北京伟景智能科技有限公司

VzNLSDK（激光线检测）使用文档

文件名称：VzNLSDK 使用文档	**文件编号：VIZUM/SY-SW(20220808)-001**
生效日期：发布日期起	**文件页数：76 页**
发文类型：使用文档	**版本号/修订日期：2023 年 6 月 21 日**
发送部门：软件部	**发布日期：2022 年 08 月 08 日**

目录

VzNLSDK 使用文档

修改记录

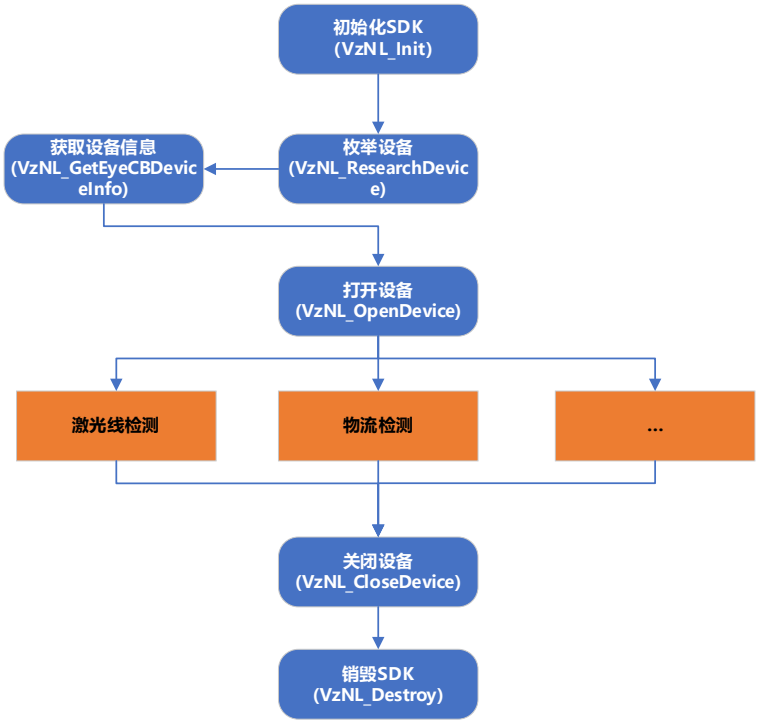
版本	时间	修改人	修改记录
V1.0.0	2019.02.13	Mjw YJ	库增加基础库，库名称修改 增加 VzKernel.dll 基础库 将 VzNL_Detect.dll 改为 VzNLDetect.dll 将 VzNL_Graphics.dll 改为 VzNLGraphics.dll 增加激光眼内容
V2.0.0	2019.07.31	Mjw	增加了 StartCapture 采图接口
V3.0.0	2019.10.17	Mjw	增加了激光线翻转的参数
V3.1.0	2019.11.16	Mjw	对应 SDK3.3.8 版本
V3.2.0	2019.12.28	Mjw	对应 3.4.0 以上版本，增加了智能眼接口
V3.2.1	2020.01.03	Mjw	增加了时间戳说明
V3.2.2	2020.02.17	Mjw	增加了 Trigger 模式 API，增加了 Idle 时可执行函数
V3.2.3	2020.02.25	Mjw	修改了 VzNL_Init 接口功能, VzNL_ResearchDevice 增加了控制搜索类型的 flag 修改了设置眼睛 IP 的接口，改为了使用 VZNLHANDLE 进行控制，以前是使用 index 号控制
V3.3.0	2020.04.20	Mjw	增加了摆动机构的控制接口支持 V3.5.0 以上
V3.3.1	2020.04.27	Mjw	支持 SDK 版本 V3.5.2 版本以上接口，修改了网络配置接口
V3.3.2	2021.04.19	Mjw	支持了 Xilinx 高帧率。 128 * 2048 = 2000FPS 256 * 2048 = 1000FPS ROI 检测最大支持 256 * 2048
V3.3.3	2021.08.13	Mjw	配合 VZNLSDK V3.6.14 更新 增加了旋转矩阵 增加了 RGBD 输出 修改了 RGB Sensor 的触发 增加了 RGB 自动曝光 增加了 RGB 自动白平衡
V3.3.4	2021.08.30	Mjw	新增扩展自动检测接口
V3.3.5	2021.09.16	Mjw	新增了 RGBD 动态相机同步
V3.3.6	2021.10.14	Mjw	增加了横向/竖向高斯的独立设置接口 增加了获取当前设备逆矩阵的接口 3D 转 2D 数据时，带入了逆矩阵的运算过程 增加了功耗配置接口
V3.3.7	2021.12.15	Mjw	增加了点填充模式
V3.3.8	2022.01.13	Mjw	增加了点云栅格化功能
V3.3.9	2022.04.26	Mjw	基于 SDK3.6.17.1 更新
V3.4.0	2022.08.08	Mjw	基于 SDK3.6.17.15 更新 增加了工作流程
V3.4.1	2023.06.02	Mjw	基于 SDK3.6.19.9 更新

			增加了控制 TriggerOut 的控制模式 增加了生成触发输出的信号 增加了启用输出 RGB 2D 点的功能
V3.4.2	2023.06.21	Mjw	基于 SDK3.6.19.13 版本更新 增加了获取扫描图像的 RGB 原图接口

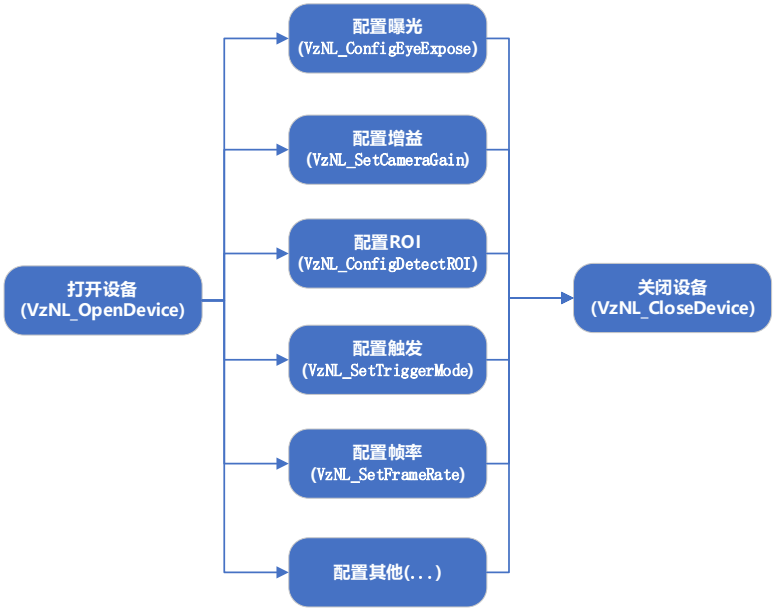
VzNLSDK 使用文档

1 相机工作流程

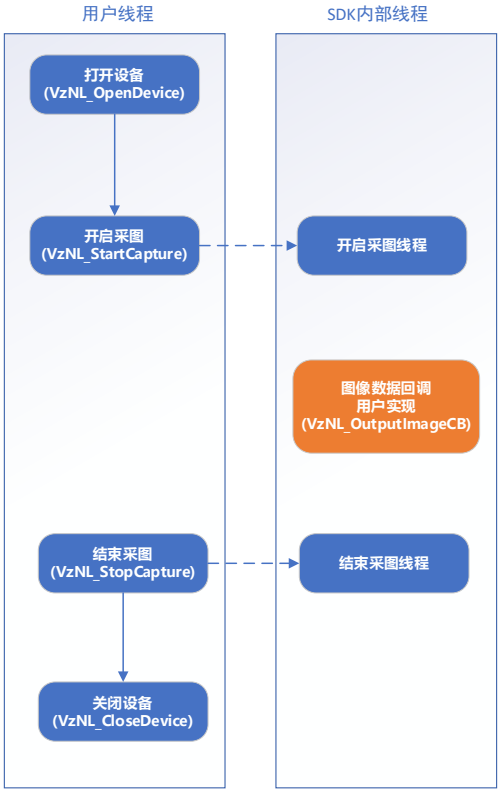
1.1 整体工作流程



1.2 参数配置流程



1.3 相机采图流程

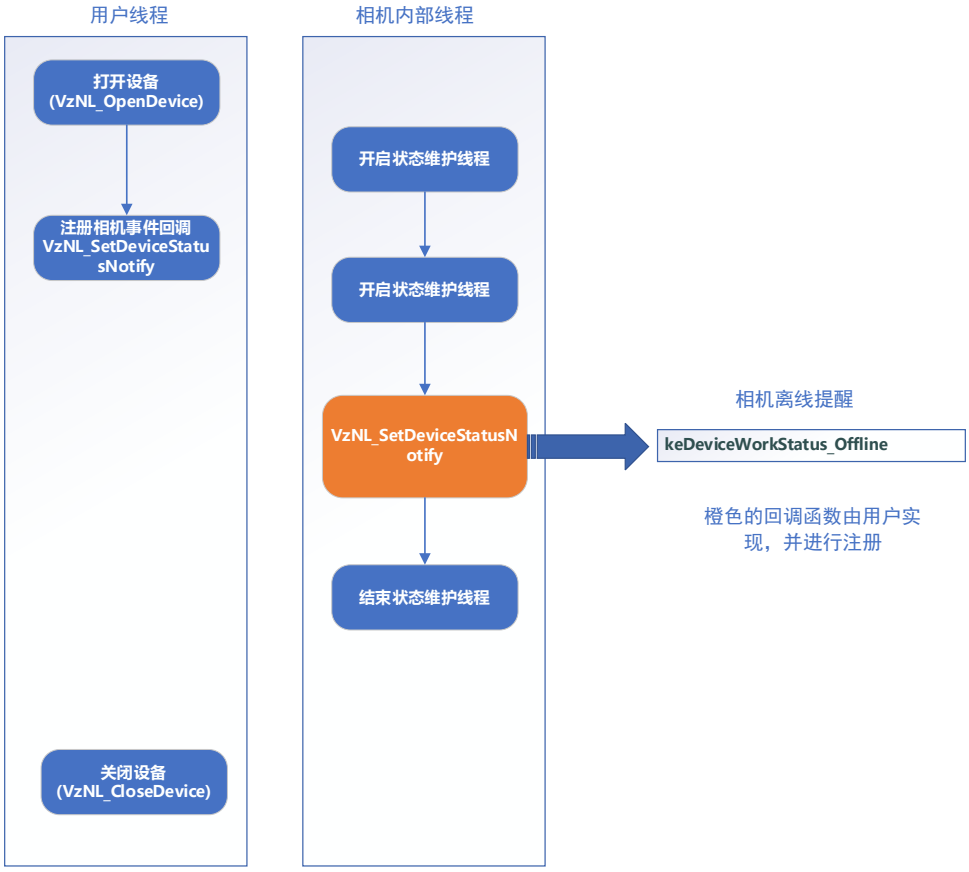


用户在实现回调函数时，请不要在回调函数中做太耗时的工作，否则可能会造成数据丢失

1.4 相机检测流程

用户在实现回调函数时，请不要在回调函数中做太耗时的工作，否则可能会造成数据丢失

1.5 相机事件注册流程



2 编程指导

2.1 伟景 SDK 库简介

伟景相机库使用 C/C++进行封装，在 Windows 上使用 Visual Studio 2015 进行编译，目前有 x86 和 x64 两个平台的版本，在 Linux 上使用 GCC 5.4 在 Ubuntu16.04 平台上进行编译，有 x86 和 x64 两个平台版本。

头文件在 SDK\VzNLSDK\Inc 目录下;

Windows 库文件在 SDK\VzNLSDK\Windows 目录下;

Linux 库文件在 SDK\VzNLSDK\Linux 目录下

头文件

头文件	头文件描述
VZNL_Export.h	SDK 输入输出宏定义
VZNL_Types.h	SDK 数据定义
VZNL_ErrorCode.h	SDK 错误码参考文件
VZNL_Common.h	SDK 公用文件
VZNL_DetectConfig.h	相机通用配置接口文件
VZNL_EyeConfig.h	相机网络配置接口文件
VZNL_FileUtils.h	SDK 文件处理接口

VZNL_Utils.h	SDK 后处理辅助接口
VZNL_RGBConfig.h	RGB 相机配置接口
VZNL_SwingMotor.h	摆动模块接口文件
VZNL_Graphics.h	绘图支持文件
VZNL_DetectLaser.h	激光检测接口文件

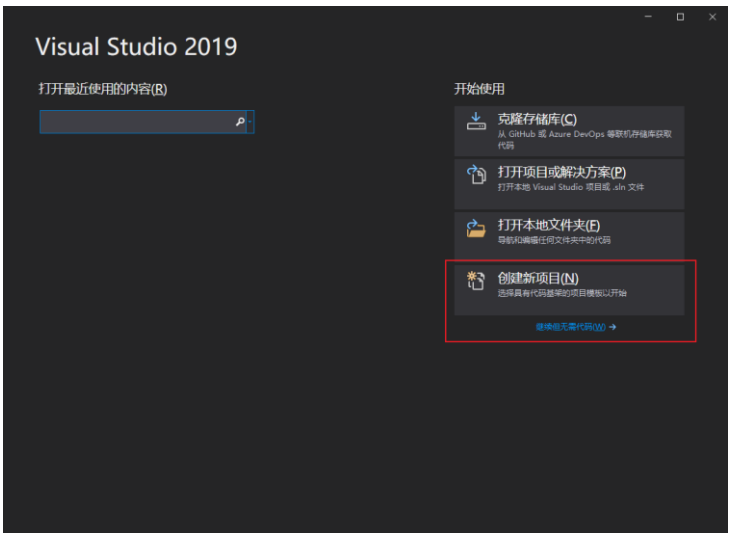
库文件

库名	库文件描述
libViEyeApi.dll	星光眼 支持库
VzKernel.dll	SDK 基础库
VzLog.dll	SDK Log 库
VzNLDetect.dll	SDK 检测库
VzNLGraphics.dll	SDK 绘图库

