HPS-3D160 面阵相机 SDK 使用手册

Hypersen

HYPERSEN TECHNOLOGIES CO., LTD. 海伯森技术

目录

一、SDK 简介

SDK 提供了 HPS3D160 面阵相机的应用程序接口，目前可供 linux 平台,windows 平台,ROS 平台以及没有跑操作系统的大多数单片机上使用；SDK 使用前请详细阅读该使用手册；

二、将 SDK 集成到 IDE 中

为方便跨平台及跨语言使用 SDK，目前提供基础数据类型定义的 API 接口，参考 HPS3DBase_IF.h，目前对此 API 在 C/C++ 及 C#上均有二次封装，方便用户集成到项目中，其他语言如 Java、python 后续根据需求将进一步拓展；

2.1 C/C++ 调用 SDK 基本步骤

为方便用户集成到项目中，我们对基础的 API 进行再次封装处理，其中 HPS3DBase_IF.h 为基础 API 接口，HPS3DUser_IF.c 及 HPS3DUser_IF.h 为二次封装；基本步骤如下：

1、将 HPS3DBase_IF.h、HPS3DUser_IF.C、HPS3DUser_IF.h 已经相应平台下的动态库，拷贝到项目中并在工程中包含 HPS3DBase_IF.h、HPS3DUser_IF.C、HPS3DUser_IF.h 三个文件；
2、定义全局参数并初始化，示例代码:

```
int g_handle = -1;
static HPS3D_MeasureData_t g_measureData;

HPS3D_StatusTypeDef ret = HPS3D_RET_ERROR;
ret = HPS3D_MeasureDataInit(&g_measureData);
if (ret != HPS3D_RET_OK)
{
    printf("MeasureDataInit failed,Err:%d\n", ret);
}
```

注：此处为方便内存控制，采用动态内存分配方式，程序退出时需执行内存释放，调用 HPS3D_MeasureDataFree 接口；
3、根据设备型号选择连接方式。示例代码：

```
HPS3D_HandleTypeDef handle;
HPS3D_StatusTypeDef ret = HPS3D_RET_ERROR;
ret = HPS3D_USBConnectDevice((char *)"/dev/ttyACM0",&g_handle); //USB Connect
//ret = HPS3D_EthernetConnectDevice((char *)"192.168.0.10", 12345, &g_handle);
if (ret != HPS3D_RET_OK)
{
    printf("connect failed,Err:%d\n", ret);
}
```

注：linux 下连接 USB 设备时，需手动修改设备权限，执行 chmod 777 /dev/ttyACM0 即可；Windows 下连接 USB 设备时，输入端口号，为解决编码问题，建议在端口号前加上_T, 如果端口号在 COM10 以上，则需要在端口号前加上 “\\\\.\\ ”, 如“ \\\\.\\ COM15”;

```
ret = HPS3D_USBConnectDevice((char *)_T("COM3"), &g_handle);
ret = HPS3D_USBConnectDevice((char *)_T("\\\\.\\COM15"), &g_handle);
```

4、注册回调函数，用于事件通知。示例代码：

```
/*定义回调函数*/
void EventCallBackFunc(int handle, int eventType, uint8_t *data, int dataLen, void *userPara)
{
    switch ((HPS3D_EventType_t)eventType)
    {
        case HPS3D_SIMPLE_ROI_EVEN:
        case HPS3D_FULL_ROI_EVEN:
        case HPS3D_FULL_DEPTH_EVEN:
        case HPS3D_SIMPLE_DEPTH_EVEN:
             printf("接收测量数据!");  /*测量结果从这里获取*/
             HPS3D_ConvertToMeasureData(data,&g_measureData,eventType);
            break;
        case HPS3D_SYS_EXCEPTION_EVEN:
            printf("SYS ERR :%s\n", data);
            break;
        case HPS3D_DISCONNECT_EVEN:
            printf("Device disconnected!\n");
            HPS3D_CloseDevice(handle);
            break;
        case HPS3D_NULL_EVEN:
        default:
            break;
    }
}
/*注册回调函数*/
HPS3D_StatusTypeDef ret = HPS3D_RET_ERROR;
ret = HPS3D_RegisterEventCallback(EventCallBackFunc, NULL);
if (ret != HPS3D_RET_OK)
{
      printf("RegisterEventCallback failed,Err:%d\n", ret);
}
```

5、开始测量，示例代码：

```
/*连续模式*/
HPS3D_StartCapture(g_handle);

/*单次模式*/
HPS3D_EventType_t type = HPS3D_NULL_EVEN;
ret = HPS3D_SingleCapture(g_handle, &type, &g_measureData);
```

```
if (ret != HPS3D_RET_OK)
{
    printf("SingleCapture failed,Err:%d\n", ret);
}
```

