setDistInterval(0.004, true);
97.     // 构造用于拼接的结构体
98.     PostProcessing::ProfileStitcherParams tmp_ProfileStitcherParams;
99.
100.    // 传入需要拼接的点云数据
101.    tmp_ProfileStitcherParams.PointVec = ALL_PC_VEC_;
102.
103.    // 传入点云数据对应的编码器值
104.    tmp_ProfileStitcherParams.FlagValues = ENCODER_VEC_;
```

```
105.
106.
107.     // 开始拼接，并指定为编码器值拼接的方式。
108.     ErrorStatus stitch_status = global_postProcessing_.ProfileStitch(tmp_ProfileStitcherParams, true);
109.
110.     // 如果你想保存灰度数据到 Tiff 文件
111.     // 数据宽度 data_width:每一帧的数据宽度
112.     AIScanner::saveGrayToTIFF(ALL_GRAY_VEC_, camera_data_width, "./raw_gray_data_EncoderCallback.tiff");
113.     // 如果你想保存灰度数据到 CSV 文件
114.     // 数据宽度 data_width:每一帧的数据宽度
115.     AIScanner::saveGrayToCSV(ALL_GRAY_VEC_, camera_data_width, "./raw_gray_data_EncoderCallback.csv");
116.
117.     // 得到拼接后的轮廓数据:tmp_ProfileStitcherParams.PointVec;
118.
119.     // 如果你想保存拼接后的点云,灰度，编码器值,帧号（如果不保存帧号，可以申请空容器占位）
120.     static std::vector < uint32_t> FRAME_VEC_EMPTY;
121.     // 可以选择用以 ASCII 或者 BINARY 的方式保存，此 demo 是以二进制的方式保存
122.     // filterInvalidZ = true，表示过滤无效点云，先择保存无效点云则会保存完整的点云以便调试
123.     AIScanner::savePointCloudToPly(tmp_ProfileStitcherParams.PointVec, ALL_GRAY_VEC_, ENCODER_VEC_, FRAME_VEC_EMP
    TY, "./ProfilePointCloud_EncoderCallback.ply",PlyFileFormat::BINARY, true);
124. }
125.
126. // 相机控制，回调的方式获取批数据（编码器触发）
127. void Encoder_Scanner_Control()
128. {
129.     // 实例化相机控制类
130.     AIScanner AI_Scanner;
131.
132.     // 搜索在线相机
133.     std::vector<AIeveR_ScannerInfo> cameras = AIScanner::discoverCameras(1000 * 2);
134.
135.     // 如果有搜索到相机则尝试连接
136.     if (cameras.size() > 0)
137.     {
138.         // 假设连接第一个
139.         ErrorStatus connect_status = AI_Scanner.connect(cameras[0]);
140.
141.         // 如果成功连接，则开始控制相机
142.         if (connect_status.isOK())
143.         {
144.             // 配置相机的参数，这里通过读取 txt 文件统一读入并下发。（txt 文件的详细说明请查阅使用手册）
145.             ErrorStatus load_status = AI_Scanner.loadParameters("./DefaultParameters.txt");
146.
147.             // 成功配置相机默认参数
148.             if (load_status.isOK())
149.             {
150.                 //设置为内部触发
```

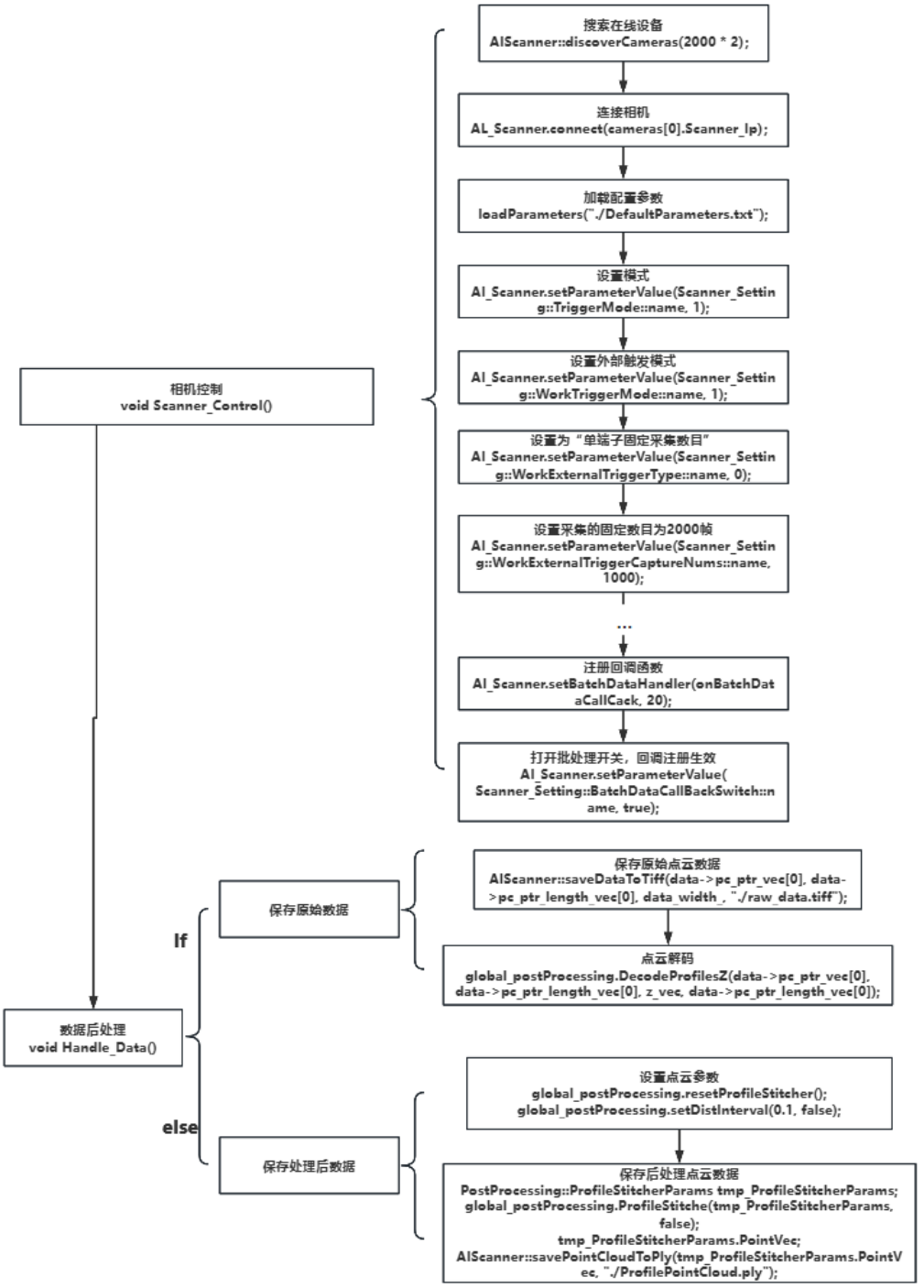
```
151.                ErrorStatus set_status =
152.                    AI_Scanner.setParameterValue(Scanner_Setting::WorkTriggerMode::name, 0);
153.                // 切换到编码器触发模式
154.                set_status =
155.                    AI_Scanner.setParameterValue(Scanner_Setting::FrameRateTriggerMode::name, 1);
156.                //获取相机数据宽度
157.                AI_Scanner.getDataWidth(camera_data_width);
158.                /*
159.                对参数设置状态（set_status）的处理
160.                ...
161.                */
162.
163.                // 注册回调函数，并指定批处理行数为 20。
164.                set_status = AI_Scanner.setBatchDataHandler(Encoder_onBatchDataCallCack, 20);
165.                /*
166.                对参数设置状态（set_status）的处理
167.                ...
168.                */
169.
170.                // 手动修改批处理行数，此时批处理行数会变成 10，上面注册的回调函数每次收到的批处理行数也会变成 10.
171.                //（如果你在注册了回调函数之后想手动修改批处理行数）
172.                set_status =
173.                    AI_Scanner.setParameterValue(Scanner_Setting::BatchDataValue::name, 10);
174.                /*
175.                对参数设置状态（set_status）的处理
176.                ...
177.                */
178.
179.                // 开启批处理，这里需要开启后，注册的回调函数才会生效。
180.                set_status = AI_Scanner.setParameterValue(
181.                    Scanner_Setting::BatchDataCallBackSwitch::name, true);
182.                /*
183.                对参数设置状态（set_status）的处理
184.                ...
185.                */
186.
187.                // 如需手动修改相机的某些参数，例如这里修改了相机的短曝光,设置为 800μs。
188.                //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
189.                set_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 800);
190.
191.                /*
192.                对参数设置状态（set_status）的处理
193.                ...
194.                */
195.
196.
197.                // 如需手动修改相机的某些参数，例如这里修改了激光的强度,设置为 80。
```