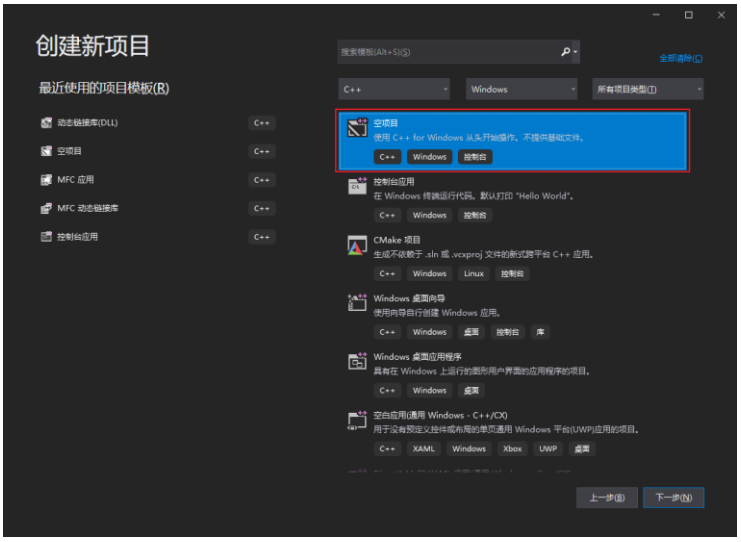
2.2 Windows 编译平台搭建

本文中以 Visual Studio 2019 为例进行描述

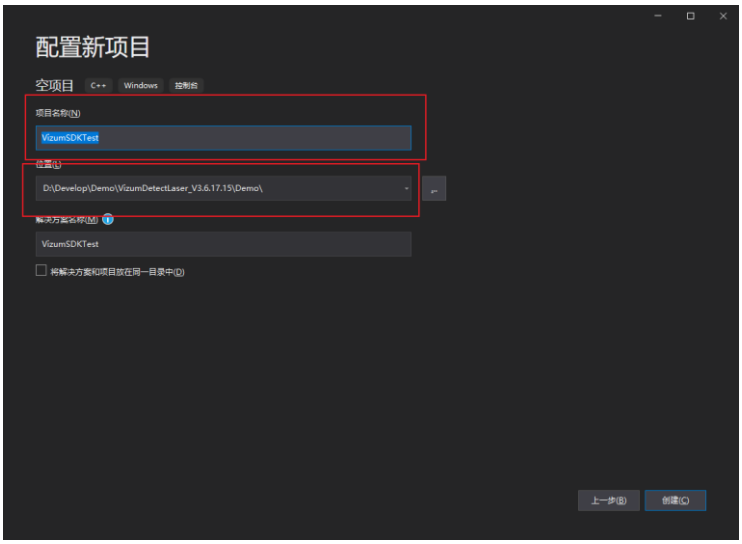
1）创建新项目



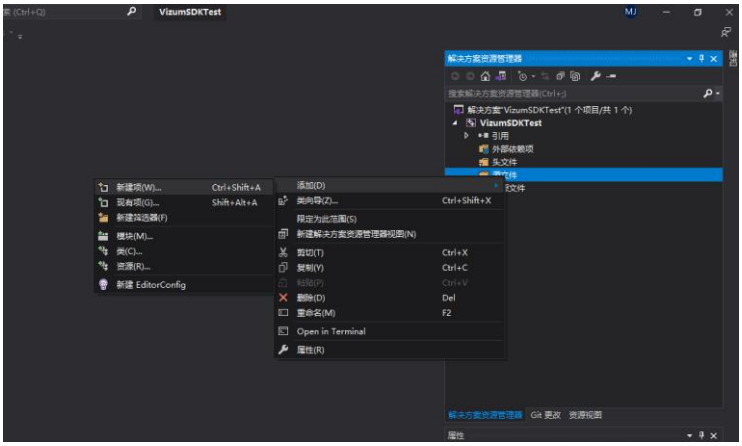
2）选择创建的项目

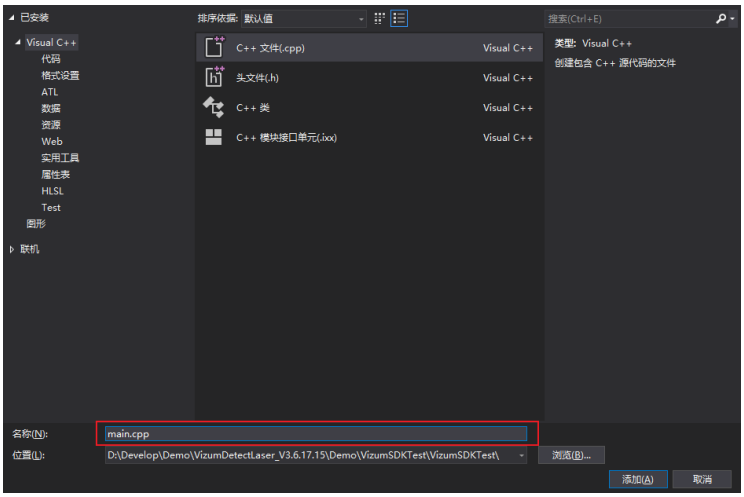


3）给要创建的项目起一个适合的名称

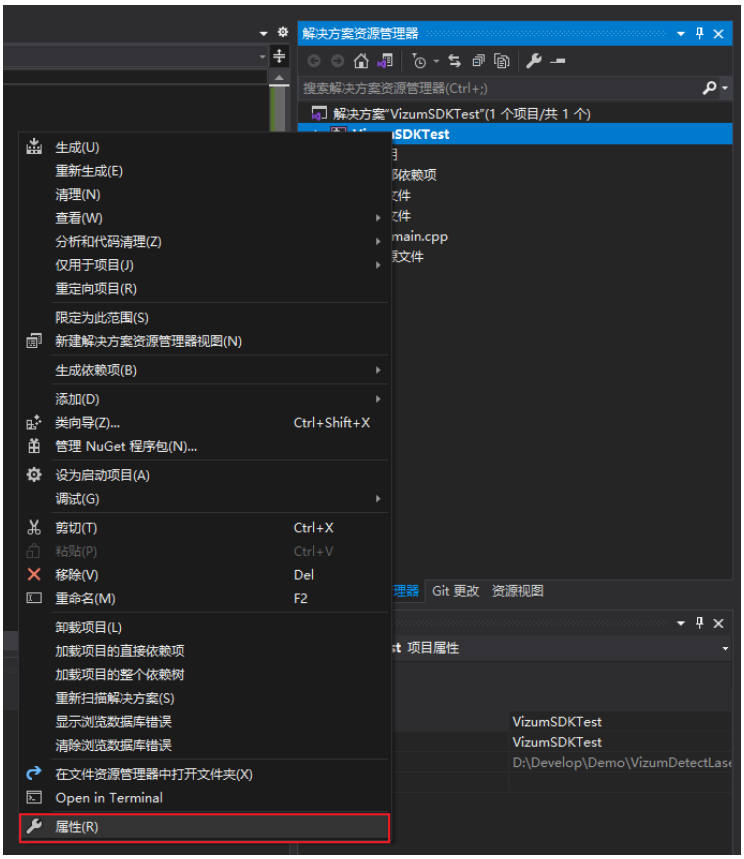


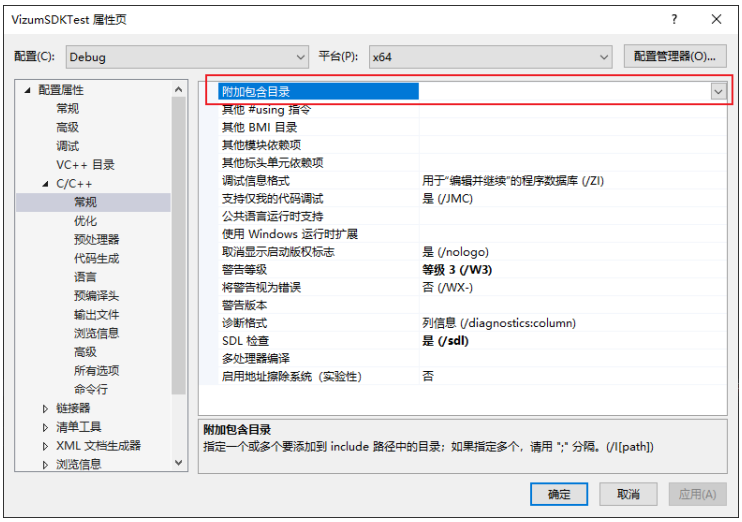
4）创建一个文件程序主文件

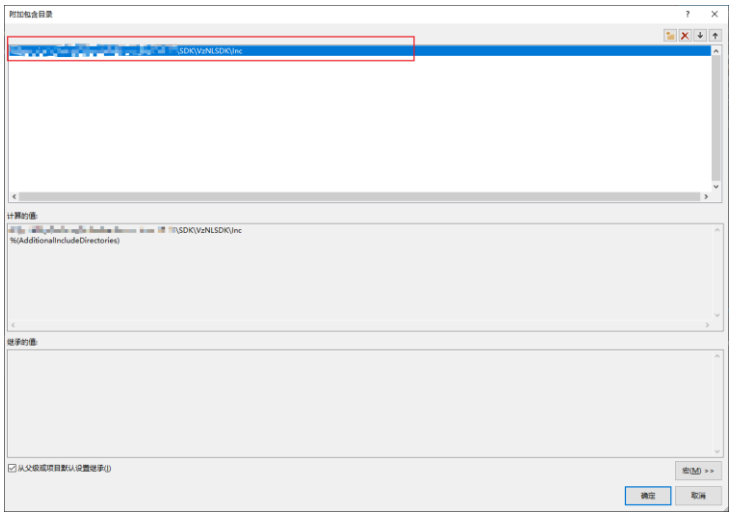


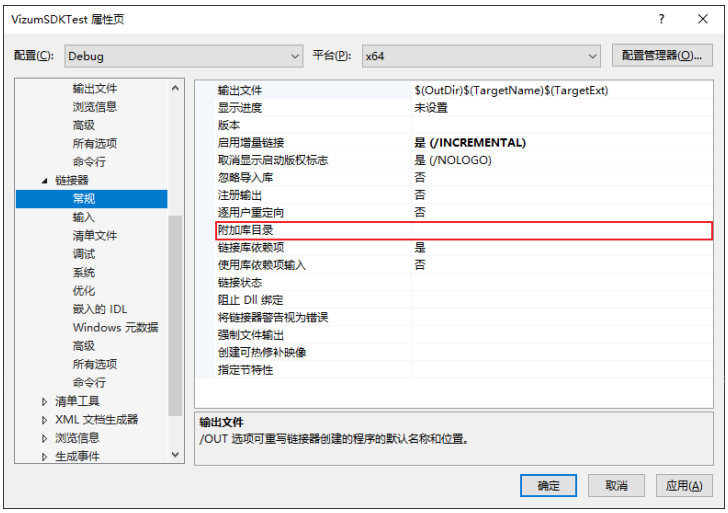


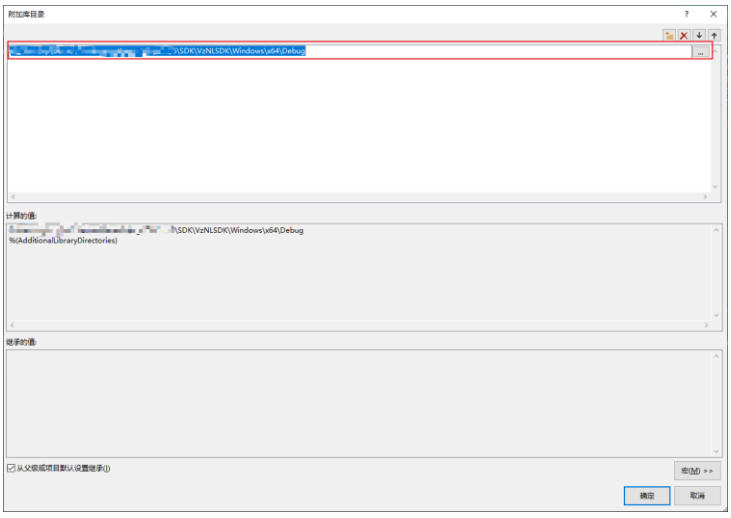
5）修改库文件包含目录


解决方案资源管理器
搜索解决方案资源管理器(Ctrl+;)
解决方案"VizumSDKTest"(1 个项目/共 1 个)
生成(U)
重新生成(E)
清理(N)
查看(W)
分析和代码清理(Z)
仅用于项目(J)
重定向项目(R)
限定为此范围(S)
新建解决方案资源管理器视图(N)
生成依赖项(B)
添加(D)
类向导(Z)... Ctrl+Shift+X
管理 NuGet 程序包(N)...
设为启动项目(A)
调试(G)
剪切(T) Ctrl+X
粘贴(P) Ctrl+V
移除(V) Del
重命名(M) F2
卸载项目(L)
加载项目的直接依赖项
加载项目的整个依赖树
重新扫描解决方案(S)
显示浏览数据库错误
清除浏览数据库错误
在文件资源管理器中打开文件夹(X)
Open in Terminal
属性(R)
Git 更改
资源视图
项目属性
VizumSDKTest
VizumSDKTest


VizumSDKTest 属性页
配置(C): Debug
平台(P): x64
配置管理器(O)...
配置属性
常规
高级
调试
VC++ 目录
C/C++
常规
优化
预处理器
代码生成
语言
预编译头
输出文件
浏览信息
高级
所有选项
命令行
链接器
清单工具
XML 文档生成器
浏览信息
附加包含目录
其他 #using 指令
其他 BMI 目录
其他模块依赖项
其他标头单元依赖项
调试信息格式 用于"编辑并继续"的程序数据库 (/ZI)
支持仅我的代码调试 是 (/JMC)
公共语言运行时支持
使用 Windows 运行时扩展
取消显示启动版权标志 是 (/nologo)
警告等级 等级 3 (/W3)
将警告视为错误 否 (/WX-)
警告版本
诊断格式 列信息 (/diagnostics:column)
SDL 检查 是 (/sdl)
多处理器编译
启用地址擦除系统（实验性） 否
附加包含目录
指定一个或多个要添加到 include 路径中的目录；如果指定多个，请用 ";" 分隔。(/I[path])
确定
取消
应用(A)







6）编写测试程序

在预先创建的文件 main.cpp 中编写代码

```
#include "stdio.h"
#include <vector>
#include "VzNL_Common.h"

int main(int argc, char* argv[])
{
    SVzNLConfigParam sConfigParam;
    sConfigParam.nDeviceTimeOut = -1;
    int nRet = VzNL_Init(&sConfigParam);
    if (0 != nRet)
    {
        printf("Init SDK Failed!");
        return 0;
    }

    // 搜索智光眼
    VzNL_ResearchDevice(keSearchDeviceFlag_EthLaserRobotEye);

    // 获取设备信息
    int nDevCnt = 0; std::vector<SVzNLEyeCBInfo> vetDevInfo;
    VzNL_GetEyeCBDeviceInfo(nullptr, &nDevCnt);
    vetDevInfo.resize(nDevCnt);
    if (nDevCnt > 0)
    {
        VzNL_GetEyeCBDeviceInfo(vetDevInfo.data(), &nDevCnt);
    }

    // 显示设备信息
    for (int nDevIdx = 0; nDevIdx < nDevCnt; nDevIdx++)
    {
        printf("DevIP: %s\n", vetDevInfo[nDevIdx].byServerIP);
    }

    // 销毁SDK
    VzNL_Destroy();
    return 0;
}
```

7) 编译程序

8) 拷贝 SDK 库(.dll)到编译生成文件夹中

9) 运行

检测库接口入参、出参说明

3 VzNLSDK 详细接口

3.1 VzNLSDK 基础接口

头文件：#include "VZNL_Common.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.1.1 初始化接口（3.4.8 有修改）

int VzNL_Init(const SVzNLConfigParam* pConfigParam);

入参：　pConfigParam　　初始化检测参数，包含眼睛分辨率，超时时间

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

注意：　3.4.8 以前此函数会自带一次搜索

3.4.8 开始 Init 不带搜索，必须调用 VzNL_ResearchDevice 才可以枚举到设备

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

// 这里编写SDK相关代码

// 销毁SDK
VzNL_Destroy();
```

3.1.2 重新搜索设备（3.4.8 有修改）

int VzNL_ResearchDevice(EVzSearchDeviceFlag nFindDevFlag);

入参：　nFindDevFlag

* keSearchDeviceFlag_EyeCB　　表示搜索 EyeCB 设备
* keSearchDeviceFlag_USBLargeEye　表示搜索 USB 眼睛设备
* keSearchDeviceFlag_EthLargeEye　表示搜索星光眼
* keSearchDeviceFlag_EthLaserRobotEye　表示搜索以太网智光眼
* keSearchDeviceFlagAll　　表示搜索所有支持的设备

注意：此接口的参数也可以拼合使用例如要同时搜索 USB 和网络极光眼的相机的时候可以使用这种写法 keSearchDeviceFlag_USBLargeEye | keSearchDeviceFlag_EthLargeEye

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

注意：　3.4.8 版本以前是无需填写参数的，直接搜索所有的设备，到 3.4.8 版本可以让用户灵活的搜索所有设备

```
// 搜索智光眼相机
```

```
VzNL_ResearchDevice(keSearchDeviceFlag_EthLaserRobotEye);
```

3.1.3 获取所有搜索的设备

int VzNL_GetEyeCBDeviceInfo(SVzNLEyeCBInfo* pInfo, int* pnCount);

入参/出参： pInfo 搜索到的所有设备存储数组的首地址 pnCount 为搜索的设备个数。

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

for (nDevIdx = 0; nDevIdx < nDevCount; nDevIdx++)
{
    printf("Device %d: %s %s\r\n", nDevIdx, pCBInfo[nDevIdx].szDeviceName,
pCBInfo[nDevIdx].byServerIP);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}
```

3.1.4 获取所有打开的设备

int VzNL_GetAllOpenDevice(SVzNLEyeCBInfo* pInfo, int* pnCount);

入参/出参： pInfo 打开设备存储数组的首地址 pnCount 为打开的设备个数。

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```

```

3.1.5 打开设备

VZNLHANDLE VzNL_OpenDevice(const SVzNLEyeCBInfo* pInfo, const SVzNLOpenDeviceParam* pOpenDeviceParam, int* pErrorCode);

入参： pInfo 所要打开设备结构体指针

pOpenDeviceParam 打开设备入参，设备检测类型

pErrorCode 错误码

返回： 设备句柄。NULL 为打开失败

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
```

```
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
    If ((hDevice != NULL) && (nErrorCode == 0))
    {
        // 对设备进行操作
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.1.6 关闭设备

int VzNL_CloseDevice(VZNLHANDLE hDevice);

入参：　hDevice　　设备句柄

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息

```
#include <stdio.h>
#include "VZNL_Common.h"
```

```
// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
    If ((hDevice != NULL) && (nErrorCode == 0))
    {
        // 对设备进行操作
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.1.7 版本信息

int VzNL_GetVersion(VZNLHANDLE hDevice, SVzNLVersionInfo* pVersionInfo);

入参： hDevice 设备句柄

pVersionInfo 版本信息结构体的地址

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
SVzNLVersionInfo sVersion = { 0 };
```

```
int nErrorCode = VzNL_GetVersion(hDevice, &sVersion);
if (0 != nErrorCode)
{
  // 打印硬件版本号
  if (sVersion.sCap.sVersionCap.bValidVizumEyeVersion)
  {
    printf("VizumEye Version:%s\r\n", sVersion.szVizumEyeVersion);
  }
}
```

3.1.8 获取设备信息

int VzNL_GetDeviceInfo(VZNLHANDLE hDevice, SVzNLEyeCBInfo* pInfo);

入参： hDevice 设备句柄

pInfo 设备信息结构体的首地址

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
SVzNLEyeCBInfo sEyeCBInfo;
int nRet = VzNL_GetDeviceInfo(hDevice, (SVzNLEyeCBInfo*)&sEyeCBInfo);
if (sEyeCBInfo.eDeviceType == keDeviceType_EyeCB)
{
    SVzNLEyeCBDeviceInfoEx sEyeDevInfo;
    sEyeDevInfo.sEyeCBInfo.nSize = sizeof(SVzNLEyeCBDeviceInfoEx);
    int nRet = VzNL_GetDeviceInfo(hDevice, (SVzNLEyeCBInfo*)&sEyeDevInfo);
    if (0 == nRet)
    {
        printf("Resolution %d %d\r\n", sEyeDevInfo.sVideoRes.nFrameWidth,
sEyeDevInfo.sVideoRes.nFrameHeight);
    }
}
else if (sEyeCBInfo.eDeviceType == keDeviceType_LaserEye ||
        sEyeCBInfo.eDeviceType == keDeviceType_LaserRobotEye)
{
    SVzNLEyeDeviceInfoEx sEyeDevInfo;
sEyeDevInfo.sEyeCBInfo.nSize = sizeof(SVzNLEyeDeviceInfoEx);
    int nRet = VzNL_GetDeviceInfo(hDevice, (SVzNLEyeCBInfo*)&sEyeDevInfo);
    if (0 == nRet)
    {
        printf("Disparity %d\r\n", sEyeDevInfo.nFixDisparity);
        printf("Version %s\r\n", sEyeDevInfo.szVersion);
        printf("Resolution %d %d\r\n", sEyeDevInfo.sVideoRes.nFrameWidth,
sEyeDevInfo.sVideoRes.nFrameHeight);
    }
}
```

3.1.9 获取图像

(SDK1.4.0 中修改，将压缩格式参数去掉了，获取出来的 Buffer 为无压缩图像)

int VzNL_GetEyeImage(VZNLHANDLE hDevice, SVzNLImageData** ppLeftEye, SVzNLImageData** ppRightEye, unsigned int nTimeOut);

入参： hDevice 设备句柄

nTimeOut 超时时间

出参： ppLeftEye 左眼睛图像的结构地址

ppRightEye 右眼睛图像的结构地址

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
SVzNLImageData* pLeftEye = NULL;
SVzNLImageData* pRightEye = NULL;

int nErrorCode = VzNL_GetEyeImage(hDevice, nullptr, &pRightEye, 5000);
if (0 == nErrorCode)
{
    VzNL_SaveImage("D:\\l.png", pLeftEye);
    VzNL_SaveImage("D:\\r.png", pRightEye);
}