注：

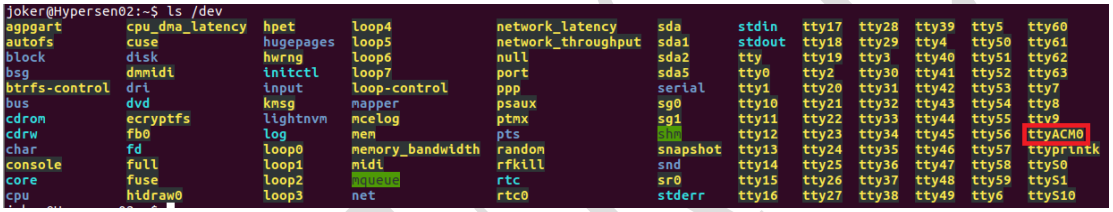
- 连续模式测量结果在回调函数中获取，回调函数用于数据通知，不建议在回调函数中执行较为耗时的操作，否则可能影响采集帧率；
- 调用 HPS3D_StartCapture 后仅执行 HPS3D_StopCapture 有效，其余接口均返回错误值；
- 单次模式测量结果立即返回，需检查返回值判断采集是否成功；

2.1.1 Linux 平台下调用 SDK

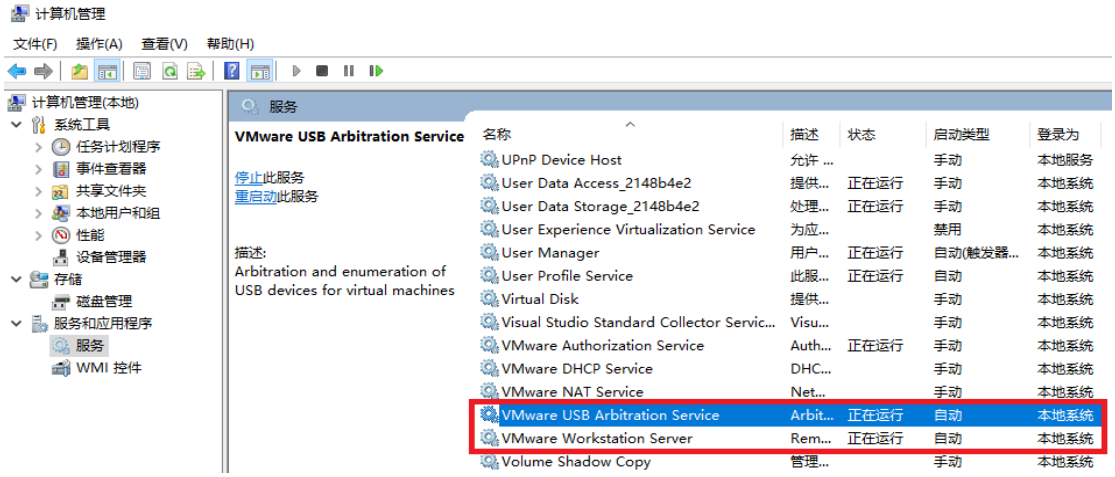
xxx.so 适合在 Linux 操作系统平台使用，这里以 Ubuntu 为例。本示例基于版本号为 V1.8.0 的 SDK 进行编写。

- 设备连接

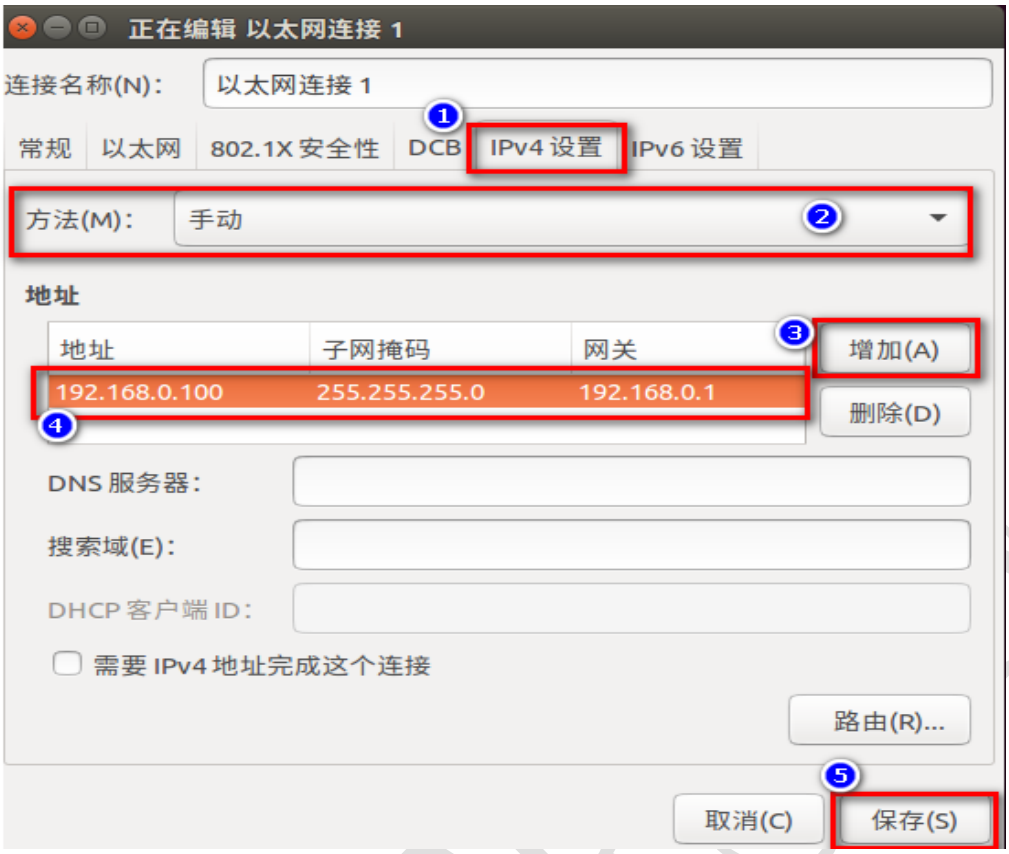
根据设备型号主要分为 HPS3D160-U/I 及 HPS3D160-L,其中 USB 设备连接步骤为：将设备连接到 Ubuntu，打开终端输入 ls /dev，查看设备 ttyACM*，如下图所示：