```
198.                 // （相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
199.                 set_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 80);
200.                 /*
201.                 对参数设置状态（set_status）的处理
202.                 ...
203.                 */
204.
205.                     std::cout << "Begin Collect." << std::endl;
206.                     // 相机开始采集
207.                     ErrorStatus start_status = AI_Scanner.start();
208.
209.                     // 开启批处理，这里需要开启后，注册的回调函数才会生效。
210.                     std::cout << start_status.isOK() << "\n";
211.                     if (start_status.isOK())
212.                     {
213.                         /*
214.                         用户可以自定义条件停止采集
215.                         */
216.                         while (1)
217.                         {
218.                             // 触发了 n 次回调之后停止
219.                             if (callBackCount_ == needCallbackCount_)
220.                             {
221.                                 break;
222.                             }
223.                             _sleep(10);
224.                         }
225.
226.                         // 停止采集
227.                         AI_Scanner.stop();
228.                         AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name, false);
229.
230.                         // 处理获取到的数据
231.                         Encoder_Handle_Data();
232.
233.                     }
234.
235.             }
236.             // 断开连接并释放资源
237.             AI_Scanner.disconnect();
238.         }
239.     }
240. }
```

9.5. （外部触发+编码器编码器触发）回调方式获取数据

基本流程图如下:

***由于每次获取到的当前帧都是最新的一帧，所以回调方式与单帧方式请勿同时使用，并且在获取单帧时，请同时获取并对齐帧号进行使用。**



代码如下：