VzNL_ReleaseImage(&pLeftEye);
VzNL_ReleaseImage(&pRightEye);
```

释放图像

int VzNL_ReleaseImage(SVzNLImageData** ppImageData);

入参： ppImageData 图像数据

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
SVzNLImageData* pLeftEye = NULL;
SVzNLImageData* pRightEye = NULL;

int nErrorCode = VzNL_GetEyeImage(hDevice, nullptr, &pRightEye, 5000);
if (0 == nErrorCode)
{
    VzNL_SaveImage("D:\\l.png", pLeftEye);
    VzNL_SaveImage("D:\\r.png", pRightEye);
}

VzNL_ReleaseImage(&pLeftEye);
VzNL_ReleaseImage(&pRightEye);
```

3.1.10 获取错误信息

int VzNL_GetErrorInfo(int nErrorCode, char szError[256]);

入参： nErrorCode 错误码

出参： szError 错误信息

返回： 0 为正确。

```
char szErrorInfo[256] = { 0 };
if (0 == nErrorCode)
```

```
{
    printf("Execute Success\r\n");
    return;
}
VzNL_GetErrorInfo(nErrorCode, szErrorInfo);
printf("Failed: %s(%d)\r\n", szErrorInfo, nErrorCode);
```

3.1.11 设置图像输出格式

void VzNL_SetOutputImageFormat(EVzNLImageType eOutputImageType);

入参： eOutputImageType 当值为 keVzNLImageType_None 时，表示输出 原图格式，为其他时则返回其对应的格式。

```
// 设置图像输出格式为RGB
VzNL_SetOutputImageFormat(keVzNLImageType_BGR888);
```

3.1.12 设置出图方式

void VzNL_SetOutputROIImage(VzBool bOutputROI);

入参： bOutputROI 是否输出 ROI 图像(默认为 VzFalse)

```
// 设置为ROI出图模式
VzNL_SetOutputROIImage(VzTrue);
```

3.1.13 连续取图

int VzNL_StartCapture(VZNLHANDLE hDevice, VzNL_OutputImageCB pImageCB, void* pCBParam);

入参： hDevice 设备 Handle

入参： pImageCB 图像回调接口

入参： pCBParam 回调接口参数

回调函数：

typedef void(*VzNL_OutputImageCB)(SVzNLImageData* pLeftImage, SVzNLImageData* pRightImage, SVzNLImageData* pCenterImage, const SVzOutputFrameProps* pFrameProps, void* pParam);

出参： pLeftImage 左图

出参： pRightImage 右图

出参： pFrameProps 输出帧的参数

出参： pParam 回调参数

返回 如果接口调用成功成功返回 0

停止取图

int VzNL_StopCapture(VZNLHANDLE hDevice);

入参：hDevice 设备 Handle

返回：调用接口成功返回 0

是否处于取图中

入参： hDevice 设备 Handle

入参： nErrorCode 当 pnErrorCode 不为 nullptr 时,输出错误码

返回： VzTrue,表示正在取图,返回 VzFalse 表示不在取图

VzBool VzNL_IsCapturing(VZNLHANDLE hDevice, int* pnErrorCode);

```
void _OnOutputImageCB(SVzNLImageData* pLeftImage, SVzNLImageData* pRightImage,
SVzNLImageData* pColorImage, const SVzOutputFrameProps* sProps, void* pParam)
{
    printf("Left Image %p Right Image %p CenterImage %p \n", pLeftImage, pRightImage,
pColorImage);
}

// 开始采集示例
{
    int nErrorCode = VzNL_StartCapture(hDevice, _OnOutputImageCB, NULL);
    if (0 != nErrorCode)
    {
        return;
    }

    VzBool bIsCapture = VzNL_IsCapturing(hDevice, &nErrorCode);
    VzBool bIsAutoDetect = VzNL_IsAutoDetecting(hDevice, &nErrorCode);

    if (bIsCapture || bIsAutoDetect)
    {
        printf("当前状态 %s %s\n", bIsCapture ? "正在采集图像" : "", bIsAutoDetect ? "
正在检测激光线" : "");
    }
    else
    {
      printf("当前状态 空闲\n");
    }

    nErrorCode = VzNL_StopCapture(hDevice);
    if (0 != nErrorCode)
    {
      return;
    }
}
```

3.1.14 设置 Log 等级

void VzNL_SetLogLevel(EVzNLLogLevel eLevel, EVzNLLogType eLogType);

入参：eLevel Log 等级

入参：eLogType Log 类型

```
// 输出所有的Log到文件中VizumLog\xxx.txt
VzNL_SetLogLevel(keNLLogLevel_Information, keNLLogType_File);
```

3.1.15 设置设备状态提醒

int VzNL_SetDeviceStatusNotify(VZNLHANDLE hDevice, VzNL_OnNotifyStatusCB pNotifyCB);

入参： hDevice 设备 Handle

入参： pNotifyCB 返回函数

返回： 成功返回 0，否则为错误码

回调函数：

void(*VzNL_OnNotifyStatusCB)(EVzDeviceWorkStatus eStatus, void* pInfoParam);

设备状态改变 CallBack 函数

eStatus 状态

pInfoParam 状态参数

```
void _OnOutputImageCB(SVzNLImageData* pLeftImage, SVzNLImageData* pRightImage,
SVzNLImageData* pColorImage, const SVzOutputFrameProps* sProps, void* pParam)
{
    printf("Left Image %p Right Image %p CenterImage %p \n", pLeftImage, pRightImage,
pColorImage);
}

int nErrorCode = VzNL_StartCapture(hDevice, _OnOutputImageCB, NULL);
if (0 != nErrorCode)
{
    return;
}

nErrorCode = VzNL_StopCapture(hDevice);
if (0 != nErrorCode)
{
    return;
}
```

3.1.16 重连设备

int VzNL_ReConnectDevice(VZNLHANDLE hDevice);

入参： hDevice 设备 Handle

返回： 成功返回 0，否则为错误码

```
// 当设备离线后，可调用VzNL_ReConnectDevice对当前设备主动重连
VzNL_ReConnectDevice(hDevice)
```

3.1.17 启用所见即所得 ROI 图像（图像矫正）

入参: hDevice 设备 Handle

入参: bEnable VzTrue 启用/VzFalse 禁用

返回 关闭成功返回 0，否则为错误码

int VzNL_EnableCalibROI(VZNLHANDLE hDevice, VzBool bEnable);

是否启用了所见即所得功能

入参： hDevice 设备 Handle

出参： pnErrorCode 错误码

返回： VzTrue 为启用了所见即所得 VzFalse 为关闭了所见即所得

VzBool VzNL_IsEnableCalibROI(VZNLHANDLE hDevice, unsigned int* pnErrorCode);

```
EnableCalibROI(hDevice, VzTrue);

// 获取所见即所得后的图像
// 采图流程

EnableCalibROI(hDevice, VzFalse);
```

3.1.18 重启相机

入参： hDevice 设备 Handle

int VzNL_RebootDevice(VZNLHANDLE hDevice);

```
// 重启设备
VzNL_RebootDevice(hDevice);
```

3.1.19 关闭 SDK

void VzNL_Destroy();

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

// 这里编写SDK相关代码

// 销毁SDK
VzNL_Destroy();
```

3.2 VzNLSDK 基础配置接口

头文件：#include "VZNL_DetectConfig.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.2.1 检测区域设置

①配置检测区域

int VzNL_ConfigDetectROI(VZNLHANDLE hDevice, const SVzNLROIRect* pLeftROIArea, const SVzNLROIRect* pRightROI);

入参： hDevice 设备句柄
pLeftROIArea 左眼图像检测区域
pRightROI 右眼图像检测区域

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

②获取检测区域

int VzNL_GetConfigDetectROI(VZNLHANDLE hDevice, SVzNLROIRect* pLeftROI, SVzNLROIRect* pRightROI);

入参： hDevice 设备句柄
pLeftROI 左图像 ROI
pRightROI 右图像 ROI

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

③格式化 ROI

int VzNL_FormatDetectROI(VZNLHANDLE hDevice, SVzNLROIRect* pLeftROI, SVzNLROIRect* pRightROI, VzBool bKeepTop);

入参： hDevice 设备句柄

入参： pLeftROI 左侧 ROI

入参： pRightROI 右侧 ROI

入参： bKeepTop 保持高度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
// 注意智光眼ROI 最小为64 * 256，在未开启所见即所得的时候，需要宽度32对齐，高度128对齐
// 开启所见即所得后，没有限制
SVzNLROIRect sLeftROI = {0, 64, 0, 1024};
SVzNLROIRect sRightROI = { 0, 64, 0, 1024 };

// 获取格式化后的ROI
Int nErrorCode = VzNL_FormatDetectROI(hDevice,  &sLeftROI, &sRightROI, VzTrue);
If (0 == nErrorCode)
{
    // 配置 ROI
    VzNL_ConfigDetectROI(hDevice, &sLeftROI, &sRightROI);
}

// 获取 ROI
SVzNLROIRect sRetLeftROI = {0, 64, 0, 1024};
SVzNLROIRect sRetRightROI = { 0, 64, 0, 1024 };
VzNL_ GetConfigDetectROI (hDevice, &sRetLeftROI, &sRetRightROI);
```

3.2.2 设置/获取眼睛曝光值

int VzNL_ConfigEyeExpose(VZNLHANDLE hDevice, EVzNLExposeMode eExposeMode, unsigned int nExposeTime);

入参: hDevice 设备句柄

入参：eExposeMode [keVzNLExposeMode_Fix=定值曝光 / keVzNLExposeMode_Auto=自动曝光]

入参：nExposeTime 曝光时间[20~100000]

返回：返回错误码

注意: 目前智光眼/星光眼仅支持定值曝光

int VzNL_GetConfigEyeExpose(VZNLHANDLE hDevice, EVzNLExposeMode* peExposeMode, unsigned int* pnExposeTime);

```
// 配置相机曝光为 100
VzNL_ConfigEyeExpose(hDevice, keVzNLExposeMode_Fix， 100);
```

3.2.3 设置/获取相机增益

入参: hDevice 设备句柄

入参: eSensorType [keEyeSensorType_Left=左眼 / keEyeSensorType_Right=右眼]

入参: nCameraGain 曝光时间[0~255]

返回: 返回错误码

注意: 目前智光眼支持

int VzNL_SetCameraGain(VZNLHANDLE hDevice, EVzEyeSensorType eSensorType, unsigned short nCameraGain);

int VzNL_GetCameraGain(VZNLHANDLE hDevice, EVzEyeSensorType eSensorType, unsigned short* pnCameraGain);

```
// 设置左右目的增益为 1
VzNL_SetCameraGain(hDevice, keEyeSensorType_Left,1);
VzNL_SetCameraGain(hDevice, keEyeSensorType_Right,1);
```

3.2.4 设置 Trigger 模式

int VzNL_SetTriggerMode(VZNLHANDLE hDevice, EVzEyeTriggerMode eTriggerMode);

入参: hDevice 设备句柄

入参: eTriggerMode Trigger 模式[被动/主动/上升沿/下降沿/低电平使能]，默认使用 Master 主动触发模式

返回: 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.2.5 获取当前 Trigger 模式

EVzEyeTriggerMode VzNL_GetTriggerMode(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

入参： pnErrorCode 设置为 null 时不对其进行赋值，否则则返回错误码，0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

返回： Trigger 模式[被动/主动/上升沿/下降沿/低电平使能]

```
// 设置为主模式，主模式下由系统按帧率进行自动检测
VzNL_SetTriggerMode(hDevice, keEyeTriggerMode_Master);
```

3.2.6 设置触发极性模式

int VzNL_SetTriggerPolar(VZNLHANDLE hDevice, VzBool bNormal);

VzBool VzNL_GetTriggerPolar(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

入参： bNormal VzTrue 为默认极性/VzFalse 反向

返回： 返回错误码

```
// 触发极性，若修改则会有高电平触发变为低电平触发
SetTriggerPolar(hDevice, VzTrue);
```

3.2.7 设置触发分频

int VzNL_SetTriggerDivCoe(VZNLHANDLE hDevice, unsigned int nDivCoe);

unsigned int VzNL_GetTriggerDivCoe(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

入参： nDivCoe 分频数

返回： 返回错误码

```
// 默认为 1，表示每 1 个触发信号可以对相机进行触发一次
VzNL_SetTriggerDivCoe(hDevice, 1);
```

3.2.8 是否忽略外部使能

int VzNL_IgnoreTriggerExtSignal(VZNLHANDLE hDevice, VzBool bIgnore);

VzBool VzNL_IsIgnoreTriggerExtSignal(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

入参： bIgnore VzTrue 忽略（默认） VzFalse (关联外部使能信号)

返回： 返回错误码

```
// 配置忽略外部使能
VzNL_IgnoreTriggerExtSignal(hDevice, VzTrue);
```

3.2.9 启用外部信号使能

int VzNL_EnableTriggerHwExtEn(VZNLHANDLE hDevice, VzBool bHwExtEnable);

VzBool VzNL_IsEnableTriggerHwExtEn(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

入参： bHwExtEnable VzTrue 为使用硬信号 VzFalse 为使用软信号

返回： 返回错误码

```
// 需要配置忽略使能信号为 VzFalse 时才会起作用

// 配置为硬使能，接受硬件使能信号
VzNL_EnableTriggerHwExtEn(hDevice, VzTrue);

// 配置为软使能，接受软件生成的使能信号
VzNL_EnableTriggerHwExtEn(hDevice, VzTrue);
```

3.2.10 生成软外部使能信号

int VzNL_GenerateTriggerSoftExtEn(VZNLHANDLE hDevice, VzBool bHighLevel);

入参： hDevice 设备句柄

入参： bHighLevel 高电平信号

返回： 返回错误码

```
// 当使能信号为软使能时，配置 VzTrue 或者 VzFalse 可以模拟其高低电平信号
VzNL_GenerateTriggerSoftExtEn(hDevice, VzTrue);
```

3.2.11 获取参数设置范围

入参：hDevice 设备句柄

入参：eType 参数类型

入参：pnMinVal 最小值

入参：pnMaxVal 最大值

返回：返回错误码

int VzNL_QueryParamRange(VZNLHANDLE hDevice, EVzDeviceParamType eType, unsigned int* pnMinVal, unsigned int* pnMaxVal);

```
// 获取当前ROI和曝光下的帧率范围
unsigned int nMinFrameRate, nMaxFrameRate;
VzNL_QueryParamRange(hDevice,keDeviceParamType_FrameRate, &nMinFrameRate,
&nMaxFrameRate);
```

3.2.12 是否启用同步 RGB 设置

入参：hDevice 设备 handle

入参：bEnable true 表示同步设置 RGB，false 表示不进行同步

返回：返回错误码

int VzNL_EnableSyncRGBD(VZNLHANDLE hDevice, VzBool bEnable);

```
// 启用 RGB 同步,当 RGB 动态相机检测时，请启用此选项
VzNL_EnableSyncRGBD(hDevice, VzTrue);
```

3.2.13 是否启用低功耗模式

入参：hDevice 设备 handle

入参：bEnable true 表示开启低功耗,false 表示关闭低功耗

返回：返回错误码

int VzNL_EnableLowPowerMode(VZNLHANDLE hDevice, VzBool bEnable);

```
// 默认静态相机是启用低功耗状态的;动态相机没有此功能
VzNL_EnableLowPowerMode(hDevice, VzTrue);
```

3.2.14 停流后等待多长时间进入低功耗模式 单位：秒

入参：hDevice 设备 handle

入参：dDuration 时长：单位(秒)

返回：返回错误码

int VzNL_EnterLowPowerDuration(VZNLHANDLE hDevice, double dDuration);

```
// 设置 3 秒进入低功耗
VzNL_EnterLowPowerDuration(hDevice, 3.);
```

3.2.15 设置高斯模式

入参：hDevice 设备 handle

入参：bHGauss 设备横向高斯

入参：bVGauss 设备纵向高斯

返回：返回错误码

int VzNL_EnableGauss(VZNLHANDLE hDevice, VzBool bHGauss, VzBool bVGauss);

int VzNL_IsEnableGauss(VZNLHANDLE hDevice, VzBool* pbHGauss, VzBool* pbVGauss);

```
VzBool bHGauss = VzFalse; VzBool bVGauss = VzFalse;
int nErrorCode = VzNL_IsEnableGauss(hDevice, &bHGauss, &bVGauss);
```

```
nErrorCode = VzNL_EnableGauss(hDevice, bHGauss, bVGauss);
```

3.2.16 设置触发输出控制模式

入参：hDevice 设备 handle

入参：eTriggerOutMode 触发输出控制类型

keStrobeTriggerOutMode_None 默认不输出

keStrobeTriggerOutMode_CameraLink 当触发模式为 keEyeTriggerMode_Master 时,相机会跟随帧率进行一个信号的输出（此模式下，会导致帧率下降)

keStrobeTriggerOutMode_UserControl 用户控制，此模式下需要用 VzNL_GenerateTriggerOutSignal 生成一个控制信号

返回：返回错误码

```
int VzNL_SetStrobeTriggerOutMode(VZNLHANDLE hDevice, EVzStrobeTriggerOutMode eTriggerOutMode);
EVzStrobeTriggerOutMode VzNL_GetStrobeTriggerOutMode(VZNLHANDLE hDevice, int* pnErrorCode);
```

3.2.17 生成一个触发输出信号

入参：hDevice 设备 handle

入参：bTriggerOn VzTrue 时为高电平，VzFalse 时为低电平信号

返回：返回错误码

```
int VzNL_GenerateTriggerOutSignal(VZNLHANDLE hDevice, VzBool bTriggerOn);
VzBool VzNL_IsTriggerOutOnSignal(VZNLHANDLE hDevice, int* pnErrorCode);
```

```
// 控制触发线生成触发信号，假设用此信号控制补光灯
VzNL_SetStrobeTriggerOutMode (hDevice, keStrobeTriggerOutMode_CameraLink );
// 打开补光灯
VzNL_GenerateTriggerOutSignal(hDevice, VzTrue);
// 关闭补光灯
VzNL_GenerateTriggerOutSignal(hDevice, VzFalse);
```

3.3 静态相机配置接口

头文件：#include " VZNL_SwingMotor.h "

库文件：VzNLDetect.dll / libVzNLDetect.so

是否为静态相机

```
VzBool VzNL_IsSupportSwingMotor(VZNLHANDLE hDevice, int* pnErrorCode);
```