若没有查看到 ttyACM*设备名称，则需要重新拔插，再次查看，若还是没有，则到“计算机管理->服务和应用程序->服务”中查看 VMware USB Arbitration Service 是否正在运行，若禁用，则开启运行，之后重新拔插设备即可；若不想每次登陆虚拟机都启动一次 USB 设备服务，则可将 VMware Workstation Server 和 VMware USB Arbitration Service 均设置成运行和自动，重启计算机即可。



网口版本连接步骤为：将设备连接到 ububtu，并配置以太网连接 IPV4 地址，以传感器默认 IP 地址 192.168.0.10 为例，配置步骤如下图所示：



- 在 Ububtu 下使用 SDK,运行 Demo 程序

  在 ubuntu 下使用 HPS3D160_SDK 步骤如下：

  1、 将 HPS3D160-Linux-C Demo 程序拷贝至 ubuntu 任意目录下，打开终端输入 make 即可对 demo 程序进行编译，执行结果如下则编译通过；

```
kevin@kevin-virtual-machine:~/HPS3D160-SDK/demo/HPS3D160-Linux-C_Demo$ make
gcc -Wall -O -g -c HPS3DUser_IF.c -o HPS3DUser_IF.o
g++ HPS3DUser_IF.o main.o -o ./app -Wl,-rpath=./ -L./ -lHPS3DSDK
chmod a+x ./app
kevin@kevin-virtual-machine:~/HPS3D160-SDK/demo/HPS3D160-Linux-C_Demo$
```

  2、 在终端输入./app 即可连接设备并开始测量，连接成功后自动导出当前设备参数如下所示，根据提示输入 1 表示单次测量，输入 2 表示连续测量；

```
kevin@kevin-virtual-machine:~/HPS3D160-SDK/demo/HPS3D160-Linux-C_Demo$ ./app
HPS3D160 C/C++ Demo (Linux)

SDK Ver:V1.8.0 21-6-10
Dev Ver:V1.8.0 21-6-10
SN:SN:HL21M03173D1901663

resolution:160 X 60
max_roi_group_number:16  cur_group_id: 0
max_roi_number:8
max_multiCamera_code:15, cur_multiCamera_code:0
user_id: 0
optical_path_calibration: 0

select capture mode: SingleCapture(1)  ContinuousCapture(2)  Exit(...)
```

  3、 不同型号的设备需要修改连接方式，修改方法如下图所示

```
130             ret = HPS3D_USBConnectDevice((char *)"/dev/ttyACM0",&g_handle); //USB Connect
131
132             //ret = HPS3D_EthernetConnectDevice((char *)"192.168.0.10", 12345, &g_handle);
133             if (ret != HPS3D_RET_OK)
134             {
135                     printf("connect failed,Err:%d\n", ret);
136                     break;
137             }
138             printf("Dev Ver:%s\n", HPS3D_GetDeviceVersion(g_handle));
139             printf("SN:%s\n\n", HPS3D_GetSerialNumber(g_handle));
140
141             HPS3D_RegisterEventCallback(EventCallBackFunc, NULL);
142
```