```
#pragma once
#include "../include/AIeveR.h"
#include <iostream>
#include <fstream>
#include <thread>
#include "../include/progressbar.h"
using namespace AIeveR::Device;
using namespace AIeveR::Algorithm;

```

```
10.
11. //全局变量声明
12. int ex_callBackCount_ = 0;
13. int needex_callBackCount_ = 100;
14. static std::vector<uint8_t> ALL_GRAY_VEC_;
15. static std::vector<AIeveR_Point3F> ALL_PC_VEC_;
16. static std::vector < uint32_t> FRAME_VEC_;
17. static std::vector < int32_t> ENCODER_VEC_;
18. int data_width = 0;
19. //实例化后处理类，这里 PostProcessing 类的构造函数入参需要区分不同的相机型号工作距离，此处使用的是 L10400 型号的相机。
20. //不同的相机型号参考使用手册，或者 Series_Properties 类里不同型号的声明。
21. PostProcessing ex_global_postProcessing_(Series_Properties::series_L10140_workdist);
22.
23. // 批处理回调函数
24. void External_Encoder_onBatchDataCallCack(const void *info, const AIeveR_Data *data)
25. {
26.     //如果达到需要回调的次数，则不执行后续的代码
27.     if (ex_callBackCount_ == needex_callBackCount_)
28.         {
29.             return;
30.         }
31.         // 每条线的点的数量 例如( L10400 : data_width = 3200)
32.         int data_width_ = data->data_width;
33.
34.         // 当前批次数据的行数
35.         int lineNums = data->encoder_value_vec.size();
36.
37.         // 如果你想保存当前批次原始 Z 值到 Tiff 文件
38.         AIScanner::saveDataToTIFF(data->pc_ptr_, data->pc_ptr_length_, data_width_,
39.                                   "./raw_external_data_EncoderCallback.tiff");
40.         // 如果你想保存当前批次原始 Z 值到 CSV 文件
41.         AIScanner::saveDataToCSV(data->pc_ptr_, data->pc_ptr_length_, data_width_,
42.                                  "./raw_external_data_EncoderCallback.csv");
43.
44.         // 手动申请下面轮廓数据解析需要用的内存空间
45.         std::vector<AIeveR_Point3F> PC_3200_VEC;
46.
47.         // 获取到的所有点的数量（ = lineNums * data_width_）
48.         int pointNum = data->pc_ptr_length_;
49.         PC_3200_VEC.reserve(data->pc_ptr_length_);
50.
51.         // 从相机获取到的整型数据解析为轮廓数据 (uint z -> float z)
52.         // 这里解析出来的数据只有 z 值不能用于后续的拼接。
53.         std::vector<float> z_vec; // （用于存储解析后 Z 值的容器）
54.         ex_global_postProcessing_.DecodeProfilesZ(data->pc_ptr_, data->pc_ptr_length_,
55.                                                   z_vec, data->pc_ptr_length_);
56.
```

```
        // 从相机获取到的整型数据解析为轮廓数据 (uint z -> float xyz)
        // 这里解析出来的是三维数据，可用于后续的拼接操作。
        ex_global_postProcessing_.DecodeProfilesXYZ(data->pc_ptr_, data->pc_ptr_length_,
                                                    PC_3200_VEC, data->pc_ptr_length_);

        // 这里将解析后所有的轮廓数据装入容器 ALL_PC_VEC。
        ALL_PC_VEC_.insert(ALL_PC_VEC_.end(), PC_3200_VEC.begin(), PC_3200_VEC.end());


        // 将每行数据对应的编码器值装入容器 ENCODER_VEC
        ENCODER_VEC_.insert(ENCODER_VEC_.end(), data->encoder_value_vec.begin(),
                            data->encoder_value_vec.end());


        // 将每行数据对应的帧号值装入容器 FRAME_VEC
        FRAME_VEC_.insert(FRAME_VEC_.end(), data->frame_cnt_vec.begin(),
                          data->frame_cnt_vec.end());
        //灰度数据保存
        if (data->gray_ptr_length_ > 0)
        {
            ALL_GRAY_VEC_.insert(ALL_GRAY_VEC_.end(), data->gray_ptr_, data->gray_ptr_+ data->gray_ptr_length
          _);
        }

        // 统计回调的次数
        ex_callBackCount_++;

        return;

}

// 数据后处理
void External_Encoder_Handle_Data()
{
    // 重置后处理类的状态
    ex_global_postProcessing_.resetProfileStitcher();

    // 设置每个编码器脉冲之间的距离,这里我们设置为 0.004 毫米，true
    // 代表设定的是编码器脉冲之间的距离值。
    ex_global_postProcessing_.setDistInterval(0.004, true);
    // 构造用于拼接的结构体
    PostProcessing::ProfileStitcherParams tmp_ProfileStitcherParams;