入参: hDevice 设备句柄

出参: pnErrorCode 非空时返回错误码

返回: 返回 VzTrue 表示支持摆动机构

```
// 获取是否支持静态摆动
VzBool bIsSupportSwing = VzNL_IsSupportSwingMotor(hDevice, nullptr);
```

3.3.1 设置工作范围

```
int VzNL_SetSwingMotorWorkRange(VZNLHANDLE hDevice, double dNearDistance, double dFarDistance);
```

入参: hDevice 设备句柄

入参: dNearDistance 近距离

入参: dFarDistance 远距离

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.2 启用/禁用摆动机构

入参: hDevice 设备句柄

入参: bEnable 启用/禁用

返回: 返回 0 表示正确

int VzNL_EnableSwingMotor(VZNLHANDLE hDevice, VzBool bEnable);

3.3.3 是否启用了摆动机构

入参: hDevice 设备句柄

出参: pnErrorCode 非空时返回错误码

返回: 返回是否启用摆动机构的标志

VzBool VzNL_IsEnableSwingMotor(VZNLHANDLE hDevice, int* pnErrorCode);

3.3.4 打开/关闭激光器

int VzNL_EnableLaserLight(VZNLHANDLE hDevice, VzBool bEnable);

入参: hDevice 设备句柄

入参: bEnable 启用/禁用

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.5 是否启用了激光器

VzBool VzNL_IsEnableLaserLight(VZNLHANDLE hDevice);

入参: hDevice 设备句柄

返回: 是否打开激光器

3.3.6 调节激光器亮度

int VzNL_SetLaserLight(VZNLHANDLE hDevice, unsigned int nLight);

入参: hDevice 设备句柄

入参: nLight 激光器亮度(0~1000)

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.7 获取激光器亮度

unsigned int VzNL_GetLaserLight(VZNLHANDLE hDevice, int* pnErrorCode);

入参: hDevice 设备句柄

出参: pnErrorCode 非空时返回错误码

返回: 返回当前激光器亮度

3.3.8 设置角速度

int VzNL_SetSwingAngleSpeed(VZNLHANDLE hDevice, double dAngleSpeed);

入参: hDevice 设备句柄

入参: dAngleSpeed 角速度

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.9 获取摆动机构角速度

int VzNL_GetSwingAngleSpeed(VZNLHANDLE hDevice, double* pdAngleSpeed);

入参:　hDevice 设备句柄

出参: pdAngleSpeed 角速度

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.10 设置摆动机构起止角度

int VzNL_SetSwingMotorAngle(VZNLHANDLE hDevice, float nStartPos, float nEndPos);

入参:　hDevice 设备句柄

入参: fStartPos 开始角度

入参: fEndPos 结束角度

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.11 获取电机采图起止角度

int VzNL_GetSwingMotorAngle(VZNLHANDLE hDevice, float* pnStartPos, float* pnEndPos);

入参:　hDevice 设备句柄

出参: fStartPos 开始角度

出参: fEndPos 结束角度

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

```
// 获取开始结束角度
float fStartPos; float fEndPos;
VzNL_GetSwingMotorAngle(hDevice, &fStartPos, &fEndPos);

// 设置开启结束角度
VzNL_SetSwingMotorAngle(hDevice, fStartPos, fEndPos);
```

3.3.12 设置当前位置为开始/结束位置

int VzNL_SetSwingCurPosToMotorStartPos(VZNLHANDLE hDevice);

入参:　hDevice 设备句柄

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.13 设置当前位置位结束角度

int VzNL_SetSwingCurPosToMotorEndPos(VZNLHANDLE hDevice);

入参:　hDevice 设备句柄

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.14 旋转到某个位置

int VzNL_RotateSwing(VZNLHANDLE hDevice, float fAngle);

入参:　hDevice 设备句柄

入参：　fAngle 旋转到的角度(0~75°)

返回:　非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.15 旋转到开始位置

int VzNL_RotateSwingToStartPos(VZNLHANDLE hDevice);

入参: hDevice 设备句柄

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.16 设置摆动机构扫描模式

int VzNL_SetSwingScanMode(VZNLHANDLE hDevice, EVzSwingMotorScanMode eScanMode);

入参: hDevice 设备句柄

入参: eScanMode 扫描模式

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.17 获取摆动机构扫描模式

EVzSwingMotorScanMode VzNL_GetSwingScanMode(VZNLHANDLE hDevice);

入参: hDevice 设备句柄

返回: 摆动机构扫描模式

```
//keSwingMotorScanMode_Once,    一次
//keSwingMotorScanMode_Loop,    轮询
// 摆动一次
VzNL_SetSwingScanMode(hDevice,  keSwingMotorScanMode_Once);
```

3.3.18 获取扫描机构信息

int VzNL_GetSwingMotorInfo(VZNLHANDLE hDevice, SVzSwingMotorDevInfo* psSwingMotorInfo);

入参: hDevice 设备句柄

出参: psSwingMotorInfo 电机信息

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

3.3.19 获取当前角度

float VzNL_GetSwingCurAngle(VZNLHANDLE hDevice, int* pnErrorCode);

入参: hDevice 设备句柄

出参: pnErrorCode 错误码(当不为 nullptr 返回)

返回: 返回当前摆动机构角度(0~75°)

3.3.20 设置停顿时间

int VzNL_SetSwingStopTime(VZNLHANDLE hDevice, unsigned int nMillionSecond);

入参: hDevice 设备句柄

入参: nMillionSecond 停顿时间(ms)

返回: 非 0 表示失败,可以使用 VzNL_GetErrorInfo 获取

```
// 单次摆动并停止
#include <stdio.h>
#include <vector>
#include "VZNL_Common.h"
#include "VZNL_EyeConfig.h"
#include "VZNL_DetectLaser.h"
```

```
#include "VZNL_DetectConfig.h"
#include "VZNL_SwingMotor.h"
#include "windows.h"
#include <fstream>

static HANDLE s_hPauseEvent = NULL;

/// @name 自动检测回调函数
/// @{
void _AutoOutputLaserLineExCB(EVzResultDataType eDataType, SVzLaserLineData*
pLaserLinePoint, void* pParam)
{
    // 当需要的点云格式为SVzNLPointXYZRGBA时
    if (keResultDataType_PointXYZRGBA == eDataType)
    {
    }
}

// 显示错误信息
void _DisplayErrorInfo(const char* pszDesc, int nErrorCode)
{
    char szOutput[256];
    if (0 == nErrorCode)
    {
        sprintf_s(szOutput, 256, "%s\r\n", pszDesc);
    }
    else
    {
        char szErrorInfo[256] = { 0 };
        VzNL_GetErrorInfo(nErrorCode, szErrorInfo);

        sprintf_s(szOutput, 256, "%s [%d] %s\r\n", pszDesc, nErrorCode, szErrorInfo);
    }
    printf(szOutput);
}

// 搜索设备
void _ResearchDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
    // 搜索相机
    VzNL_ResearchDevice(keSearchDeviceFlag_EthLaserRobotEye);

    // 遍历相机
    int nDevCount = 0;
```

```
    VzNL_GetEyeCBDeviceInfo(nullptr, &nDevCount);
    if (nDevCount <= 0)
    {
        return;
    }

    vetDevice.resize(nDevCount);
    VzNL_GetEyeCBDeviceInfo(vetDevice.data(), &nDevCount);
}

// @brief 搜索且进行绑定设备
void _SearchAndBindDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
    int nErrorCode = 0;
    VzBool bCanResearch = VzFalse;
    do
    {
        bCanResearch = VzFalse;
        _ResearchDevice(vetDevice);
        std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
        while (itDevice != vetDevice.end())
        {
            // 相机没有被识别
            if (VzFalse == itDevice->bValidDevice)
            {
                SVzNLEyeCBInfo sCurDevInfo = *itDevice;
                SVzNLEthernetEyeConfigInfo sOutputEthConfigInfo;
                nErrorCode = VzNL_GetEthernetEyeConfigInfo(&sCurDevInfo,
&sOutputEthConfigInfo);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("获取网络信息失败", nErrorCode);
                }

                //
                if (0 == memcmp(sOutputEthConfigInfo.byLocalIP,
sOutputEthConfigInfo.sNetCardInfo.byLocalIP, 4))
                {
                    _DisplayErrorInfo("请查看是否有网络权限和管理员权限\r\n", 0);
                }
                else
                {
                    nErrorCode = VzNL_BindEthernetEye(&sCurDevInfo);
                    if (0 != nErrorCode)
```

```
                    {
                        _DisplayErrorInfo("绑定相机失败", nErrorCode);
                    }
                    bCanResearch = VzTrue;
                }
            }
            itDevice++;
        }
    } while (bCanResearch);
}

/**
 * @brief 设备状态改变CallBack函数
 * @param eStatus [out] 状态
 * @param pInfoParam [out] 状态参数
 */
void _OnNotifyStatusCB(EVzDeviceWorkStatus eStatus, void* pInfoParam)
{
    if (keDeviceWorkStatus_Device_Auto_Stop == eStatus)
    {
        printf("收到停止信号\r\n");
        if (nullptr != s_hPauseEvent)
        {
            SetEvent(s_hPauseEvent);
        }
    }
}

int main()
{
    // 创建停止信号
    s_hPauseEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    // 初始化SDK
    SVzNLConfigParam sConfigParam;
    sConfigParam.nDeviceTimeOut = 0;
    int nRet = VzNL_Init(&sConfigParam);
    if (0 != nRet)
    {
        _DisplayErrorInfo("查看是否有其他程序在使用SDK?", nRet);
    }

    // 枚举相机
    std::vector<SVzNLEyeCBInfo> vetDevice;
```

```
    _SearchAndBindDevice(vetDevice);

    //
    int nErrorCode = 0;
    if (vetDevice.size() > 0)
    {
        _DisplayErrorInfo("显示当前可用相机\r\n", 0);
        std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
        while (itDevice != vetDevice.end())
        {
            // 相机没有被识别
            if (VzFalse == itDevice->bValidDevice)
            {
                itDevice++;
            }

            char szOutput[256];
            sprintf_s(szOutput, 256, "----------------------准备打
开 %s----------------------\r\n", itDevice->byServerIP);
            _DisplayErrorInfo(szOutput, 0);

            itDevice++;
        }
    }

    //
    do
    {
        if (vetDevice.size() == 0)
        {
            break;
        }

        // 打开相机
        SVzNLOpenDeviceParam sOpenDevParam;
        VZNLHANDLE hDevice = VzNL_OpenDevice(&vetDevice[0], &sOpenDevParam, &nErrorCode);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("打开设备失败", nErrorCode);
            break;
        }
        else
        {
            char szOutput[256];
```

```
            sprintf_s(szOutput, 256, "打开相机%s成功\r\n", vetDevice[0].byServerIP);
            _DisplayErrorInfo(szOutput, 0);
        }
        VzNL_SetDeviceStatusNotify(hDevice, _OnNotifyStatusCB, NULL);

        // 创建激光线检测工具
        nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("创建检测激光线工具", nErrorCode);
            break;
        }
        else
        {
            _DisplayErrorInfo("创建激光线检测工具成功\r\n", 0);
        }

        // 设置到主模式
        nErrorCode = VzNL_SetTriggerMode(hDevice, keEyeTriggerMode_Master);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("设置主模式失败", nErrorCode);
            break;
        }
        else
        {
            _DisplayErrorInfo("主模式设置成功\r\n", 0);
        }

        unsigned int nMinFrameRate, nMaxFrameRate;
        VzNL_QueryParamRange(hDevice, keDeviceParamType_FrameRate, &nMinFrameRate,
&nMaxFrameRate);

        VzNL_SetFrameRate(hDevice, nMaxFrameRate);

        ResetEvent(s_hPauseEvent);

        VzBool bIsSupportMotor = VzNL_IsSupportSwingMotor(hDevice, nullptr);
        if (bIsSupportMotor)
        {
            VzNL_EnableCalibROI(hDevice, VzFalse);

            VzNL_EnableSwingMotor(hDevice, VzTrue);
        }
```

```
        // 开始流模式检测
        nErrorCode = VzNL_StartAutoDetectEx(hDevice, keResultDataType_PointXYZRGBA,
keFlipType_None, _AutoOutputLaserLineExCB, nullptr);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("开流失败", nErrorCode);
            break;
        }
        else
        {
            _DisplayErrorInfo("开流成功\r\n", 0);
        }

        // 等待摆动结束
        WaitForSingleObject(s_hPauseEvent, INFINITE);

        nErrorCode = VzNL_StopAutoDetect(hDevice);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("关流失败", nErrorCode);
            break;
        }
        else
        {
            _DisplayErrorInfo("关流成功\r\n", 0);
        }

        VzNL_EndDetectLaser(hDevice);

        VzNL_CloseDevice(hDevice);

    } while (false);

    // 销毁SDK
    VzNL_Destroy();

    if (nullptr != s_hPauseEvent)
    {
        CloseHandle(s_hPauseEvent);
        s_hPauseEvent = nullptr;
    }

    return 0;
```

```
}
```

3.4 VzNLSDK 网络眼睛配置接口

头文件：#include "VZNL_EyeConfig.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.4.1 获取设备信息

int VzNL_GetEthernetEyeConfigInfo(const SVzNLEyeCBInfo* pEyeCBInfo, SVzNLEthernetEyeConfigInfo* psConfigInfo);

入参： pEyeCBInfo 设备信息

入参： psConfigInfo 眼睛参数

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLEthernetEyeConfigInfo sOutputEthConfigInfo;
    int nErrorCode = VzNL_GetEthernetEyeConfigInfo(&pCBInfo[0], &sOutputEthConfigInfo);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
```

```
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.4.2 重置网络极光眼的配置信息

int VzNL_ConfigEthernetEyeConfigInfo(const SVzNLEyeCBInfo* pEyeCBInfo, unsigned char byEyeIP[4]);

入参： pEyeCBInfo 设备信息

入参： byEyeIP 要配置的眼睛 IP

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    unsigned char bySetEyeIP[4] = { 192, 168 , 10 , 113 };
    int nEthRet = VzNL_ConfigEthernetEyeConfigInfo(&pCBInfo[0], bySetEyeIP);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}
```

```
// 销毁SDK
VzNL_Destroy();
```

3.4.3 重置网络配置信息

入参： pEyeCBInfo 设备信息

入参： byEyeIP 眼睛 IP

入参： byMask 子网掩码

入参： byGateWay 网关

返回：返回 0 表示成功

```
int VzNL_ConfigEthernetInfo(const SVzNLEyeCBInfo* pEyeCBInfo,
                            unsigned char byEyeIP[4],
                            unsigned char byMask[4],
                            unsigned char byGateWay[4]);
```

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    unsigned char bySetEyeIP[4] = { 192, 168 , 10 , 113 };
    unsigned char byMask[4] = { 255, 255 , 255 , 0 };
    unsigned char byGW[4] = { 192, 168 ,10, 1 };
    int nEthRet = VzNL_ConfigEthernetInfo(&pCBInfo[0], bySetEyeIP, byMask, byGW);
}
```

```
if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.4.4 设置 IP 类型

入参： pEyeCBInfo 设备信息

入参： eIPType IP 类型[keNLEthernetEyeIPType_StaticIP:静态 IP keNLEthernetEyeIPType_DHCP:动态分配 IP]

返回： 返回 0 表示成功

int VzNL_ChangeEthernetIPType(const SVzNLEyeCBInfo* pEyeCBInfo, EVzNLEthernetEyeIPType eIPType);

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    int nEthRet = VzNL_ChangeEthernetIPType(&pCBInfo[0], keNLEthernetEyeIPType_StaticIP);
}

if (NULL != pCBInfo)
{
```

```
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.4.5 查询网络配置信息

入参： pEyeCBInfo 设备信息

出参： peIPType IP 类型[keNLEthernetEyeIPType_StaticIP:静态 IP keNLEthernetEyeIPType_DHCP:动态分配 IP]

出参： byEyeIP 眼睛 IP

出参： byMask 子网掩码

出参： byGateWay 网关

返回： 返回 0 表示成功