注：USB 版本连接失败，原因可能是设备无权限，可执行命令 sudo chmod 777 /dev/ttyACM0 后重试；

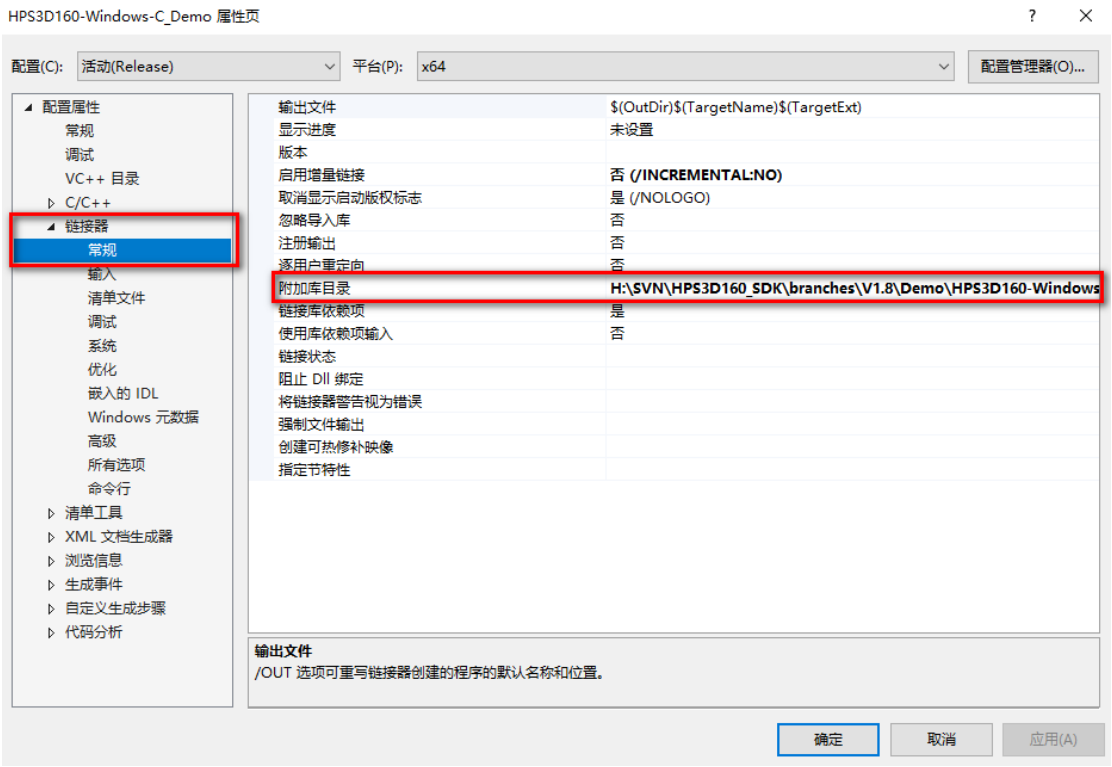
2.1.2 Windows 平台下调用 SDK

xxx.dll 适合在 Windows 操作系统平台使用，这里以 windows 下的 Microsoft Visual Studio 2017 为例。本示例基于版本号为 V1.8.0 的 SDK 进行编写。

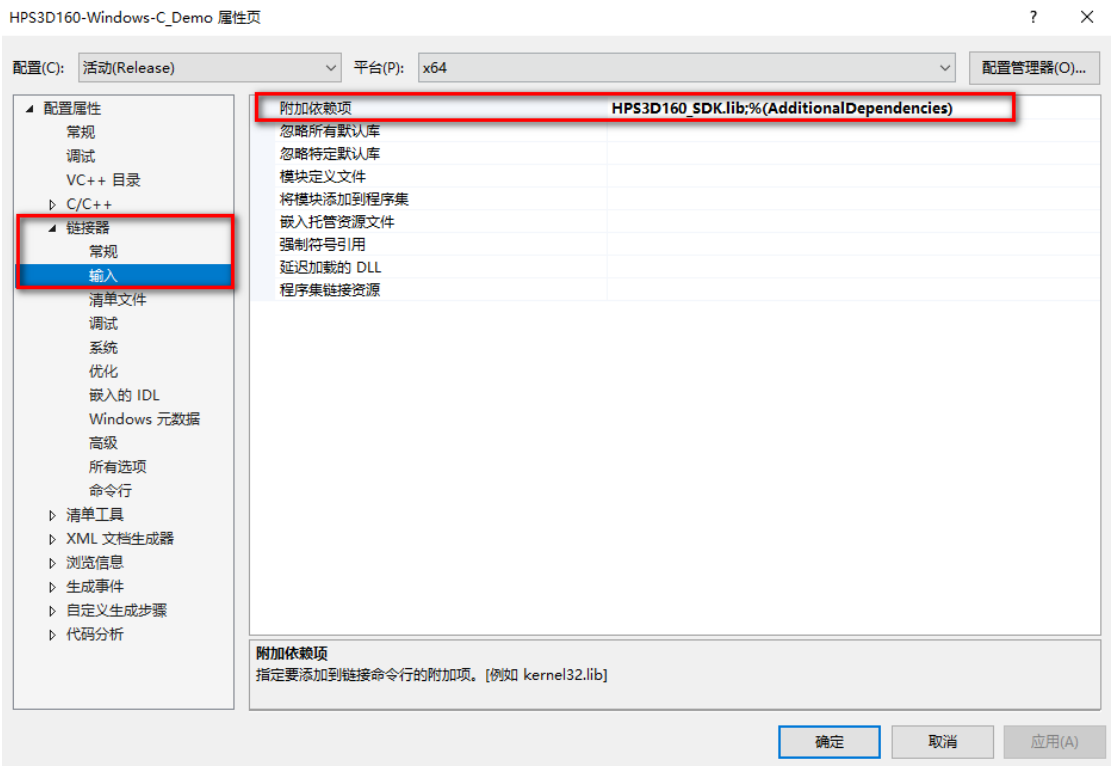
在 Visual Studio 2017 下使用 HPS3D160_SDK 步骤如下：