    // 传入需要拼接的点云数据
     tmp_ProfileStitcherParams.PointVec = ALL_PC_VEC_;

     // 传入点云数据对应的编码器值
```

```
    tmp_ProfileStitcherParams.FlagValues = ENCODER_VEC_;


    // 开始拼接，并指定为编码器值拼接的方式。
    ErrorStatus stitch_status = ex_global_postProcessing_.ProfileStitch(tmp_ProfileStitcherParams, true);

    // 如果你想保存灰度数据到 Tiff 文件
    // 数据宽度 data_width:每一帧的数据宽度
    AIScanner::saveGrayToTIFF(ALL_GRAY_VEC_, data_width, "./raw_external_gray_data_EncoderCallback.tiff");
    // 如果你想保存灰度数据到 CSV 文件
    // 数据宽度 data_width:每一帧的数据宽度
    AIScanner::saveGrayToCSV(ALL_GRAY_VEC_, data_width, "./raw_external_gray_data_EncoderCallback.csv");

    // 得到拼接后的轮廓数据:tmp_ProfileStitcherParams.PointVec;

    // 如果你想保存拼接后的点云,灰度，编码器值,帧号（如果不保存帧号，可以申请空容器占位）
    static std::vector < uint32_t> FRAME_VEC_EMPTY;
    // 可以选择用以 ASCII 或者 BINARY 的方式保存，此 demo 是以二进制的方式保存
    // filterInvalidZ = true，表示过滤无效点云，先择保存无效点云则会保存完整的点云以便调试
    AIScanner::savePointCloudToPly(tmp_ProfileStitcherParams.PointVec, ALL_GRAY_VEC_, ENCODER_VEC_, FRAME_VE
        C_EMPTY, "./external_ProfilePointCloud_EncoderCallback.ply",PlyFileFormat::BINARY, true);
}

// 相机控制，回调的方式获取批数据（编码器触发）
void External_Encoder_Scanner_Control()
{
    // 实例化相机控制类
    AIScanner AI_Scanner;

    // 搜索在线相机
    std::vector<AIeveR_ScannerInfo> cameras = AIScanner::discoverCameras(1000 * 2);

    // 如果有搜索到相机则尝试连接
    if (cameras.size() > 0)
    {
        // 假设连接第一个
        ErrorStatus connect_status = AI_Scanner.connect(cameras[0]);

        // 如果成功连接，则开始控制相机
        if (connect_status.isOK())
        {
            // 配置相机的参数，这里通过读取 txt 文件统一读入并下发。（txt 文件的详细说明请查阅使用手册）
            ErrorStatus load_status = AI_Scanner.loadParameters("./DefaultParameters.txt");

            // 成功配置相机默认参数
            if (load_status.isOK())
            {
```

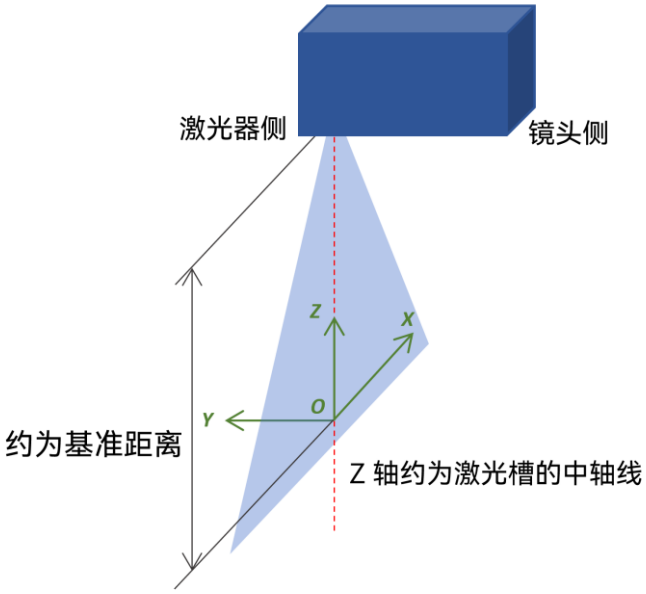
```
149.                //设置为外部触发
150.                ErrorStatus set_status =
151.                    AI_Scanner.setParameterValue(Scanner_Setting::WorkTriggerMode::name, 1);
152.                //外部触发设置：设置为“单端子固定采集数目”
153.                set_status =
154.                    AI_Scanner.setParameterValue(Scanner_Setting::WorkExternalTriggerType::name, 0);
155.                //外部触发设置：设置采集的固定数目为 2000 帧
156.                set_status =
157.                    AI_Scanner.setParameterValue(Scanner_Setting::WorkExternalTriggerCaptureNums::name, 1000
            );
158.                // 切换到编码器触发模式
159.                set_status =
160.                    AI_Scanner.setParameterValue(Scanner_Setting::FrameRateTriggerMode::name, 1);
161.                //获取相机数据宽度
162.                AI_Scanner.getDataWidth(data_width);
163.                /*
164.                对参数设置状态（set_status）的处理
165.                ...
166.                */
167.
168.                // 注册回调函数，并指定批处理行数为 20。
169.                set_status = AI_Scanner.setBatchDataHandler(External_Encoder_onBatchDataCallCack, 20);
170.                /*
171.                对参数设置状态（set_status）的处理
172.                ...
173.                */
174.
175.                // 手动修改批处理行数，此时批处理行数会变成 10，上面注册的回调函数每次收到的批处理行数也会变成 10.
176.                //（如果你在注册了回调函数之后想手动修改批处理行数）
177.                set_status =
178.                    AI_Scanner.setParameterValue(Scanner_Setting::BatchDataValue::name, 10);
179.                /*
180.                对参数设置状态（set_status）的处理
181.                ...
182.                */
183.
184.                // 开启批处理，这里需要开启后，注册的回调函数才会生效。
185.                set_status = AI_Scanner.setParameterValue(
186.                    Scanner_Setting::BatchDataCallBackSwitch::name, true);
187.                /*
188.                对参数设置状态（set_status）的处理
189.                ...
190.                */
191.
192.                // 如需手动修改相机的某些参数，例如这里修改了相机的短曝光,设置为 800μs。
193.                //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
194.                set_status = AI_Scanner.setParameterValue(Scanner_Setting::ShortExposureTime::name, 800);
```