```
int VzNL_QueryEthernetInfo(const SVzNLEyeCBInfo* pEyeCBInfo,
                           EVzNLEthernetEyeIPType* peIPType,
                           unsigned char byEyeIP[4],
                           unsigned char byMask[4],
                           unsigned char byGateWay[4]);
```

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
```

```
    EVzNLEthernetEyeIPType eIPType;
    unsigned char bySetEyeIP[4] = { 192, 168 , 10 , 113 };
    unsigned char byMask[4] = { 255, 255 , 255 , 0 };
    unsigned char byGW[4] = { 192, 168 ,10, 1 };
    int nEthRet = VzNL_QueryEthernetInfo(&pCBInfo[0], &eIPType, bySetEyeIP, byMask, byGW);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.4.6 解绑设备

int VzNL_UnBindEthernetEye(const SVzNLEyeCBInfo* pEyeCBInfo);

入参： pEyeCBInfo 设备信息

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.4.7 绑定设备

int VzNL_BindEthernetEye(const SVzNLEyeCBInfo* pEyeCBInfo);

入参： pEyeCBInfo 设备信息

返回： 检测到激光点的个数

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;
```

```
pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo[0], &sOpenDevParam, &nErrorCode);
    If ((hDevice != NULL) && (nErrorCode == 0))
    {
        if (VzFalse == pCBInfo[0].bValidDevice)
        {
            SVzNLEyeCBInfo sCurDevInfo = pCBInfo[0];
            SVzNLEthernetEyeConfigInfo sOutputEthConfigInfo;
            nErrorCode = VzNL_GetEthernetEyeConfigInfo(&sCurDevInfo,
&sOutputEthConfigInfo);
            if (0 != nErrorCode)
            {
                _DisplayErrorInfo("获取网络信息失败", nErrorCode);
            }

            //
            if (0 == memcmp(sOutputEthConfigInfo.byLocalIP,
sOutputEthConfigInfo.sNetCardInfo.byLocalIP, 4))
            {
                _DisplayErrorInfo("请查看是否有网络权限和管理员权限\r\n", 0);
            }
            else
            {
                nErrorCode = VzNL_BindEthernetEye(&sCurDevInfo);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("绑定相机失败", nErrorCode);
                }
                bCanResearch = VzTrue;
            }
        }
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
```

```
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.5 VzNLSDK 激光检测接口

头文件：#include "VZNL_DetectLaser.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.5.1 开始激光检测

int VzNL_BeginDetectLaser(VZNLHANDLE hDevice);

入参：　hDevice　　设备句柄

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.2 结束激光检测

void VzNL_EndDetectLaser(VZNLHANDLE hDevice);

入参：　hDevice　　设备句柄

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
```

```
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
    if ((hDevice != NULL) && (nErrorCode == 0))
    {
        int nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }

        // 执行激光线检测相关函数
        nErrorCode = VzNL_EndDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.5.3 激光单线检测

int VzNL_DetectLaser(VZNLHANDLE hDevice, int nPointInterval);

入参： hDevice 设备句柄

nPointInterval 点的间隔

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.4 获取激光线结果点的个数

int VzNL_GetLaserResultPointCount(VZNLHANDLE hDevice);

入参： hDevice 设备句柄

返回： 检测到激光点的个数

3.5.5 获取激光线 2D 结果

int VzNL_GetLaser2DResult(VZNLHANDLE hDevice, SVzNL2DPosition* p2DPoint, int* pnCount);

入参： hDevice 设备句柄
pnCount 2D 点的个数
出参： p2DPoint 2D 点坐标首地址
返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.6 获取激光线 3D 结果

int VzNL_GetLaser3DResult(VZNLHANDLE hDevice, SVzNL3DPosition* p3DPoint, int* pnCount);

入参： hDevice 设备句柄
pnCount 3D 点的个数
出参： p3DPoint 3D 点坐标首地址
返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

int VzNL_GetLaserImageResult(VZNLHANDLE hDevice, SVzNLImageData** ppLeftImageData, SVzNLImageData** ppRightImageData);

入参： hDevice 设备句柄
出参： ppLeftImageData 左图像地址
ppRightImageData 右图像地址
返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
#include <stdio.h>
#include "VZNL_Common.h"

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
```

```
    if ((hDevice != NULL) && (nErrorCode == 0))
    {
        int nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }

        // 执行激光线检测相关函数
        SVzNL3DPosition* p3DPosition = NULL;
        Int n3DPointCount = 0;
        SVzNL2DPosition*    p2DPosition = NULL;
        int                 n2DPositionCount = 0;
        SVzNLImageData* pLeftImageData = NULL;
        SVzNLImageData* pRightImageData = NULL;
      nErrorCode = VzNL_DetectLaser(hDevice, 1);
    if (0 != nErrorCode)
    {
        _ShowErrorInfo(nErrorCode);
        return nErrorCode;
    }
    if (0 == VzNL_GetLaser3DResult(hDevice, NULL, &n3DPointCount))
    {
        p3DPosition = (SVzNL3DPosition*)malloc(n3DPointCount * sizeof(SVzNL3DPosition));
        VzNL_GetLaser3DResult(hDevice, p3DPosition, &n3DPointCount);
    }

    if (0 == VzNL_GetLaser2DResult(hDevice, NULL, &n2DPositionCount))
    {
        p2DPosition = (SVzNL2DPosition*)malloc(n2DPositionCount *
sizeof(SVzNL2DPosition));
        VzNL_GetLaser2DResult(hDevice, p2DPosition, &n2DPositionCount);
    }
    for (n = 0; n < n2DPositionCount && n < n3DPointCount; n++)
    {
        printf("index %d Left X:%d Y:%d Right X:%d Y:%d \t X:%.5f Y:%.5f Z:%.5f\n",
        p3DPosition[n].nPointIdx, p2DPosition[n].ptLeft2D.x, p2DPosition[n].ptLeft2D.y,
        p2DPosition[n].ptRight2D.x, p2DPosition[n].ptRight2D.y,
        p3DPosition[n].pt3D.x, p3DPosition[n].pt3D.y, p3DPosition[n].pt3D.z);
    }

    VzNL_ReleaseImage(&pLeftImageData);
    VzNL_ReleaseImage(&pRightImageData);
```

```
    if (NULL != p3DPosition)
    {
       free(p3DPosition);
        p3DPosition = NULL;
    }
     if (NULL != p2DPosition)
     {
        free(p2DPosition);
        p2DPosition = NULL;
     }

        nErrorCode = VzNL_EndDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
             return;
        }
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.5.7 设置波峰波谷凹槽最小深度

int VzNL_SetLaserMinSlotDeep(VZNLHANDLE hDevice, double dMinSlotDeep);

入参： hDevice 设备句柄

出参： dMinSlotDeep 凹槽最小深度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.8 取激光凹槽数据结果的波峰点的个数

int VzNL_GetLaserSlotResultPeakPonitCount(VZNLHANDLE hDevice);

入参： hDevice 设备句柄

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.9 获取激光凹槽数据结果的波谷点的个数

int VzNL_GetLaserSlotResultValleyPonitCount(VZNLHANDLE hDevice);

入参： hDevice 设备句柄

出参： dMinSlotDeep 凹槽最小深度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.10 获取激光凹槽 2D 结果

int VzNL_GetLaserSlot2DResult(VZNLHANDLE hDevice, SVzNL2DPosition* pPeakPoint, int* pnPeakPointCount, SVzNL2DPosition* pValleyPoint, int* pnValleyPointCount);

入参： hDevice 设备句柄

出参： dMinSlotDeep 凹槽最小深度

pPeakPoint 波峰点，内存空间由用户分配

pnPeakPointCount pPeakPoint 的结构个数

pValleyPoint 波谷点，内存空间由用户分配

pnValleyPointCount pValleyPoint 的结构个数

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.11 获取激光凹槽 3D 结果

int VzNL_GetLaserSlot3DResult(VZNLHANDLE hDevice, SVzNL3DPosition* pPeakPoint, int* pnPeakPointCount, SVzNL3DPosition* pValleyPoint, int* pnValleyPointCount);

入参： hDevice 设备句柄

出参： dMinSlotDeep 凹槽最小深度

pPeakPoint 波峰点，内存空间由用户分配

pnPeakPointCount pPeakPoint 的结构个数

pValleyPoint 波谷点，内存空间由用户分配

pnValleyPointCount pValleyPoint 的结构个数

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

激光线自动检测并回调

3.5.12 开始自动检测

int VzNL_StartAutoDetect(VZNLHANDLE hDevice, EVzFlipType eFlipType, VzNL_GetAutoDtectResultCB pCB， void* pCBParam);

入参： hDevice 设备句柄

eFlipType 激光线方向

pCB 回调接口

pCBParam 对调中会原值返回

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

回调函数声明方式

typedef int(*VzNL_GetAutoDtectResultCB)(SVzNL3DPosition* p3DPoint, SVzNL2DPosition* p2DPoint, int nCount, unsigned long long nTimeStamp, double dTotleOffset, double dStep,SVzNLImageData* pLeftEye, SVzNLImageData* pRightEye， void* pParam);

结束自动检测

int VzNL_StopAutoDetect(VZNLHANDLE hDevice);

入参： hDevice 设备句柄

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

是否在自动检测中

VzBool VzNL_IsAutoDetecting(VZNLHANDLE hDevice, int* pnErrorCode);

```
#include <stdio.h>
#include "VZNL_Common.h"

// 激光线回调
int _GetAutoDetectResultCB(SVzNL3DPosition* p3DPosition, SVzNL2DPosition* p2DPosition,
int n3DPointCount, unsigned long long nTimeStamp, double dTotleStep, double dStep,
SVzNLImageData* pLeftImage, SVzNLImageData* pRightImage, void* pParam)
{
    if (n3DPointCount > 0)
    {
        printf("Auto Detect Count %.1f %.1f %.1f / %.1lf\r\n", p3DPosition[0].pt3D.x,
p3DPosition[0].pt3D.y, p3DPosition[0].pt3D.z, dTotleStep);
    }

// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
    if ((hDevice != NULL) && (nErrorCode == 0))
    {
        int nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
```

```
        {
            return;
        }

        // 执行激光线检测相关函数
        nErrorCode = VzNL_StartAutoDetect(hDevice, keFlipType_Vertical,
_GetAutoDetectResultCB, hDevice);
        if (0 != nErrorCode)
        {
            return;
        }

        // 等待 5s(采集 5s 数据)
        Sleep(5000);

        VzNL_StopAutoDetect(hDevice);
        nErrorCode = VzNL_EndDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.5.13 激光线自动检测函数

int VzNL_StartAutoDetectEx(VZNLHANDLE hDevice, EVzResultDataType eResultType, EVzFlipType eFlipType, VzNL_AutoOutputLaserLineExCB pCB, void* pCBParam);

入参：　hDevice　　设备句柄

　　eResultType　　结果类型

　　eFlipType　　激光线方向

　　pCB　　回调函数

　　pCBParam　　回调参数

返回：　0 为正确，其他值可通过 VzNL_GetErrorInfo 获取

回调函数：

typedef void(*VzNL_AutoOutputLaserLineExCB)(EVzResultDataType eDataType, SVzLaserLineData* pLaserLinePoint, void* pParam);

eDataType　　数据类型，与开始检测入参相同

枚举值	3d 数据类型	2d 数据类型
keResultDataType_Position	SVzNL3DPosition	SVzNL2DPosition
keResultDataType_PointF	SVzNL3DPointF	SVzNL2DLRPoint
keResultDataType_PointXYZ,	SVzNLPointXYZ	SVzNL2DLRPoint
keResultDataType_PointXYZRGBA	SVzNLPointXYZRGBA	SVzNL2DLRPoint

pLaserLinePoint　　输出激光线数据

pParam　　返回回调时传入的参数

```
typedef struct
{
    void*               p3DPoint;       //< 3D 点  根据 ResultDataType 使用不同的数据结构
    void*               p2DPoint;       //< 2D 点  根据 ResultDataType 使用不同的数据结构
    int                 nPointCount;    //< 2D 3D  数据个数
    double              dTotleOffset;   //<  总偏移
    double              dStep;          //<  距离上一条线的偏移
    unsigned long long  llFrameIdx;     //<  帧号
    unsigned long long  llTimeStamp;    //<  时间戳
    unsigned int        nEncodeNo;      //<  编码器值
} SVzLaserLineData;
```

结束自动检测

int VzNL_StopAutoDetect(VZNLHANDLE hDevice);

入参：　hDevice　　设备句柄

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

```
#include <stdio.h>
#include "VZNL_Common.h"

//
void _AutoOutputLaserLineExCB(EVzResultDataType eDataType, SVzLaserLineData* pLaserLinePoint, void* pParam)
{
    if (nullptr == pLaserLinePoint)
    {
        return;
    }
     if (pLaserLinePoint->nPointCount > 0)
    {
        if (keResultDataType_Position == eDataType)      //< Position数据类型 使用 SVzNL3DPosition, SVzNL2DPosition
```

```
        {
            SVzNL3DPosition* p3DPosition = (SVzNL3DPosition*)pLaserLinePoint->p3DPoint;
            SVzNL2DPosition* p2DPosition = (SVzNL2DPosition*)pLaserLinePoint->p2DPoint;
            printf("{%.11f, %.11f, %.11f}-{%d,%d}-{%d,%d}\r\n", p3DPosition->pt3D.x,
p3DPosition->pt3D.y, p3DPosition->pt3D.z,
                p2DPosition->ptLeft2D.x, p2DPosition->ptLeft2D.y,
p2DPosition->ptRight2D.x, p2DPosition->ptRight2D.y);
        }
        else if (keResultDataType_PointF == eDataType)        //< PointF数据类型    使用
SVzNL3DPointF SVzNL2DPointF
        {
            SVzNL3DPointF* p3DPoint = (SVzNL3DPointF*)pLaserLinePoint->p3DPoint;
            SVzNL2DLRPoint* p2DPoint = (SVzNL2DLRPoint*)pLaserLinePoint->p2DPoint;
            printf("{%.11f, %.11f, %.11f}-{%d,%d}-{%d,%d}\r\n", p3DPoint->x,
p3DPoint->y, p3DPoint->z,
                p2DPoint->sLeft.x, p2DPoint->sLeft.y, p2DPoint->sRight.x,
p2DPoint->sRight.y);
        }
        else if (keResultDataType_PointXYZ == eDataType)      //< PointXYZ数据类型  使用
PointXYZ
        {
            SVzNLPointXYZ* p3DPoint = (SVzNLPointXYZ*)pLaserLinePoint->p3DPoint;
            SVzNL2DLRPoint* p2DPoint = (SVzNL2DLRPoint*)pLaserLinePoint->p2DPoint;
            printf("{%.11f, %.11f, %.11f}-{%d,%d}-{%d,%d}\r\n", p3DPoint->x,
p3DPoint->y, p3DPoint->z,
                p2DPoint->sLeft.x, p2DPoint->sLeft.y, p2DPoint->sRight.x,
p2DPoint->sRight.y);
        }
        else if (keResultDataType_PointGray == eDataType)     //< PointXYZRGB数据  使用
SVzPointXYZRGB
        {
        }
        else if (keResultDataType_PointXYZRGBA == eDataType) //< PointXYZRGB数据  使用
SVzPointXYZRGB
        {
        }
    }
}

// 主函数相关片段
// 初始化SDK
SVzNLConfigParam sConfigParam;
sConfigParam.nDeviceTimeOut = 0;
int nRet = VzNL_Init(&sConfigParam);
```

```
if (0 != nRet)
{
   printf("SDK 多开\n")
}

SVzNLEyeCBInfo* pCBInfo = NULL;
int nDevIdx = 0; int nDevCount = 0;

VzNL_GetEyeCBDeviceInfo(NULL, &nDevCount);
if (nDevCount == 0)
    return;

pCBInfo = (SVzNLEyeCBInfo*)malloc(sizeof(SVzNLEyeCBInfo) * nDevCount);
VzNL_GetEyeCBDeviceInfo(pCBInfo, &nDevCount);

if (0 < nDevCount)
{
    SVzNLOpenDeviceParam sOpenDevParam;
    VZNLHANDLE hDevice = VzNL_OpenDevice(&pCBInfo [0], &sOpenDevParam, &nErrorCode);
    if ((hDevice != NULL) && (nErrorCode == 0))
    {
        int nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }

        nErrorCode = VzNL_StartAutoDetectEx(hDevice, keResultDataType_PointXYZRGBA,
keFlipType_None, _AutoOutputLaserLineExCB, hDevice);
        if(0 != nErrorCode)
        {
            return;
        }

        // 等待 5s(采集 5s 数据)
        Sleep(5000);

        VzNL_StopAutoDetect(hDevice);

        nErrorCode = VzNL_EndDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }
```

```
    }
    VzNL_CloseDevice(hDevice);
}

if (NULL != pCBInfo)
{
    free(pCBInfo);
    pCBInfo = NULL;
}

// 销毁SDK
VzNL_Destroy();
```

3.5.14 设置/获取激光检测门限

int VzNL_SetLaserThres(VZNLHANDLE hDevice, int nLaserThres);

入参： hDevice 设备句柄

nLaserThres 激光门限

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

int VzNL_GetLaserThres(VZNLHANDLE hDevice, int* pnLaserThres);

入参： hDevice 设备句柄

返回： pnLaserThres 激光门限

返回值 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.15 激光高度标定

int VzNL_SetLaserStandard(VZNLHANDLE hDevice, double dStandardObjectHeight, const SVzNLROIRect* pTopROI, const int nTopROICount,const SVzNLROIRect* pBottomROI,const int nBottomROICount);

入参： hDevice 设备句柄

dStandardObjectHeight 标块的高度

pTopROI 标块表面 ROI 结构体数组首地址

nTopROICount 标块表面 ROI 的个数

pBottomROI 参考面/零面的 ROI 结构体数组首地址

nBottomROICount 参考面/零面的 ROI 的个数

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

int VzNL_ClearLaserStandard(VZNLHANDLE hDevice);

入参： hDevice 设备句柄

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.16 设置过滤高度

int VzNL_ConfigLaserLineFilterHeight(VZNLHANDLE hDevice, double dFilterHeight);

入参： hDevice 设备句柄

入参： dFilterHeight 过滤高度（未高度标定时，大于此高度的数据不输出，高度标定后，

小于此数据的高度不输出）

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.17 获取过滤高度

int VzNL_GetLaserLineFilterHeight(VZNLHANDLE hDevice, double* pdFilterHeight);

入参： hDevice 设备句柄

出参： pdFilterHeight 过滤高度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.18 启用/禁用过滤高度

int VzNL_EnableLaserLineFilterHeight(VZNLHANDLE hDevice, VzBool bEnable);

入参： hDevice 设备句柄

入参： bEnable 启用高度过滤

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.19 是否启用过滤高度

VzBool VzNL_IsEnableLaserLineFilterHeight(VZNLHANDLE hDevice, int* pnErrorCode);

入参： hDevice 设备句柄

出参： pnErrorCode 错误码

返回： 启用返回 VzTrue 失败返回 VzFalse

3.5.20 设置杂点过滤阈值

int VzNL_ConfigLaserLineMaxDeviation(VZNLHANDLE hDevice, double dMaxDeviation);

入参： hDevice 设备句柄

入参： dMaxDeviation 杂点过滤阈值

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息

3.5.21 设置偏移量

int VzNL_ConfigLaserBeginOffsetValue(VZNLHANDLE hDevice, double dOffsetValue);

入参： hDevice 设备句柄

入参： dOffsetValue 偏移位置

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息

3.5.22 设置传送带速度值

int VzNL_ConfigLaserObjRunSpeedValue(VZNLHANDLE hDevice, EVzObjRunDirect eDirect, double dSpeed);

入参： hDevice 设备句柄

eDirect keObjRunDirect_Plus 为正方向,

keObjRunDirect_Minus 为反方向

dSpeed 传送带移动速度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.23 是否标定过

VzBool VzNL_LaserIsCalibration(VZNLHANDLE hDevice, int* pErrorCode);

入参： hDevice 设备句柄

pErrorCode 错误代码

返回： 若返回 VzTrue 则说明标定过

3.5.24 计算平均高度/最低高度/最高高度

void VzNL_CalcLaserZHeight(SVzNL3DPosition* p3DPoint, int nCount, double* pdAvgHeight, double* pdMinHeight, double* pdMaxHeight);