1、 打开 HPS3D160-Windows-C_Demo 工程，并检查工程配置如下：

右键项目 - 属性 - 链接器 - 常规。在附加库目录中指定 HPS3D160_SDK.dll 的路径。



右键项目 - 属性 - 链接器 - 输入。在附加依赖选项中填入 HPS3D160_SDK.lib。



拷贝 HPS3D160_SDK.dll 到程序运行目录下，即可正常运行 Demo 程序；

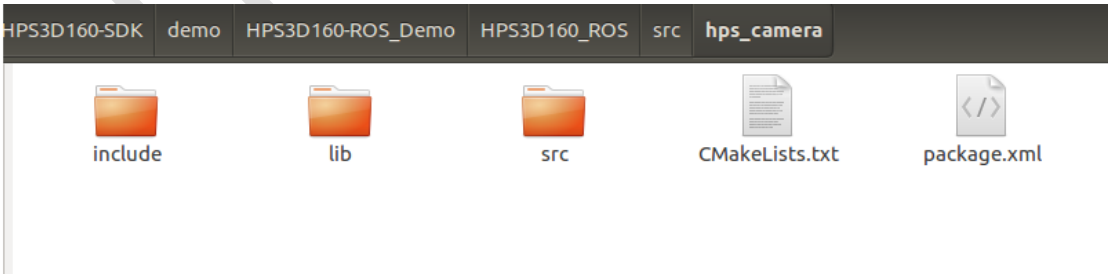
注：USB 设备连接时由于编码问题，可能会导致设备连接失败，可在端口号前加入_T 解决此问题，如果端口号在 COM10 以上，则需要在端口号前加入”\\\\.\\”，如 \\\\.\\COM10；

```
ret = HPS3D_USBConnectDevice((char *)_T("COM3"), &g_handle);
ret = HPS3D_USBConnectDevice((char *)_T("\\\\.\\COM10"), &g_handle);
```

2.1.3 ROS 平台下调用 SDK

这里以 Ubuntu 16.04.6 LTS 以及 kinetic 为例。本示例基于版本号为 V1.8.0 的 SDK 进行编写。由于 HPS3D160-ROS Demo 受限于当前 ROS 版本，此处不详描述工程创建方法；

参考 Demo 示例：进入源码目录 HPS3D160-ROS_Demo/HPS3D160_ROS/src/hps_camera



- 将 include 目录下的 HPS3DBase_IF、HPS3DUser_IF 拷贝至工程目录的 include 目录
- 将 lib 目录下的 libHPS3DSDK.so 拷贝到工程目录下的 lib 目录下，如果不存在 lib 则创建；
- 将 src 目录下的 HPS3DUser_IF.c 拷贝到工程目录的 src 目录下，并参考 hps_camera.cpp 将相关代码引用至工程中；

- 修改 CMakeLists.txt 文件内容，如下所示：

添加相关依赖库

```
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
  std_srvs
  sensor_msgs
  cv_bridge
  image_transport
  pcl_conversions
  pcl_ros
)
```