```
195.
196.                 /*
197.                 对参数设置状态（set_status）的处理
198.                 ...
199.                 */
200.
201.
202.                 // 如需手动修改相机的某些参数，例如这里修改了激光的强度,设置为 80。
203.                 //（相机参数对应的具体参数名和范围值可以参考 Scanner_Setting 中的参数类或者参考 SDK 说明手册）
204.                 set_status = AI_Scanner.setParameterValue(Scanner_Setting::LaserInten::name, 80);
205.                 /*
206.                 对参数设置状态（set_status）的处理
207.                 ...
208.                 */
209.
210.                     std::cout << "Begin Collect." << std::endl;
211.                     // 相机开始采集
212.                     ErrorStatus start_status = AI_Scanner.start();
213.                     if (start_status.isOK())
214.                     {
215.                         /*
216.                         用户可以自定义条件停止采集
217.                         */
218.                         while (1)
219.                         {
220.                             // 触发了 n 次回调之后停止
221.                             if (ex_callBackCount_ == needex_callBackCount_)
222.                             {
223.                                 break;
224.                             }
225.                             _sleep(10);
226.                         }
227.
228.                         // 停止采集
229.                         AI_Scanner.stop();
230.                         AI_Scanner.setParameterValue(Scanner_Setting::BatchDataCallBackSwitch::name, false);
231.
232.                         // 处理获取到的数据
233.                         External_Encoder_Handle_Data();
234.
235.                     }
236.
237.             }
238.             // 断开连接并释放资源
239.             AI_Scanner.disconnect();
240.         }
```

```
241.     }
242. }
```

10. 附录

10.1.坐标系



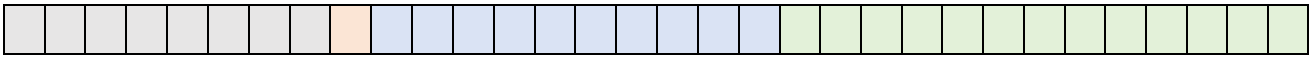
相机固有的坐标系如上图所示。其中:

- X 轴基本水平，且位于激光面内;
- Y 轴和激光平面的法向量平行，从镜头侧指向激光器侧;
- Z 轴垂直向上，且基本和激光槽的中轴线重合。

获取的数据默认是在此坐标系下。

10.2.数据解析

调取数据获取函数（如 getSingleProfile）后，获取的是未解析的定点数据，数据长度为 32 位，其中 24 位有效。示意图如下:



补 0	符号	整数部分	小数部分
8 位	1 位	10 位	13 位

能表示的最小数的 16 进制表示为 0x00800000，即-1024；最大数为 0x007FFFFF，约为 1023.99988.

解析方法为:

```
if (未解析的整数 > 8388608):
    解析结果 = (未解析的整数 - 16777216) / 8192
else:
    解析结果 = 未解析的整数 / 8192
```

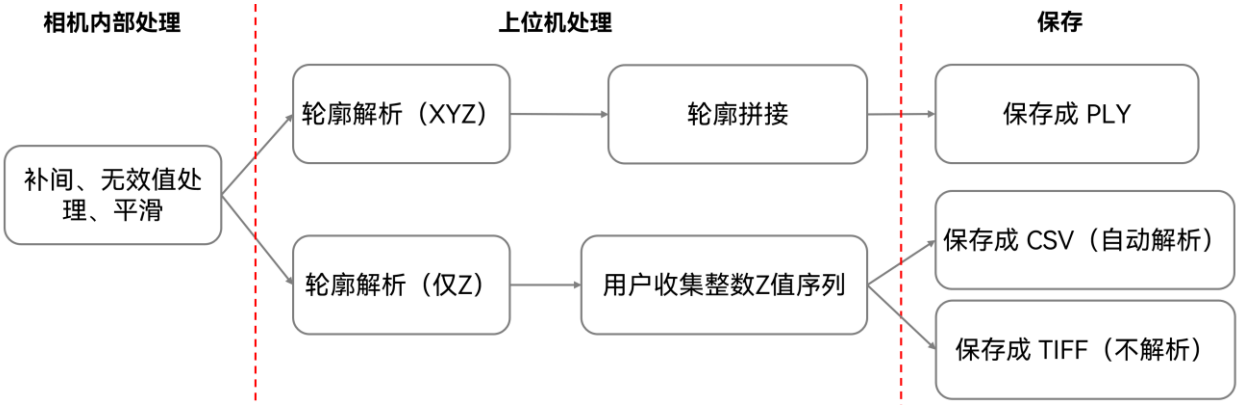
用户可以使用 SDK 中的 DecodeProfilesXYZ 函数或 DecodeProfilesZ 函数(在下一小节数据处理中介绍)，将数据解析成浮点（单位：毫米）的三维数据或 Z 值数据。

10.3.编码器值

使用 32 位 int 类型存储，实际有效范围为-2147483647 至 2147483648. 从 0 开始对脉冲计数，当增大至 2147483648，会回到-2147483647；同理，当减小到-2147483647 时，会回到 2147483648.

10.4.轮廓后处理

完整的后处理和保存流程如下：



下面介绍各部分的原理。

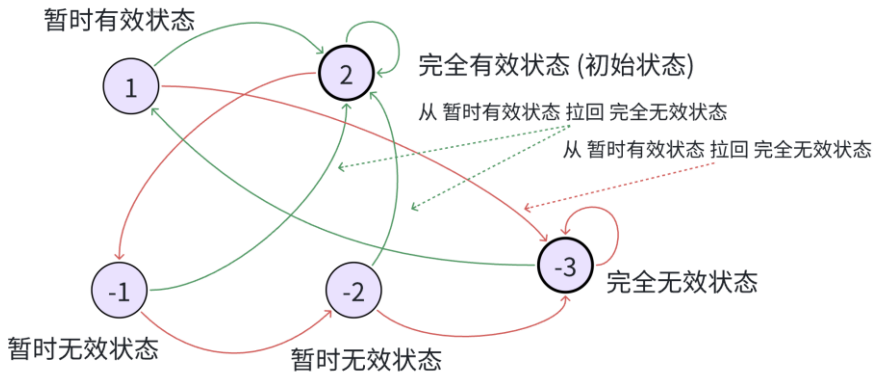
- **补间、无效值处理、平滑：**

  可以通过 SDK 配置相机回传图像之前需要进行的一些处理操作，以减小噪声的影响以及增强点云质量。方法列举如下。

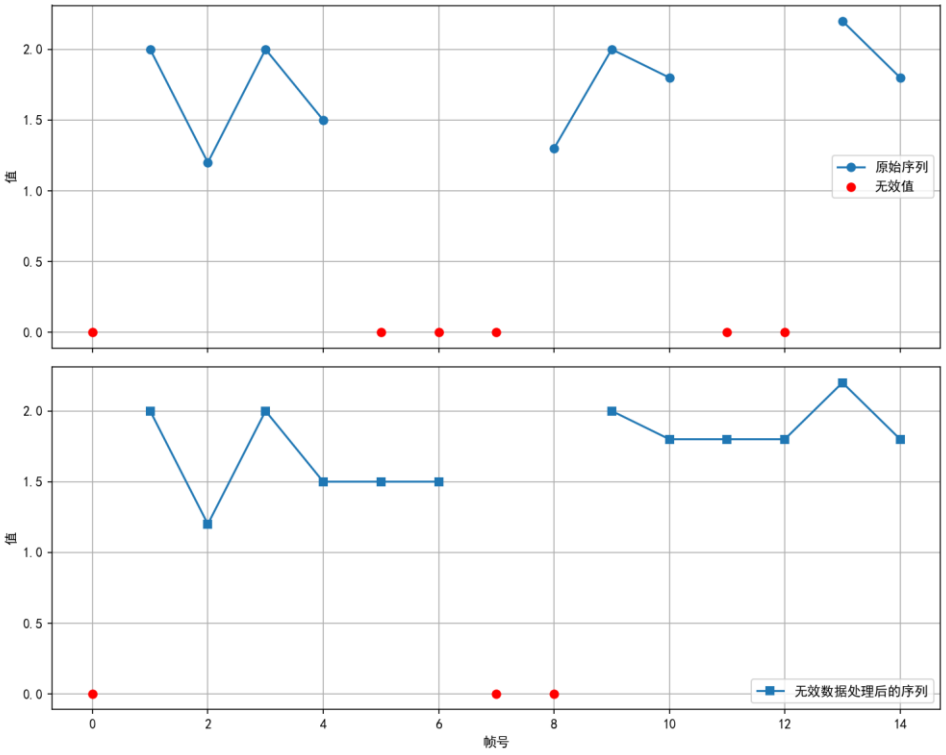
  - **平滑处理（X 轴）：**使用滑动窗口对单条轮廓线进行平滑处理，支持滑动平均和滑动中位数处理。其中滑动平均有助于减少噪点，且使得轮廓更加流畅，但可能损失边缘细节；中位数平滑有助于减少较为独立的离群噪点，并能较好地保持边缘信息。