入参： p3DPoint 要计算的 3D 点【Array】

入参： nCount 点个数

入参： pdAvgHeight 平均高度

入参： pdMinHeight 最小高度

入参： pdMaxHeight 最大高度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.5.25 设置点云处理模式

入参： hDevice 设备句柄

入参： ePointCloudProcMode

*kePointCloudProcMode_Speed 速度计算模式

*kePointCloudProcMode_Encoder 编码器模式

*kePointCloudProcMode_FixedStep 固定步长模式

返回： 成功返回 0，否则为其他错误码

int VzNL_SetPointCloudProcMode(VZNLHANDLE hDevice, EVzPointCloudProcMode ePointCloudProcMode);

3.5.26 设置固定步长

设置前请先设置为 kePointCloudProcMode_FixedStep 模式

int VzNL_SetLaserLineFixedStep(VZNLHANDLE hDevice, double dStep);

3.5.27 配置转动轴半径

入参： hDevice 设备句柄

入参： dRadius 转动轴半径

返回： 成功返回 0，否则为其他错误码

int VzNL_ConfigLaserRollerRadius(VZNLHANDLE hDevice, double dRadius);

3.5.28 设置编码器脉冲精度

入参：hDevice 设备句柄

入参： nPulsePerRound 脉冲个数

返回： 成功返回 0，否则为其他错误

int VzNL_SetLaserEncoderResolution(VZNLHANDLE hDevice, unsigned int nPulsePerRound);

3.5.29 设置数据采集间隔

入参： hDevice 设备句柄

入参： nCaptureInterval 采集间隔

返回： 成功返回 0，否则为其他错误码

int VzNL_SetLaserCaptureInterval(VZNLHANDLE hDevice, unsigned int nCaptureInterval);

3.5.30 激光检测填充点模式

入参： hDevice 设备句柄

入参： bFillPoint VzTrue 为填充，VzFalse 为不填充(默认为 VzFalse)

返回： 成功返回 0，否则为其他错误码

int VzNL_EnableFillLaserPoint(VZNLHANDLE hDevice, VzBool bFillPoint);

3.5.31 激光检测扫描 RGB 原图的获取

入参： hDevice 设备句柄

出参： ppResultImage 输出 RGB 图像，需要用户调用 VzNL_ReleaseImage 进行数据释放

int VzNL_GetAutoDetectResultSurface(VZNLHANDLE hDevice, SVzNLImageData** ppResultImage);

```
#include <stdio.h>
#include <vector>
#include <list>
#include "VZNL_Common.h"
#include "VZNL_EyeConfig.h"
#include "VZNL_DetectLaser.h"
#include "VZNL_Graphics.h"
#include "VZNL_DetectConfig.h"
#include "VZNL_Utils.h"
#include "VZNL_SwingMotor.h"
#include "VZNL_RGBConfig.h"
#ifdef _WIN32
#include "windows.h"
#else
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#endif
#include <fstream>
#include <atomic>


/*****************************************************************************
 * File: vizumsdk_save_color_image_and_depth_map.cpp
 * Detail: 静态相机 第三目图像 点云图像 深度图 生成的 Sample
 * Author: Vizum
 * Date: 20210609
 *****************************************************************************/
```

```
static SVzVideoResolution g_sRGBVideoRes;
static SVzNLPointXYZRGBA* g_p2DToPointMap = nullptr;
static unsigned char* g_p2DRGBPic = nullptr;
static bool* g_pb2DInvalidPt = nullptr;
static std::atomic<bool> g_bPauseEvent(true);

/// @name 自动检测回调函数
/// @{
void _AutoOutputLaserLineExCB(EVzResultDataType eDataType, SVzLaserLineData* pLaserLinePoint, void*
pParam)
{
    // 当需要的点云格式为SVzNLPointXYZRGBA时
    if (keResultDataType_PointXYZRGBA == eDataType)
    {
        SVzNLPointXYZRGBA* p3DPoint = (SVzNLPointXYZRGBA*)pLaserLinePoint->p3DPoint;
        SVzNL2DLRPoint* p2DPoint = (SVzNL2DLRPoint*)pLaserLinePoint->p2DPoint;
        for (int nPtIdx = 0; nPtIdx < pLaserLinePoint->nPointCount; nPtIdx++)
        {
            if ((p2DPoint->sLeft.x < 0) || (p2DPoint->sLeft.x >= g_sRGBVideoRes.nFrameWidth))
            {
                p3DPoint++;
                p2DPoint++;
                continue;
            }

            if ((p2DPoint->sLeft.y < 0) || (p2DPoint->sLeft.y >= g_sRGBVideoRes.nFrameHeight))
            {
                p3DPoint++;
                p2DPoint++;
                continue;
            }

            int nPos = p2DPoint->sLeft.x + p2DPoint->sLeft.y * g_sRGBVideoRes.nFrameWidth;

            // 更新Depth Map
            g_p2DToPointMap[nPos] = *p3DPoint;
            g_pb2DInvalidPt[nPos] = true;

            // 更新点云RGB图
            unsigned char* pRGB = (unsigned char*)&p3DPoint->nRGB;
            unsigned char* pCurRGB = g_p2DRGBPic + nPos * 3;
            pCurRGB[0] = pRGB[0];
            pCurRGB[1] = pRGB[1];
            pCurRGB[2] = pRGB[2];
```

```
                p3DPoint++;
                p2DPoint++;
            }
        }
}

/// @brief
///  获取文件纯路径
int _GetPurePath(const char* szFilePath, char szPurePath[VZ_MAXPATH])
{
        int nFindPos = -1;
        const char* pPos = szFilePath;
        while ('\0' != *pPos)
        {
            if ('/' == *pPos || '\\' == *pPos)
            {
                nFindPos = static_cast<int>(pPos - szFilePath);
            }
            pPos++;
        }

        if (-1 == nFindPos)
            return -1;

        memcpy(szPurePath, szFilePath, static_cast<unsigned int>(nFindPos));
        szPurePath[nFindPos] = '\0';
        return 0;
}

//  显示错误信息
void _DisplayErrorInfo(const char* pszDesc, int nErrorCode)
{
        char szOutput[256];
        if (0 == nErrorCode)
        {
#ifdef _WIN32
            sprintf_s(szOutput, 256, "%s\r\n", pszDesc);
#else
            sprintf(szOutput, "%s\r\n", pszDesc);
#endif
        }
        else
        {
            char szErrorInfo[256] = { 0 };
```

```
            VzNL_GetErrorInfo(nErrorCode, szErrorInfo);
#ifdef _WIN32
            sprintf_s(szOutput, 256, "%s [%d] %s\r\n", pszDesc, nErrorCode, szErrorInfo);
#else
            sprintf(szOutput, "%s [%d] %s\r\n", pszDesc, nErrorCode, szErrorInfo);
#endif
        }
        printf("%s", szOutput);
}

// 搜索设备
void _ResearchDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
        // 搜索相机
        VzNL_ResearchDevice(keSearchDeviceFlag_EthLaserRobotEye);

        // 遍历相机
        int nDevCount = 0;
        VzNL_GetEyeCBDeviceInfo(nullptr, &nDevCount);
        if (nDevCount <= 0)
        {
            return;
        }

        vetDevice.resize(nDevCount);
        VzNL_GetEyeCBDeviceInfo(vetDevice.data(), &nDevCount);
}

// @brief 搜索且进行绑定设备
void _SearchAndBindDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
        int nErrorCode = 0;
        VzBool bCanResearch = VzFalse;
        do
        {
            bCanResearch = VzFalse;
            _ResearchDevice(vetDevice);
            std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
            while (itDevice != vetDevice.end())
            {
                // 相机没有被识别
                if (VzFalse == itDevice->bValidDevice)
                {
                    SVzNLEyeCBInfo sCurDevInfo = *itDevice;
```

```
                        SVzNLEthernetEyeConfigInfo sOutputEthConfigInfo;
                        nErrorCode = VzNL_GetEthernetEyeConfigInfo(&sCurDevInfo, &sOutputEthConfigInfo);
                        if (0 != nErrorCode)
                        {
                            _DisplayErrorInfo("获取网络信息失败", nErrorCode);
                        }

                        //
                        if (0 == memcmp(sOutputEthConfigInfo.byLocalIP,
sOutputEthConfigInfo.sNetCardInfo.byLocalIP, 4))
                        {
                            _DisplayErrorInfo("请查看是否有网络权限和管理员权限\r\n", 0);
                        }
                        else
                        {
                            nErrorCode = VzNL_BindEthernetEye(&sCurDevInfo);
                            if (0 != nErrorCode)
                            {
                                _DisplayErrorInfo("绑定相机失败", nErrorCode);
                            }
                            bCanResearch = VzTrue;
                        }
                    }
                    itDevice++;
                }
        } while (bCanResearch);
}

/**
 * @brief 设备状态改变CallBack函数
 * @param eStatus [out] 状态
 * @param pInfoParam [out] 状态参数
 */
void _OnNotifyStatusCB(EVzDeviceWorkStatus eStatus, void* pInfoParam)
{
    if (keDeviceWorkStatus_Device_Auto_Stop == eStatus)
    {
        printf("收到停止信号\r\n");
        g_bPauseEvent = true;
    }
}

int main()
{
```

```
	g_bPauseEvent = true;

	// 初始化SDK
	SVzNLConfigParam sConfigParam;
	sConfigParam.nDeviceTimeOut = 0;
	int nRet = VzNL_Init(&sConfigParam);
	if (0 != nRet)
	{
		_DisplayErrorInfo("查看是否有其他程序在使用SDK?", nRet);
		return 0;
	}

	VzNL_SetLogLevel(keNLLogLevel_Information, keNLLogType_File);

	// 枚举相机
	std::vector<SVzNLEyeCBInfo> vetDevice;
	_SearchAndBindDevice(vetDevice);

	//
	int nErrorCode = 0;
	std::vector<VZNLHANDLE>		vetHandles;
	if (vetDevice.size() > 0)
	{
		_DisplayErrorInfo("显示当前可用相机\r\n", 0);
		std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
		while (itDevice != vetDevice.end())
		{
			// 相机没有被识别
			if (VzFalse == itDevice->bValidDevice)
			{
				itDevice++;
				continue;
			}

			char szOutput[256];
#ifdef _WIN32
			sprintf_s(szOutput, 256, "----------------------准备打开  %s----------------------\r\n",
itDevice->byServerIP);
#else
			sprintf(szOutput, "----------------------准备打开  %s----------------------\r\n",
itDevice->byServerIP);
#endif
			_DisplayErrorInfo(szOutput, 0);
```

```
                // 打开相机
                SVzNLOpenDeviceParam sOpenDevParam;
                VZNLHANDLE hDevice = VzNL_OpenDevice(&(*itDevice), &sOpenDevParam,
&nErrorCode);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("打开设备失败", nErrorCode);
                    break;
                }
                else
                {
                    char szOutput[256];
#ifdef _WIN32
                    sprintf_s(szOutput, 256, "打开相机%s成功\r\n", itDevice->byServerIP);
#else
                    sprintf(szOutput, "打开相机%s成功\r\n", itDevice->byServerIP);
#endif
                    _DisplayErrorInfo(szOutput, 0);
                }

                if (NULL != hDevice)
                {
                    vetHandles.push_back(hDevice);
                }

                VzNL_SetOutputImageFormat(keVzNLImageType_BGR888);

                if (VzFalse == VzNL_IsSupportRGBCamera(hDevice, nullptr))
                {
                    _DisplayErrorInfo("当前设备不支持RGB", nErrorCode);
                    break;
                }

                if (VzFalse == VzNL_IsSupportSwingMotor(hDevice, nullptr))
                {
                    break;
                }

                // 启用RGB Sensor
                VzNL_EnableRGB(hDevice, VzTrue);

                // 注册回调
                VzNL_SetDeviceStatusNotify(hDevice, _OnNotifyStatusCB, NULL);
```

```
                // 创建激光线检测工具
                nErrorCode = VzNL_BeginDetectLaser(hDevice);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("创建检测激光线工具", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("创建激光线检测工具成功\r\n", 0);
                }

                VzNL_GetRGBResolution(hDevice, &g_sRGBVideoRes);

                int nFrameSize = g_sRGBVideoRes.nFrameWidth * g_sRGBVideoRes.nFrameHeight;
                g_p2DToPointMap = new SVzNLPointXYZRGBA[nFrameSize];
                g_p2DRGBPic = new unsigned char[nFrameSize * 3];
                g_pb2DInvalidPt = new bool[nFrameSize];
                memset(g_pb2DInvalidPt, 0, sizeof(bool) * nFrameSize);

                // 设置到主模式
                nErrorCode = VzNL_SetTriggerMode(hDevice, keEyeTriggerMode_Master);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("设置主模式失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("主模式设置成功\r\n", 0);
                }

                _DisplayErrorInfo("启用计算中间2D图像模式", 0);

                // 采集激光线
                unsigned int nMinFrameRate, nMaxFrameRate;
                VzNL_QueryParamRange(hDevice, keDeviceParamType_FrameRate, &nMinFrameRate,
&nMaxFrameRate);

                VzNL_SetFrameRate(hDevice, nMaxFrameRate);

                g_bPauseEvent = false;

                VzNL_EnableSwingMotor(hDevice, VzTrue);
```

```
                // 开始流模式检测
                nErrorCode = VzNL_StartAutoDetectEx(hDevice, keResultDataType_PointXYZRGBA,
keFlipType_None, _AutoOutputLaserLineExCB, nullptr);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("开流失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("开流成功\r\n", 0);
                }

                // 等待结束信号
                while (false == g_bPauseEvent)
                {
#ifdef _WIN32
                    Sleep(1);
#else
                    usleep(1);
#endif
                }

                nErrorCode = VzNL_StopAutoDetect(hDevice);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("关流失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("关流成功\r\n", 0);
                }

                SVzNLImageData* psCenterImage = nullptr;
                VzNL_GetAutoDetectResultSurface(hDevice, &psCenterImage);

                // 存储灰度图
                char szPurePath[VZ_MAXPATH] = { 0 };
#ifdef _WIN32
                char szModule[VZ_MAXPATH] = { 0 };
                GetModuleFileNameA(nullptr, szModule, VZ_MAXPATH);
                _GetPurePath(szModule, szPurePath);
#else
```

```
                getcwd(szPurePath, VZ_MAXPATH);
#endif
                char szGrayFile[VZ_MAXPATH] = { 0 };
                if (psCenterImage)
                {
#ifdef _WIN32
                    sprintf_s(szGrayFile, VZ_MAXPATH, "%s\\Gray.png", szPurePath);
#else
                    sprintf(szGrayFile, "%s\\Gray.png", szPurePath);
#endif
                    VzNL_SaveImage(szGrayFile, psCenterImage);
                }

                if (g_p2DRGBPic)
                {
                    // 根据点云生成的RGB图像
                    SVzNLImageData sImageData;
                    sImageData.byBitDepth = 8;
                    sImageData.eImageType = keVzNLImageType_BGR888;
                    sImageData.nBufferSize = nFrameSize * 3;
                    sImageData.nChannels = 3;
                    sImageData.nHeight = g_sRGBVideoRes.nFrameHeight;
                    sImageData.nWidth = g_sRGBVideoRes.nFrameWidth;
                    sImageData.pBuffer = g_p2DRGBPic;
                    sImageData.ptOriPos.x = 0;
                    sImageData.ptOriPos.y = 0;
#ifdef _WIN32
                    sprintf_s(szGrayFile, VZ_MAXPATH, "%s\\PointRGB.png", szPurePath);
#else
                    sprintf(szGrayFile, "%s\\PointRGB.png", szPurePath);
#endif
                    VzNL_SaveImage(szGrayFile, &sImageData);
                }

                // 存储深度图
                if (nullptr != g_p2DToPointMap)
                {
                    char szTifFile[VZ_MAXPATH] = { 0 };
#ifdef _WIN32
                    sprintf_s(szTifFile, VZ_MAXPATH, "%s\\Depth.tif", szPurePath);
#else
                    sprintf(szTifFile, "%s\\Depth.tif", szPurePath);
#endif
                    VzNL_SaveDepthMapTiffImage(szTifFile, g_sRGBVideoRes.nFrameWidth,
```

```
g_sRGBVideoRes.nFrameHeight, keResultDataType_PointXYZRGBA, g_p2DToPointMap);
                    }

                    // 输出点云映射数据
                    if (nullptr != g_p2DToPointMap)
                    {
                        delete[] g_p2DToPointMap;
                        g_p2DToPointMap = nullptr;
                    }
                    if (nullptr != g_pb2DInvalidPt)
                    {
                        delete[] g_pb2DInvalidPt;
                        g_pb2DInvalidPt = nullptr;
                    }

                    if (psCenterImage)
                    {
                        VzNL_ReleaseImage(&psCenterImage);
                    }
                    VzNL_EndDetectLaser(hDevice);
                    VzNL_CloseDevice(hDevice);

                    itDevice++;
                }
            }

            // 销毁SDK
            VzNL_Destroy();

            getchar();

            return 0;
}
```

3.6 工具类接口

头文件：#include "VZNL_Utils.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.6.1 检测运行环境

出参: 环境信息

返回: 返回 VzTrue 说明一切正常，返回 VzFalse 表示有信息不匹配的地方

int VzNL_CheckPlatform(SVzCheckPlatformInfo* psPlatformInfo);

3.6.2 获取当前系统列表

入参： hDevice 设备 Handle

出参： pnCount 输出系统的个数

返回： 系统列表数组

SVzSystemProjectInfo* VzNL_GetSystemPrjArray(VZNLHANDLE hDevice, int* pnCount);

3.6.3 设置当前系统索引

入参： hDevice 设备 Handle

入参： nCurSel 要配置为第几个系统为运行时的系统

返回： 返回 0 为正确，其他错误码请用 GetErrorString。

int VzNL_SwitchSystemPrj(VZNLHANDLE hDevice, int nCurSel);

3.6.4 获取当前系统索引

入参： hDevice 设备 Handle

返回： 当前系统的索引

int VzNL_GetCurrentSystemPrj(VZNLHANDLE hDevice);

3.6.5 查询运行时间(s)

入参: hDevice 设备 Handle

出参: pnErrorCode 错误码

返回: 运行时间 s

double VzNL_GetSystemRuntime(VZNLHANDLE hDevice, int* pnErrorCode);