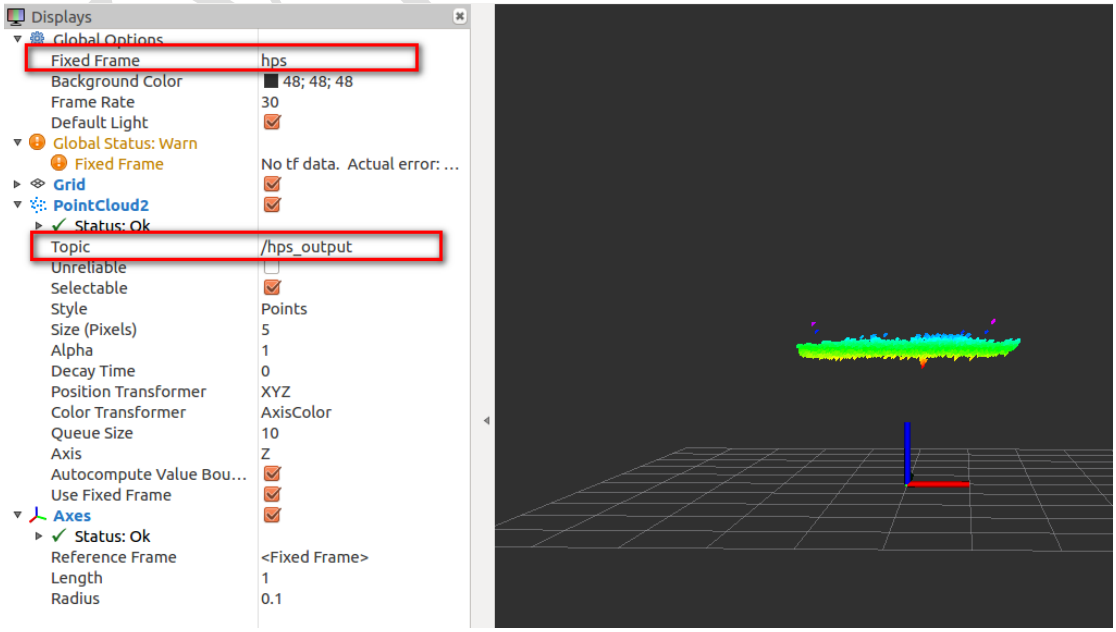
指定包含头文件及库路径

```
## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
 include
  ${catkin_INCLUDE_DIRS}
)
link_directories(
 lib
  ${catkin_LIB_DIRS}
)
```

添加依赖库名称 HPS3DSDK，必须与 lib 目录下库名一致

```
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_E
find_package(PCL REQUIRED)
include_directories(include${PCL_INCLUDE_DIRS})
add_executable(hps_camera src/hps_camera.cpp)
target_link_libraries(hps_camera ${catkin_LIBRARIES}  ${PCL_LIBRARIES} HPS3DSDK)
```