  - **平滑处理（时间轴）：**在时间轴上进行轮廓平滑处理，同样支持滑动平均和滑动中位数处理。需要注意，使用该功能，会导致轮廓出现时间轴上的滞后。比如滑动平均窗口为w_1，中位数窗口为w_2，则会导致$\frac{w_1+w_2}{2}-1$条轮廓的滞后，我们会对前$\frac{w_1+w_2}{2}-1$条无效的轮廓使用-999 进行填充。
  - **无效数据处理：**用于减小时间轴上无效值跳变带来的影响。该功能有两个参数：

    1) 保持次数（有效点保持数）k_{valid}：假设 t 时刻的轮廓的第 i 个点是（完全）有效的，在第$t+1, t+2, \dots, t+k_{invalid}$时刻，如果轮廓的第 i 个点暂时变成无效，我们还是保持该点值为最近时刻的有效值。之后如果再出现无效点，则不再保持。
    2) 恢复次数（无效点保持数）$k_{invalid}$：和有效点保持数相反，如果轮廓突然变得有效，那么会暂时忽略，直到连续出现$k_{invalid}$后才会视为有效。

    状态转移示意图（假设有效点保持数为 2，无效点保持数为 1）：



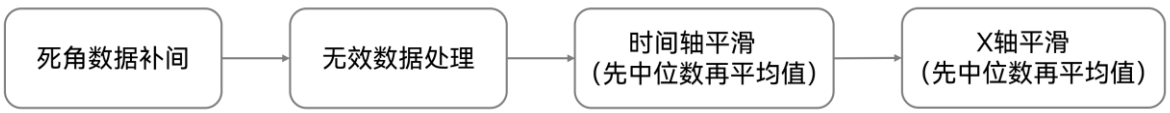
处理的时间序列示例如下：

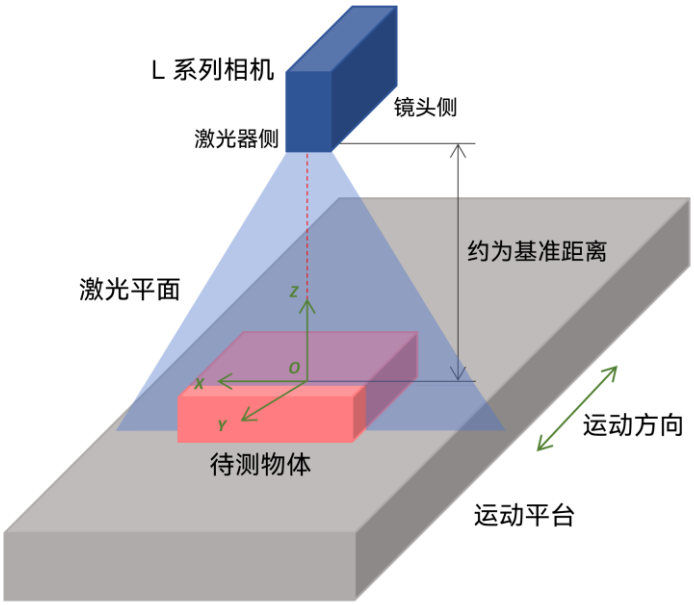


说明：状态初始化为 2；在 0 时刻，数据本身无效，此时只能输出无效值，状态为-1；在 1 时刻，原始数据恢复成有效，状态恢复为 2；在 5 时刻，原始数据暂时无效，保持输出 1.5，状态为-1；在 6 时刻，仍无效，保持输出 1.5，状态为-2，在 7 时刻，仍无效，状态变为-3（完全无效），不再保持；在 8 时刻，原始数据有效，但恢复次数为 1，导致此时仍输出无效值，状态为 1；在 9 时刻，原始数据有效，状态恢复为 2，输出原始数据.