3.6.6 创建点云后处理工具 Handle

VZNLPOINTCLOUDHANDLE VzNL_CreateCloudPointAPTool();

当配置自适应参数时,绑定设备 Handle

int VzNL_AttachHandleForCloudPointAP(VZNLPOINTCLOUDHANDLE hCloudPointTool, VZNLHANDLE hDevice);

当配置自适应参数时,绑定设备 Handle

int VzNL_DetachHandleForCloudPointAP(VZNLPOINTCLOUDHANDLE hCloudPointTool);

3.6.7 设置 3D 点云基准激光线

(当生成灰度/深度图时需要配置), 配置后内部的图像参数会进行改变,若查看参数信息请调用 VzNL_GetCloudPointAPParam

入参: hCloudPointTool 点云工具句柄

入参: p3DPosition 3D 数据

入参: nPointCount 3D 点的个数

int VzNL_SetCloudPointAPBaseLine(VZNLPOINTCLOUDHANDLE hCloudPointTool, SVzNL3DPosition* p3DPosition, unsigned int nPointCount);

3.6.8 设置深度图/灰度图配置参数(当生成灰度/深度图时需要配置)

入参: hCloudPointTool 点云工具句柄

入参: psDepthMapParam 深度图配置参数

int VzNL_SetCloudPointAPParam(VZNLPOINTCLOUDHANDLE hCloudPointTool,
SVzDepthMapImgCtlPara* psDepthMapParam);

3.6.9 启用基准线深度值计算模式(当生成灰度/深度图时需要配置)

入参: hCloudPointTool 点云工具句柄

入参: bEnable VzTrue 启用 / VzFalse 禁用

注意: 基准线模式:bEnable = VzTrue 当深度计算需要用当前一条线作为高度基准时,配置为此模式,例如激光线照射 U 型皮带，会形成高低不等的一条激光线，这个时候 U 型皮带上的各个点均为自身位置的 0 点位置

int VzNL_EnableCloudPointAPBaseLineMode(VZNLPOINTCLOUDHANDLE hCloudPointTool, VzBool bEnable);

VzBool VzNL_IsEnableCloudPointAPBaseLineMode(VZNLPOINTCLOUDHANDLE hCloudPointTool);

3.6.10 设置输出图像的最小宽度(当生成灰度 / 深度图时需要配置)

入参: hCloudPointTool 点云工具句柄

入参: nMinImageWidth 输出图像的最小宽度

int VzNL_SetCloudPointAPImageWidth(VZNLPOINTCLOUDHANDLE hCloudPointTool, unsigned int nMinImageWidth);

3.6.11 获取点云配置

入参: hCloudPointTool 点云工具句柄

出参: psDepthMapParam [out] 点云配置

int VzNL_GetCloudPointAPParam(VZNLPOINTCLOUDHANDLE hCloudPointTool,
SVzDepthMapImgCtlPara* psDepthMapParam);

3.6.12 设置输出图像的最小宽度(当生成灰度 / 深度图时需要配置)

入参: hCloudPointTool 点云工具句柄

出参: pnImageHeight 输出图像高度

int VzNL_GetCloudPointAPImageHeight(VZNLPOINTCLOUDHANDLE hCloudPointTool, int* pnImageHeight);

3.6.13 开始点云处理(目前仅支持 keResultDataType_Position)

入参: hCloudPointTool 点云工具句柄

入参: eAfterProcType 图像后处理类型

* kePointAfterProcType_DepthMap 深度图

* kePointAfterProcType_GrayPic(灰度图)

入参: eDataType 激光线数据 SVzLaserLineData 的数据类型

入参: eDirect 激光线扫描方向

入参: pDepthMap 深度图的回调

入参: pParam 回调函数的参数

返回:返回 0 则为正确，返回非 0 为错误，可通过 VzNL_GetErrorInfo 获取错误信息

int VzNL_BeginCloudPointAfterProc(VZNLPOINTCLOUDHANDLE hCloudPointTool, EVzPointAfterProcType eAfterProcType, EVzResultDataType eDataType, EVzObjRunDirect eDirect, VzNL_OnOutputCloudPointAPData pDepthMap, void* pParam);

3.6.14 点云处理(目前仅支持 keResultDataType_Position)

入参: hCloudPointTool 点云工具句柄

入参: psLineData 激光线数据

int VzNL_CloudPointAfterProc(VZNLPOINTCLOUDHANDLE hCloudPointTool, SVzLaserLineData* psLineData);

3.6.15 结束点云处理

入参: hCloudPointTool 点云工具句柄

int VzNL_EndCloudPointAfterProc(VZNLPOINTCLOUDHANDLE hCloudPointTool);

3.6.16 销毁点云处理工具

入参: hCloudPointTool 点云工具句柄

int VzNL_DestroyCloudPointAPTool(VZNLPOINTCLOUDHANDLE hCloudPointTool);

3.6.17 创建图像生成工具

入参: hDevice 设备句柄

VZNLIMAGEGENERATOR VzNL_CreateImageGenerator(VZNLHANDLE hDevice);

3.6.18 开始送入点云数据

入参:hImageGenerator 图像生成工具

入参:eDataType 送入数据的类型,目前仅支持 keResultDataType_PointXYZRGBA 与 keResultDataType_PointGray

* keResultDataType_PointGray 输出图像为 8 bit 灰度图

* keResultDataType_PointGray 输出图像为 24 bit 彩色图

int VzNL_BeginPushPointToImageGenerator(VZNLIMAGEGENERATOR hImageGenerator, EVzResultDataType eDataType);

3.6.19 送入点云数据进行处理

入参: hImageGenerator 图像生成工具

入参: psLineData 激光线数据

int VzNL_PushPointToImageGenerator(VZNLIMAGEGENERATOR hImageGenerator, SVzLaserLineData* psLineData);

3.6.20 结束点云传送

入参: hImageGenerator 图像生成工具

int VzNL_EndPushPointToImageGenerator(VZNLIMAGEGENERATOR hImageGenerator);

3.6.21 通过图像生成工具，生成灰度图像

入参: hImageGenerator 图像生成工具

入参: psLineData 图像数据

int VzNL_GetImageFromImageGenerator(VZNLIMAGEGENERATOR hImageGenerator, SVzNLImageData** ppImageData);

3.6.22 通过图像生成工具，生成 Tif 图像

入参: hImageGenerator 图像生成工具

入参: lpszFile 文件名 tif

int VzNL_SaveDepthMapImageFromImageGenerator(VZNLIMAGEGENERATOR hImageGenerator, const char* lpszFile);

3.6.23 通过 2D 位置找 3D 点

入参: hImageGenerator 图像生成工具

入参: psLineData 3D 数据

int VzNL_Find3DPointFrom2DPosForImageGenerator(VZNLIMAGEGENERATOR hImageGenerator, int nX, int nY, SVzNLPointXYZ* p3DPoint);

3.6.24 销毁图像创建工具

入参: hImageGenerator 图像生成工具

int VzNL_DestroyImageGenerator(VZNLIMAGEGENERATOR hImageGenerator);

3.6.25 使用固定点云数据生成深度图

入参: eDataType 数据类型

入参: p3DPoint 3D 点云数组

入参: nPointCount 点的个数

入参: sDepthMapParam 深度图参数

入参: ppImageData 图像

返回: 返回 0 表示正常

int VzNL_CreateDepthMapImage(EVzResultDataType eDataType, void* p3DPoint, unsigned int nPointCount, SVzDepthMapImgCtlPara* sDepthMapParam, SVzNLImageData** ppImageData);

3.6.26 使用固定点云生成灰度图

入参: eDataType 数据类型

入参: p3DPoint 3D 点云数组

入参: nPointCount 点的个数

入参: fXYScale XY 比例

入参: ppImageData 图像

返回: 返回 0 表示正常

int VzNL_CreateGrayImage(EVzResultDataType eDataType, void* p3DPoint, unsigned int nPointCount, float fXYScale, SVzNLImageData** ppImageData);

3.6.27 读取图像

入参: szFileName [in] 要读取的文件全路径

入参: ppImageData [out] 无压缩图像实例

返回: 返回 0 表明正确

int VzNL_LoadImageFromFile(const char* szFileName, SVzNLImageData** ppImageData);

3.6.28 从 tif 文件中加载 3D 点云

入参: lpszFile 文件全路径

入参: nImageWidth 图像宽

入参: nImageHeight 图像高

入参: eDataType 数据类型

入参: p3DPoint 3D 点指针

返回: 返回 0 表示正常

int VzNL_LoadDepthMapTiffImage(const char* lpszFile, unsigned int* nImageWidth, unsigned int* nImageHeight, SVzNL3DPointF* p3DPoint, unsigned int* nPointCount);

3.6.29 将 3D 点云数据存储为 Tif 深度图

入参: lpszFile 文件全路径

入参: nImageWidth 图像宽

入参: nImageHeight 图像高

入参: eDataType 数据类型

入参: p3DPoint 3D 点指针

返回: 返回 0 表示正常

int VzNL_SaveDepthMapTiffImage(const char* lpszFile, unsigned int nImageWidth, unsigned int nImageHeight, EVzResultDataType eDataType, void* p3DPoint);

```
#include <stdio.h>
#include <vector>
#include <list>
#include "VZNL_Common.h"
#include "VZNL_EyeConfig.h"
#include "VZNL_DetectLaser.h"
#include "VZNL_Graphics.h"
#include "VZNL_DetectConfig.h"
#include "VZNL_Utils.h"
#include "VZNL_SwingMotor.h"
#include "windows.h"
#include <fstream>

/*****************************************************************************
 * File: vizumsdk_save_gray_depth_map_sample.cpp
 * Detail: 静态相机 保存灰度图像 和 深度图的 Sample
 * Author: Vizum
 * Date: 20210609
```

```
 ********************************************************************************/

static SVzVideoResolution g_sVideoRes;
static SVzNLPointXYZRGBA* g_p2DToPointMap = nullptr;
static bool* g_pb2DInvalidPt = nullptr;
static HANDLE hPauseEvent = NULL;
static SVzNLROIRect g_sImageLeftROI = { 0, 1536, 0, 2048 };
static SVzNLROIRect g_sImageRightROI = { 0, 1536, 0, 2048 };

/// @name  自动检测回调函数
/// @{
void _AutoOutputLaserLineExCB(EVzResultDataType eDataType, SVzLaserLineData* pLaserLinePoint, void*
pParam)
{
	//  当需要的点云格式为 SVzNLPointXYZRGBA 时
	if (keResultDataType_PointXYZRGBA == eDataType)
	{
		SVzNLPointXYZRGBA* p3DPoint = (SVzNLPointXYZRGBA*)pLaserLinePoint->p3DPoint;
		SVzNL2DLRPoint* p2DPoint = (SVzNL2DLRPoint*)pLaserLinePoint->p2DPoint;
		for (int nPtIdx = 0; nPtIdx < pLaserLinePoint->nPointCount; nPtIdx++)
		{
			if ((p2DPoint->sLeft.x < g_sImageLeftROI.left) || (p2DPoint->sLeft.x >=
g_sImageLeftROI.right))
			{
				p3DPoint++;
				p2DPoint++;
				continue;
			}

			if ((p2DPoint->sLeft.y < g_sImageLeftROI.top) || (p2DPoint->sLeft.y >=
g_sImageLeftROI.bottom))
			{
				p3DPoint++;
				p2DPoint++;
				continue;
			}

			int nPos = (p2DPoint->sLeft.x - g_sImageLeftROI.left) + (p2DPoint->sRight.y -
g_sImageLeftROI.top) * (g_sImageLeftROI.right - g_sImageLeftROI.left);

			g_p2DToPointMap[nPos] = *p3DPoint;
			g_pb2DInvalidPt[nPos] = true;
			p3DPoint++;
			p2DPoint++;
```

```
            }
        }
}

/// @brief
///  获取文件纯路径
int _GetPurePath(const char* szFilePath, char szPurePath[VZ_MAXPATH])
{
        int nFindPos = -1;
        const char* pPos = szFilePath;
        while ('\0' != *pPos)
        {
            if ('/' == *pPos || '\\' == *pPos)
            {
                nFindPos = static_cast<int>(pPos - szFilePath);
            }
            pPos++;
        }

        if (-1 == nFindPos)
            return -1;

        memcpy(szPurePath, szFilePath, static_cast<unsigned int>(nFindPos));
        szPurePath[nFindPos] = '\0';
        return 0;
}

//  显示错误信息
void _DisplayErrorInfo(const char* pszDesc, int nErrorCode)
{
        char szOutput[256];
        if (0 == nErrorCode)
        {
            sprintf_s(szOutput, 256, "%s\r\n", pszDesc);
        }
        else
        {
            char szErrorInfo[256] = { 0 };
            VzNL_GetErrorInfo(nErrorCode, szErrorInfo);

            sprintf_s(szOutput, 256, "%s [%d] %s\r\n", pszDesc, nErrorCode, szErrorInfo);
        }
        printf(szOutput);
}
```

```
// 搜索设备
void _ResearchDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
	// 搜索相机
	VzNL_ResearchDevice(keSearchDeviceFlag_EthLaserRobotEye);

	// 遍历相机
	int nDevCount = 0;
	VzNL_GetEyeCBDeviceInfo(nullptr, &nDevCount);
	if (nDevCount <= 0)
	{
		return;
	}

	vetDevice.resize(nDevCount);
	VzNL_GetEyeCBDeviceInfo(vetDevice.data(), &nDevCount);
}

// @brief 搜索且进行绑定设备
void _SearchAndBindDevice(std::vector<SVzNLEyeCBInfo>& vetDevice)
{
	int nErrorCode = 0;
	VzBool bCanResearch = VzFalse;
	do
	{
		bCanResearch = VzFalse;
		_ResearchDevice(vetDevice);
		std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
		while (itDevice != vetDevice.end())
		{
			// 相机没有被识别
			if (VzFalse == itDevice->bValidDevice)
			{
				SVzNLEyeCBInfo sCurDevInfo = *itDevice;
				SVzNLEthernetEyeConfigInfo sOutputEthConfigInfo;
				nErrorCode = VzNL_GetEthernetEyeConfigInfo(&sCurDevInfo, &sOutputEthConfigInfo);
				if (0 != nErrorCode)
				{
					_DisplayErrorInfo("获取网络信息失败", nErrorCode);
				}

				//
				if (0 == memcmp(sOutputEthConfigInfo.byLocalIP,
```

```
sOutputEthConfigInfo.sNetCardInfo.byLocalIP, 4))
                        {
                            _DisplayErrorInfo("请查看是否有网络权限和管理员权限\r\n", 0);
                        }
                        else
                        {
                            nErrorCode = VzNL_BindEthernetEye(&sCurDevInfo);
                            if (0 != nErrorCode)
                            {
                                _DisplayErrorInfo("绑定相机失败", nErrorCode);
                            }
                            bCanResearch = VzTrue;
                        }
                    }
                    itDevice++;
                }
        } while (bCanResearch);
}

/**
 * @brief 设备状态改变 CallBack 函数
 * @param eStatus [out] 状态
 * @param pInfoParam [out] 状态参数
 */
void _OnNotifyStatusCB(EVzDeviceWorkStatus eStatus, void* pInfoParam)
{
    if (keDeviceWorkStatus_Device_Auto_Stop == eStatus)
    {
        printf("收到停止信号\r\n");
        if (nullptr != hPauseEvent)
        {
            SetEvent(hPauseEvent);
        }
    }
}

int main()
{
    hPauseEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    // 初始化 SDK
    SVzNLConfigParam sConfigParam;
    sConfigParam.nDeviceTimeOut = 0;
    int nRet = VzNL_Init(&sConfigParam);
```

```
	if (0 != nRet)
	{
		_DisplayErrorInfo("查看是否有其他程序在使用 SDK?", nRet);
	}

	VzNL_SetLogLevel(keNLLogLevel_Information, keNLLogType_File);

	// 枚举相机
	std::vector<SVzNLEyeCBInfo> vetDevice;
	_SearchAndBindDevice(vetDevice);

	//
	int nErrorCode = 0;
	std::vector<VZNLHANDLE>		vetHandles;
	if (vetDevice.size() > 0)
	{
		_DisplayErrorInfo("显示当前可用相机\r\n", 0);
		std::vector<SVzNLEyeCBInfo>::iterator itDevice = vetDevice.begin();
		while (itDevice != vetDevice.end())
		{
			// 相机没有被识别
			if (VzFalse == itDevice->bValidDevice)
			{
				itDevice++;
			}

			char szOutput[256];
			sprintf_s(szOutput, 256, "----------------------准备打开 %s----------------------\r\n",
itDevice->byServerIP);
			_DisplayErrorInfo(szOutput, 0);

			// 打开相机
			SVzNLOpenDeviceParam sOpenDevParam;
			VZNLHANDLE hDevice = VzNL_OpenDevice(&(*itDevice), &sOpenDevParam,
&nErrorCode);
			if (0 != nErrorCode)
			{
				_DisplayErrorInfo("打开设备失败", nErrorCode);
				break;
			}
			else
			{
				char szOutput[256];
				sprintf_s(szOutput, 256, "打开相机%s 成功\r\n", itDevice->byServerIP);
```

```
        _DisplayErrorInfo(szOutput, 0);
}

if (NULL != hDevice)
{
        vetHandles.push_back(hDevice);
}

VzNL_SetOutputROIImage(VzTrue);

VzNL_SetOutputImageFormat(keVzNLImageType_BGR888);

VzNL_SetDeviceStatusNotify(hDevice, _OnNotifyStatusCB, NULL);

// 创建激光线检测工具
nErrorCode = VzNL_BeginDetectLaser(hDevice);
if (0 != nErrorCode)
{
        _DisplayErrorInfo("创建检测激光线工具", nErrorCode);
        break;
}
else
{
        _DisplayErrorInfo("创建激光线检测工具成功\r\n", 0);
}

VzNL_EnableCalibROI(hDevice, VzFalse);

unsigned int nOldExpose = 0;
EVzNLExposeMode eExposeMode = keVzNLExposeMode_Fix;
VzNL_GetConfigEyeExpose(hDevice, &eExposeMode, &nOldExpose);

SVzNLROIRect sOldLeftROI, sOldRightROI;
VzNL_GetConfigDetectROI(hDevice, &sOldLeftROI, &sOldRightROI);

VzNL_EnableCalibROI(hDevice, VzTrue);

SVzNLEyeDeviceInfoEx sDeviceInfoEx;
sDeviceInfoEx.sEyeCBInfo.nSize = sizeof(SVzNLEyeDeviceInfoEx);
VzNL_GetDeviceInfo(hDevice, &sDeviceInfoEx.sEyeCBInfo);
g_sVideoRes = sDeviceInfoEx.sVideoRes;

g_sImageLeftROI.right = g_sVideoRes.nFrameWidth;
g_sImageLeftROI.bottom = g_sVideoRes.nFrameHeight;
```

```
        g_sImageRightROI.right = g_sVideoRes.nFrameWidth;
        g_sImageRightROI.bottom = g_sVideoRes.nFrameHeight;

        int nFrameSize = (g_sImageRightROI.right - g_sImageRightROI.left) *
(g_sImageRightROI.bottom - g_sImageRightROI.top);
        g_p2DToPointMap = new SVzNLPointXYZRGBA[nFrameSize];
        g_pb2DInvalidPt = new bool[nFrameSize];
        memset(g_pb2DInvalidPt, 0, sizeof(bool) * nFrameSize);

        // 配置曝光
        nErrorCode = VzNL_ConfigEyeExpose(hDevice, keVzNLExposeMode_Fix, 10000);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("曝光配置失败", nErrorCode);
        }

        // 配置 ROi
        nErrorCode = VzNL_ConfigDetectROI(hDevice, &g_sImageLeftROI, &g_sImageRightROI);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("ROI 检测配置失败", nErrorCode);
        }

        SVzNLImageData* pLeftImage = nullptr; SVzNLImageData* pRightImage = nullptr;
        nErrorCode = VzNL_GetEyeImage(hDevice, &pLeftImage, &pRightImage, 5000);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("获取图像失败", nErrorCode);
        }

        VzNL_EnableCalibROI(hDevice, VzFalse);

        nErrorCode = VzNL_ConfigEyeExpose(hDevice, keVzNLExposeMode_Fix, nOldExpose);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("曝光配置失败", nErrorCode);
        }

        // 配置 ROi
        nErrorCode = VzNL_ConfigDetectROI(hDevice, &sOldLeftROI, &sOldRightROI);
        if (0 != nErrorCode)
        {
            _DisplayErrorInfo("ROI 检测配置失败", nErrorCode);
        }
```

```
                // 设置到主模式
                nErrorCode = VzNL_SetTriggerMode(hDevice, keEyeTriggerMode_Master);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("设置主模式失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("主模式设置成功\r\n", 0);
                }

                unsigned int nMinFrameRate, nMaxFrameRate;
                VzNL_QueryParamRange(hDevice, keDeviceParamType_FrameRate, &nMinFrameRate,
&nMaxFrameRate);

                VzNL_SetFrameRate(hDevice, nMaxFrameRate);

                ResetEvent(hPauseEvent);

                VzBool bIsSupportMotor = VzNL_IsSupportSwingMotor(hDevice, nullptr);
                if (bIsSupportMotor)
                {
                    VzNL_EnableSwingMotor(hDevice, VzTrue);
                }

                // 开始流模式检测
                nErrorCode = VzNL_StartAutoDetectEx(hDevice, keResultDataType_PointXYZRGBA,
keFlipType_None, _AutoOutputLaserLineExCB, nullptr);
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("开流失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("开流成功\r\n", 0);
                }

                // 等待摆动停止
                WaitForSingleObject(hPauseEvent, INFINITE);

                nErrorCode = VzNL_StopAutoDetect(hDevice);
```

```
                if (0 != nErrorCode)
                {
                    _DisplayErrorInfo("关流失败", nErrorCode);
                    break;
                }
                else
                {
                    _DisplayErrorInfo("关流成功\r\n", 0);
                }
                VzNL_EndDetectLaser(hDevice);
                VzNL_CloseDevice(hDevice);
                // 存储灰度图
                char szModule[VZ_MAXPATH] = { 0 }; char szPurePath[VZ_MAXPATH] = { 0 };
                GetModuleFileNameA(nullptr, szModule, VZ_MAXPATH);
                _GetPurePath(szModule, szPurePath);
                if (nullptr != pLeftImage)
                {
                    char szGrayFile[VZ_MAXPATH] = { 0 };
                    sprintf_s(szGrayFile, VZ_MAXPATH, "%s\\Gray.png", szPurePath);
                    VzNL_SaveImage(szGrayFile, pLeftImage);
                }

                // 存储深度图
                if (nullptr != g_p2DToPointMap)
                {
                    char szTifFile[VZ_MAXPATH] = { 0 };
                    sprintf_s(szTifFile, VZ_MAXPATH, "%s\\Depth.tif", szPurePath);
                    VzNL_SaveDepthMapTiffImage(szTifFile, g_sVideoRes.nFrameWidth,
g_sVideoRes.nFrameHeight, keResultDataType_PointXYZRGBA, g_p2DToPointMap);
                }

                // 输出点云映射数据
                if (nullptr != g_p2DToPointMap)
                {
                    delete[] g_p2DToPointMap;
                    g_p2DToPointMap = nullptr;
                }
                if (nullptr != g_pb2DInvalidPt)
                {
                    delete[] g_pb2DInvalidPt;
                    g_pb2DInvalidPt = nullptr;
                }
                VzNL_ReleaseImage(&pLeftImage);
                VzNL_ReleaseImage(&pRightImage);
```

```
                itDevice++;
            }
        }
        // 销毁 SDK
        VzNL_Destroy();
        getchar();
        return 0;
}
```