修改完成后即可编译并运行；可通过 RVIZ 实时显示点云数据，如下所示：

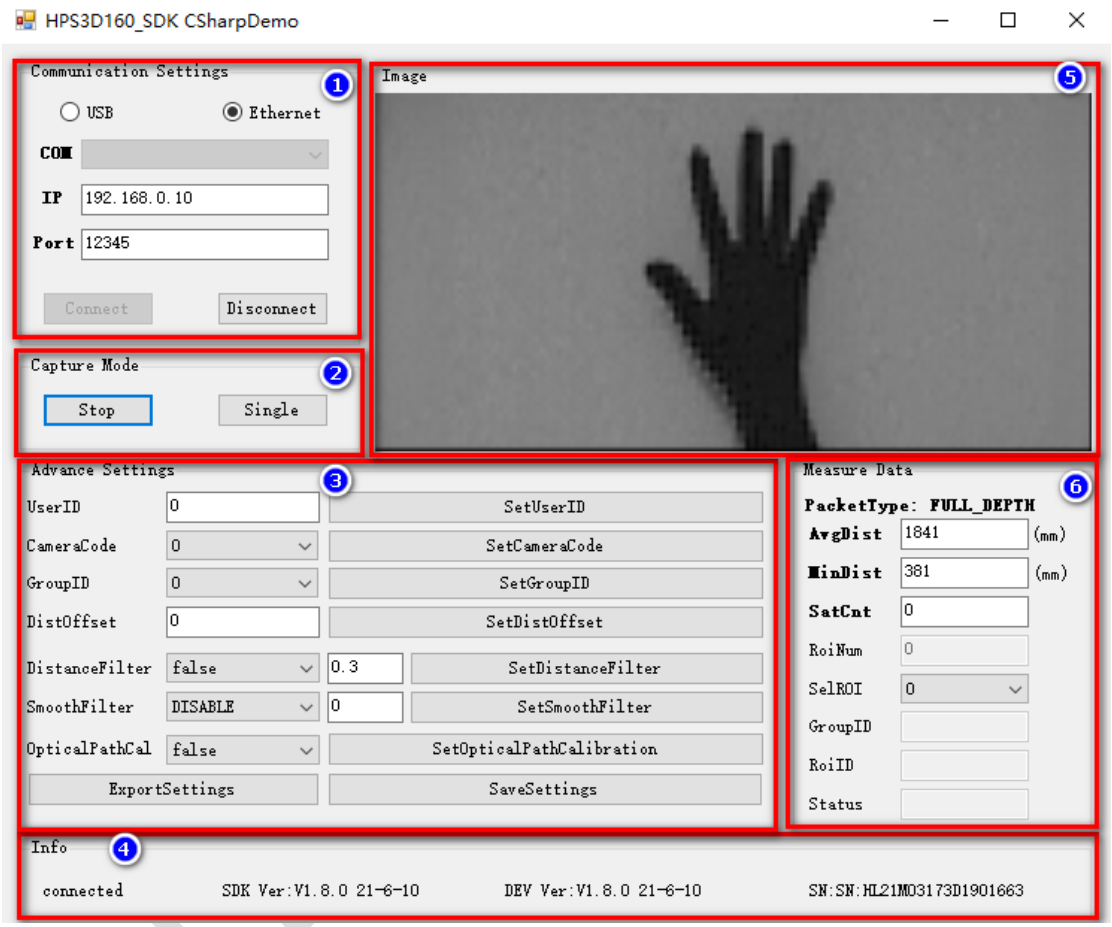


2.2 C# 调用 SDK 基本步骤

为方便用户集成到项目中，我们对基础的 API 进行再次封装处理，在 C#环境下我们封装出 HPS3D160_Device 类，极大的方便了客户在 C#环境下对 HPS3D160 设备进行二次开发处理；

这里以 windows 下的 Microsoft Visual Studio 2017 为例，基于.NET Framework 4.6.1 框架。本示例基于版本号为 V1.8.0 的 SDK 进行编写。

- 打开 HPS3D160_CSharpDemo 工程，检查软件运行目录下是否包含 HPS3D160_SDK.dll；
- 对工程构建并运行，如无错误则可正常连接并执行测量；



在 C#工程中集成 HPS3D160_SDK，仅需将 Demo 程序中 HPS3D160_Device 设备类及 HPS3D160_SDK.dll 动态库拷贝至程序运行目录下即可；

三、API 函数接口

3.1 USB 设备连接

HPS3DAPI_USBConnectDevice

功能	USB 版本设备连接
函数定义	int HPS3DAPI_USBConnectDevice(__IN char* portName,__OUT int* deviceHandler);
参数	portName : USB 设备名(如:COM3 、/dev/ttyACM0) deviceHandler: 返回当前设备的句柄 ID
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	当端口号大于 10 时，需在端口号前加入 \\\\.\\字符，如 \\\\.\\COM15

例:

```
int handle = -1;
int ret = 0;
ret = HPS3DAPI_USBConnectDevice ("COM3", &handle);
if (ret != 0)
{
    printf("USBConnectDevice failed,err:%d\r\n",ret);
    return;
}
```

3.2 以太网设备连接

HPS3DAPI_EthernetConnectDevice

功能	以太网版本设备连接
函数定义	int HPS3DAPI_EthernetConnectDevice(__IN char* controllerIp, __IN uint16_t controllerPort, __OUT int* deviceHandler);
参数	controllerIp : 设备 IP 地址 (默认为 192.168.0.10) controllerPort : 设备端口号（默认为 12345） deviceHandler : 返回当前设备的句柄 ID
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例:

```
int handle = -1;
int ret = 0;
ret = HPS3DAPI_EthernetConnectDevice("192.168.0.10",12345, &handle);
if (ret != 0)
{
    printf("EthernetConnectDevice failed,err:%d\r\n",ret);
```

```
        return;
    }
```

3.3 关闭设备

HPS3DAPI_CloseDevice

功能	关闭设备
函数定义	int HPS3DAPI_CloseDevice(__IN int handle);
参数	handle ： 设备句柄 ID
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例:

```
    int ret = 0;
    ret = HPS3DAPI_CloseDevice(handle);
    if (ret != 0)
    {
        printf("CloseDevice failed,err:%d\r\n",ret);
        return;
    }
```

3.4 设备是否连接

HPS3DAPI_IsConnect

功能	判断设备是否连接
函数定义	int HPS3DAPI_IsConnect(__IN int handle);
参数	handle ： 设备句柄 ID
返回	设备已连接返回 1，否则未连接
注意	

例:

```
    int ret = 0;
    ret = HPS3DAPI_IsConnect(handle);
    if (ret != 1)
    {
        printf("Device is disconnected\r\n");
        return;
    }
```

3.5 设备是否开始测量

HPS3DAPI_IsStart

功能	判断设备是否处于连续测量模式
函数定义	int HPS3DAPI_IsStart(__IN int handle);
参数	handle ： 设备句柄 ID
返回	连续测量返回 1，否则处于待机模式
注意	

例：

```
int ret = 0;
ret = HPS3DAPI_IsStart(handle);
if (ret != 1)
{
    printf("Device is standby.\r\n");
    return;
}
```