■ **死角数据补间：**如果某个区间间隔点数不超过用户所设阈值，该功能可以将该区间内的点使用两侧的有效点进行线性插值。如果区间位于轮廓的左右边缘，将进行线性外推。

组合使用时，这几种处理方法的调用顺序如下：

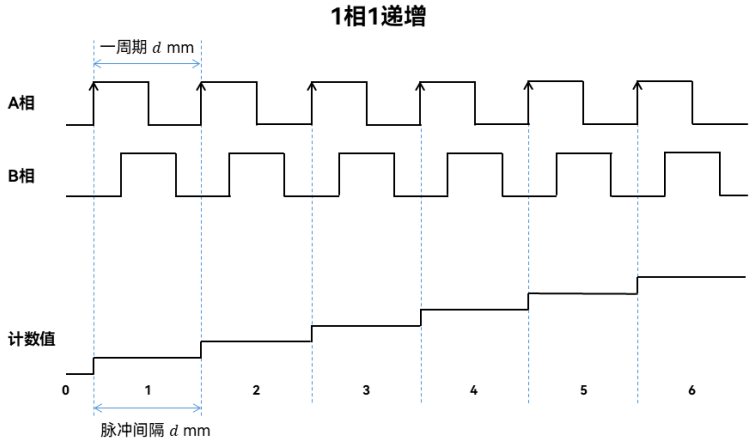




上图是相机和运动平台位置关系，平台的运动方向通常基本和相机坐标系的 Y 方向一致，但也可能存在一些偏差。用户可以设计算法对平台的运动方向进行标定。SDK 中提供了沿着某一固定方向拼接的函数 ProfileStitch，用户可以使用编码器值序列+编码器值脉冲间隔，或帧号序列+帧间间隔（假设平台匀速运动）进行拼接。

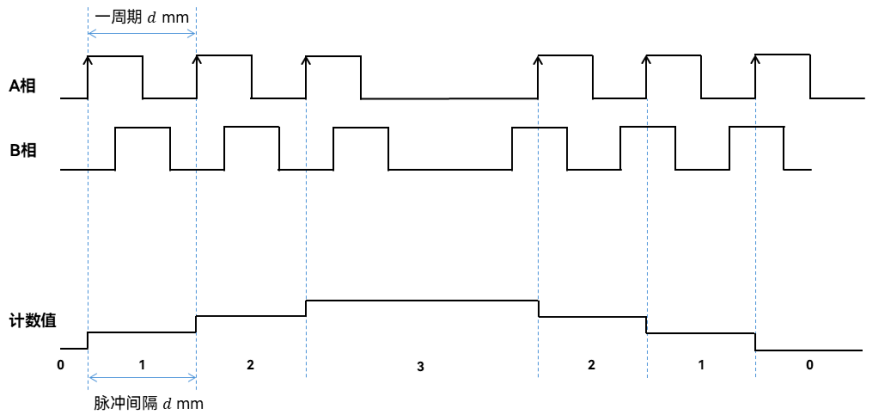
10.5.编码器触发原理

这涉及到 EncoderCountMode 的选择。



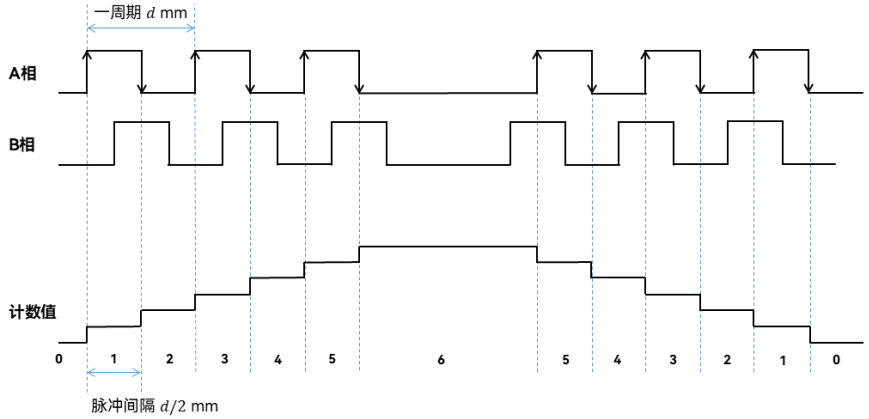
仅使用编码器A相上升沿触发，计数值递增，无法递减

2相1递增



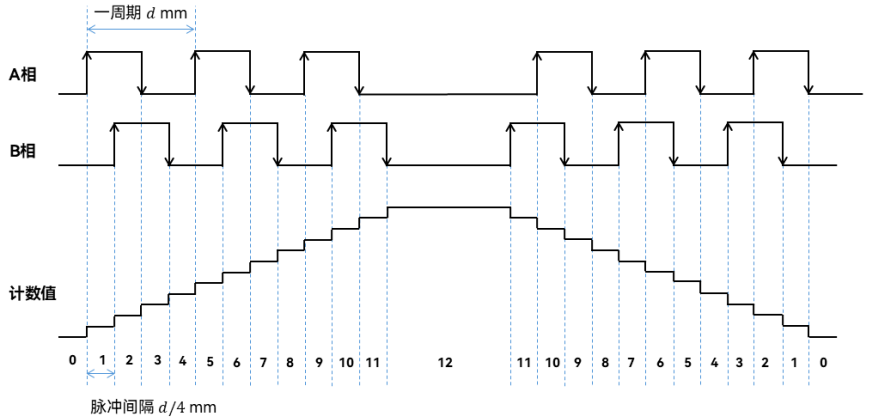
编码器A相比B相提前90度时，A相上升沿激活触发，计数值递增
编码器A相比B相延后90度时，A相上升沿激活触发，计数值递减

2相2递增



编码器A相比B相提前90度时，A相上升沿和下降沿都触发，计数值递增
编码器A相比B相延后90度时，A相上升沿和下降沿都触发，计数值递减

2相4递增



编码器A相比B相提前90度时，A相和B相上升沿和下降沿都触发，计数值递增
编码器A相比B相延后90度时，A相和B相上升沿和下降沿都触发，计数值递减