3.7 VzNLSDK 图像操作接口

头文件：#include "VZNL_Graphics.h"

库文件：VzNL_Graphics.dll / lib VzNL_Graphics.so

3.7.1 开始绘图

VZNLHANDLE VzNL_BeginPaint(SVzNLImageData* pImageData);

入参：　pImageData　　图像结构体地址

返回：　绘图句柄，NULL 为创建失败

3.7.2 绘制直线

int VzNL_DrawLine(VZNLHANDLE hHandle, const SVzNL2DPoint sPoint[2]);

入参：　hDevice　　设备句柄

　　dwFrameColor　　边框颜色

　　dwFillColor　　填充色

　　nBorder　　边框宽度

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.3 绘制矩形

int VzNL_DrawRect(VZNLHANDLE hHandle, const SVzNL2DPoint sRect[4], unsigned long dwFrameColor, unsigned long dwFillColor, unsigned int nBorder);

入参：　hDevice　　设备句柄

　　sRect　　矩形四个点的坐标

　　dwFrameColor　　边框颜色

　　dwFillColor　　填充色

　　nBorder　　边框宽度

返回：　0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.4 绘制多边形

int VzNL_DrawPolygon(VZNLHANDLE hHandle, const SVzNL2DPoint* psPoint, const unsigned int nPtCount, unsigned long dwFrameColor, unsigned long dwFillColor, unsigned int nBorder);

入参：　hDevice　　设备句柄

　　psPoint　　多变形顶点坐标数组

	nPtCount	顶点个数
	dwFrameColor	边框颜色
	dwFillColor	填充色
	nBorder	边框宽度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.5 绘制圆形

int VzNL_DrawCircle(VZNLHANDLE hHandle, const SVzNL2DPoint sPoint, int nRadius, unsigned long dwFrameColor, unsigned long dwFillColor, unsigned int nBorder);

入参：	hDevice	设备句柄
	sPoint	圆心坐标
	nRadius	边框颜色
	dwFrameColor	边框颜色
	dwFillColor	填充色
	nBorder	边框宽度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.6 绘制椭圆

int VzNL_DrawEllipe(VZNLHANDLE hHandle, const SVzNL2DPoint sPoint, unsigned long dwFrameColor, unsigned long dwFillColor, unsigned int nLongAxes, unsigned int nShortAxes, double angle, unsigned int nBorder);

入参：	hDevice	设备句柄
	sPoint	圆心坐标
	dwFrameColor	边框颜色
	dwFillColor	填充色
	nLongAxes	长轴长度
	nShortAxes	短轴长度
	angle	旋转角度
	nBorder	边框宽度

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.7 结束绘图

void VzNL_EndPaint(VZNLHANDLE hHandle);

入参：	hDevice	设备句柄

返回： 无

3.7.8 显示图像

void VzNL_ShowImage(const char* szWindowName, SVzNLImageData* pImageData);

入参：	szWindowName	窗口名称
	pImageData	图像数据

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

注：该接口在 linux 下无法显示图像。（linux 下显示库版本/种类多，无法做到通用）

3.7.9 存储图像

int VzNL_SaveImage(const char* szFileName, SVzNLImageData* pImageData);

入参： szFileName 存储文件的名称

pImageData 图像数据

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.10 压缩图片

int VzNL_CompressImage(EVzNLImageCompress eCompress, SVzNLImageData* pSrc, VzNLCompressData** pDst);

入参： handle 绘图工具 handle

eCompress 压缩类型

pSrc 数据源（无压缩）

pDst 压缩后的数据

返回： 0 为正确，其他错误码可通过 VzNL_GetErrorInfo 获取错误信息。

3.7.11 释放压缩图片

void VzNL_ReleaseCompressImage(SVzNLCompressData** pDst);

入参： pDst 所要释放图像的地址

返回： 无返回

3.8 文件相关接口

头文件：#include "VZNL_FileUtils.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.8.1 创建文件工具

入参: eFileType 文件类型

* keLaserFileType_PureTxt 纯文本
* keLaserFileType_Txt 本公司自用
* keLaserFileType_Pcd PCD
* keLaserFileType_Las LAS
* keLaserFileType_VzBinaryData Binary File

返回: 返回文件句柄

VZNLFILE VzNL_CreateLaserFile(EVzLaserFileType eFileType);

3.8.2 打开文件

入参: hFile 文件句柄

入参: lpszFile 文件名

入参: eFileMode 文件打开模式

入参: eDataType 数据类型

入参: dSpeed 速度 mm/s

返回: 返回值为 0 表示正确

int VzNL_OpenLaserFile(VZNLFILE hFile, const char* lpszFile, EVzFileMode eFileMode, EVzResultDataType

eDataType, double dSpeed);

3.8.3 写入文件

入参: hFile 文件句柄

入参: psLaserLinePoint 激光线数据

返回:返回值为 0 表示正确

int VzNL_WriteLaserFile(VZNLFILE hFile, const SVzLaserLineData* psLaserLinePoint);

3.8.4 关闭文件

入参: hFile [in] 文件句柄

返回: 返回值为 0 表示正确

int VzNL_CloseLaserFile(VZNLFILE hFile);

```
static VZNLFILE s_hFile = nullptr;

// 数据回调函数
void _AutoOutputLaserLineExCB(EVzResultDataType eDataType, SVzLaserLineData*
pLaserLinePoint, void* pParam)
{
    // 当需要的点云格式为SVzNLPointXYZRGBA时
    if (keResultDataType_PointXYZRGBA == eDataType)
    {
        if (nullptr != s_hFile)
        {
            VzNL_WriteLaserFile(s_hFile, pLaserLinePoint);
        }
    }
}

// 创建文件句柄
void _SaveFile()
{
         // 创建激光线检测工具
        int nErrorCode = VzNL_BeginDetectLaser(hDevice);
        if (0 != nErrorCode)
        {
            return;
        }

         s_hFile = VzNL_CreateLaserFile(keLaserFileType_Pcd);
         if (NULL != s_hFile)
         {
               VzNL_OpenLaserFile(s_hFile, "D:/test.pcd", keFileModeWrite,
keResultDataType_PointXYZRGBA, 0.);
         }
```

```
        // 开始流模式检测
         nErrorCode = VzNL_StartAutoDetectEx(hDevice, keResultDataType_PointXYZRGBA,
keFlipType_None, _AutoOutputLaserLineExCB, nullptr);
         if (0 != nErrorCode)
         {
             break;
         }

         // 保存5s的数据
         Sleep(5000);

         nErrorCode = VzNL_StopAutoDetect(hDevice);
         if (0 != nErrorCode)
         {
             break;
         }

          if (NULL != s_hFile)
          {
                VzNL_CloseLaserFile(s_hFile);
          }

          // 销毁激光线检测工具
          VzNL_EndDetectLaser(hDevice);
}
```

3.9 RGB 相关配置接口

头文件：#include "VZNL_RGBConfig.h"

库文件：VzNLDetect.dll / libVzNLDetect.so

3.9.1 当前相机是否支持 RGB

入参: hDevice 当前设备句柄

入参: pnErrorCode 错误信息，如果不需要可填 NULL

返回: VzTrue 表示支持 VzFalse 表示不支持

VzBool VzNL_IsSupportRGBCamera(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.2 获取 RGB 图像分辨率

入参: hDevice [in] 设备 Handle

返回: 关闭成功返回 0，否则为错误码

int VzNL_GetRGBResolution(VZNLHANDLE hDevice, SVzVideoResolution* psVideoRes);

3.9.3 启用/获取 RGB 功能

入参: hDevice 当前设备句柄

入参: bEnable VzTrue 启用 / VzFalse 禁用

返回: 正确返回 0， 失败返回其他值

int VzNL_EnableRGB(VZNLHANDLE hDevice, VzBool bEnable);

VzBool VzNL_IsEnableRGB(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.4 格式化彩色 ROI

入参: hDevice 当前设备句柄

入参: psROI[in/out] 想要格式化的 ROI 大小

返回: 正确返回 0， 失败返回其他值

int VzNL_FormatRGBROI(VZNLHANDLE hDevice, SVzNLROIRect* psROI);

3.9.5 配置 RGB ROI

入参: hDevice 当前设备句柄

入参: psROI 想要设置的 ROI 大小

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBROI(VZNLHANDLE hDevice, const SVzNLROIRect* psROI);

VZNLAPI int VzNL_GetRGBROI(VZNLHANDLE hDevice, SVzNLROIRect* psROI);

3.9.6 启用/获取 自动白平衡

入参: hDevice 当前设备句柄

入参: bEnable 启用 VzTrue /禁用 VzFalse 白平衡

返回: 正确返回 0， 失败返回其他值

int VzNL_EnableRGBAWB(VZNLHANDLE hDevice, VzBool bEnable);

VzBool VzNL_IsEnableRGBAWB(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.7 设置/获取 自动曝光状态

入参: hDevice 当前设备句柄

入参: bEnable 启用 VzTrue / 禁用 VzFalse

返回: 正确返回 0， 失败返回其他值

int VzNL_EnableRGBAutoExpose(VZNLHANDLE hDevice, VzBool bEnable);

VzBool VzNL_IsEnableRGBAutoExpose(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.8 设置 R Value

入参: hDevice 当前设备句柄

入参: fValue R 分量值

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBRValue(VZNLHANDLE hDevice, float fValue);

float VzNL_GetRGBRValue(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.9 设置 G Value

入参: hDevice 当前设备句柄

入参: fValue G 分量值

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBGValue(VZNLHANDLE hDevice, float fValue);

float VzNL_GetRGBGValue(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.10 设置 B Value

入参: hDevice 当前设备句柄

入参: fValue B 分量值

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBBValue(VZNLHANDLE hDevice, float fValue);

float VzNL_GetRGBBValue(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.11 设置彩色 Sensor 帧率

入参: hDevice 当前设备句柄

入参: nValue 帧率

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBFrameRate(VZNLHANDLE hDevice, unsigned int nValue);

unsigned int VzNL_GetRGBFrameRate(VZNLHANDLE hDevice, int* pErrorCode);

3.9.12 设置彩色 Sensor 曝光

入参: hDevice 当前设备句柄

入参: nValue 曝光值[20~100000]

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBExpose(VZNLHANDLE hDevice, unsigned int nValue);

unsigned int VzNL_GetRGBExpose(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.13 设置彩色 sensor 增益

入参: hDevice 当前设备句柄

入参: nValue 设置增益值

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBGain(VZNLHANDLE hDevice, unsigned int nValue);

unsigned int VzNL_GetRGBGain(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.14 设置 RGBSensor 触发方式

入参: hDevice 当前设备句柄

入参: eTriggerMode 触发模式

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBTriggerMode(VZNLHANDLE hDevice, EVzEyeTriggerMode eTriggerMode);

EVzEyeTriggerMode VzNL_GetRGBTriggerMode(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.15 设置彩色 Sensor 期望值

入参: hDevice 当前设备句柄

入参: fExposeThres 期望值[1~255]

返回: 正确返回 0， 失败返回其他值

int VzNL_SetRGBAutoExposeThres(VZNLHANDLE hDevice, float fExposeThres);

float VzNL_GetRGBAutoExposeThres(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.16 生成一个 RGB 信号

入参:　hDevice 当前设备句柄

返回:　正确返回 0，　失败返回其他值

int VzNL_GenRGBSoftSignal(VZNLHANDLE hDevice);

3.9.17 设置自动调节超时时间

入参:　hDevice 当前设备句柄

入参:　nTimeOut 设置自动调节超时时间

返回:　正确返回 0，　失败返回其他值

int VzNL_SetRGBAutoAdjustTimeOut(VZNLHANDLE hDevice, unsigned int nTimeOut);

unsigned int VzNL_GetRGBAutoAdjustTimeOut(VZNLHANDLE hDevice, int* pnErrorCode);

3.9.18 设置 CCM

入参:　hDevice 当前设备句柄

入参:　dCCMVal CCM 值

返回:　正确返回 0，　失败返回其他值

int VzNL_SetRGBColorCortectMatrix(VZNLHANDLE hDevice, double dCCMVal[3][3]);

int VzNL_GetRGBColorCortectMatrix(VZNLHANDLE hDevice, double dCCMVal[3][3]);

3.9.19 启用是否输出 RGB 2D 数据

入参:　hDevice 当前设备句柄

入参:　bEnable [in] 启用为 VzTrue,关闭为 VzFalse

返回:　正确返回 0，　失败返回其他值

提示:　当配置为 VzTrue 时，开始检测输出的 2D 左目数据为 RGB 图像的位置。

int VzNL_EnableOutputColor2D(VZNLHANDLE hDevice, VzBool bEnable);

4 示例代码

由于 SDK 示例较多，在这里将不再占用篇幅，详见 SDK 的 Demo 文件