3.6 开始连续测量

HPS3DAPI_StartCapture

功能	开始连续采集
函数定义	int HPS3DAPI_StartCapture(__IN int handle);
参数	handle ： 设备句柄 ID
返回	成功返回 1，失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
int ret = 0;
ret = HPS3DAPI_StartCapture(handle);
if (ret != 1)
{
    printf("StartCapture err:%d\r\n",ret);
    return;
}
```

3.7 停止测量

HPS3DAPI_StopCapture

功能	停止采集
函数定义	int HPS3DAPI_StopCapture(__IN int handle);
参数	handle ： 设备句柄 ID
返回	成功返回 1，失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
int ret = 0;
ret = HPS3DAPI_StopCapture(handle);
if (ret != 1)
{
    printf("StopCapture err:%d\r\n",ret);
    return;
}
```

3.8 单次测量

HPS3DAPI_SingleCapture

功能	停止采集
函数定义	int HPS3DAPI_SingleCapture(__IN int handle, __OUT int *type, __OUT uint8_t **data, __OUT int *dataLen);
参数	handle ： 设备句柄 ID type ： 返回采集事件(0:无事件, 1:简单 ROI 数据包 2:完整 ROI 数据包 3:完整深度数据包 4:简单深度数据包 7:系统异常事件 8:异常断开事件 5-6 保留) data ： 返回的数据指针 data 解析方式可参考 demo 程序 dataLen: 返回数据包长度
返回	成功返回 1，失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	单次测量必须同时判定返回值及 type 类型，仅当返回值为 1 且 type 为有效数据包时才表明采集成功，否则采集失败；

例：

```
int ret = 0;
uint8_t *ret_data = NULL;
int dataLen = 0;
int type = 0;
ret = HPS3DAPI_SingleCapture(handle, (int *)type, (uint8_t **)&ret_data,&dataLen);
if (ret != 1)
{
   printf("SingleCapture failed err:%d\r\n",ret);
```

```
        return;
    }
    switch (*type)
    {
        case 1:
        case 2:
        case 3:
        case 4:
                printf("SingleCapture succeed\r\n");
                break;
        default:
                printf("SingleCapture failed type:%d\r\n",*type);
                break;
    }
```

3.7 注册事件回调函数

HPS3DAPI_RegisterEventCallback

功能	注册事件回调函数
函数定义	int HPS3DAPI_RegisterEventCallback(__IN HPS3DAPI_EVENT_CALLBACK eventHandle, __IN void *userPara);
参数	HPS3DAPI_EVENT_CALLBACK : 回调函数指针 原型 : void(*HPS3DAPI_EVENT_CALLBACK)(int handle, int eventType, uint8_t *data,int dataLen, void *userPara); 参数与单次测量一致 userPara : 用户自定义参数，可为空
返回	成功返回 1，失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
void EventCallBackFunc(int handle, int eventType, uint8_t *data,int dataLen, void *userPara)
{
  switch (eventType)
  {
        case 1:
        case 2:
        case 3:
        case 4:
                printf("Is measure data\r\n");
                break;
        default:
```

```
                printf("SYS ERR\r\n");
                break;
        }
    }
    int ret = 0;
    ret = HPS3DAPI_RegisterEventCallback(EventCallBackFunc,NULL);
    if (ret != 1)
    {
        printf("RegisterEventCallback err:%d\r\n",ret);
        return;
    }
```

3.9 注销事件回调函数

HPS3DAPI_UnregisterEventCallBack

功能	注销事件回调函数
函数定义	int HPS3DAPI_UnregisterEventCallBack();
参数	
返回	成功返回 1，失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例:

```
    int ret = 0;
    ret = HPS3DAPI_UnregisterEventCallBack();
    if (ret != 1)
    {
        printf("UnregisterEventCallBack err:%d\r\n",ret);
        return;
    }
```

3.10 获取设备版本信息

HPS3DAPI_GetDeviceVersion

功能	获取设备版本信息
函数定义	const uint8_t* HPS3DAPI_GetDeviceVersion(__IN int handle);
参数	handle: 设备句柄 ID
返回	成功返回设备的版本号，如: V1.8.0 21-6-11
注意	

例：

```
printf("Device Version:%s\n", HPS3DAPI_GetDeviceVersion(handle);
```

3.11 获取 SDK 版本信息

HPS3DAPI_GetSDKVersion

功能	获取 SDK 版本信息
函数定义	const uint8_t* HPS3DAPI_GetSDKVersion(__IN int handle);
参数	handle：设备句柄 ID
返回	成功返回设备的版本号，如：V1.8.0 21-6-11
注意	

例：

```
printf("SDK Version:%s\n", HPS3DAPI_GetSDKVersion(handle);
```

3.12 获取设备序列号

HPS3DAPI_GetSerialNumber

功能	获取设备序列号
函数定义	const uint8_t* HPS3DAPI_GetSerialNumber(__IN int handle);
参数	handle：设备句柄 ID
返回	成功返回设备的版本号，如：SN:HU21M06083D2100162
注意	

例：

```
printf("%s\n", HPS3DAPI_GetSerialNumber(handle);
```

3.13 导出设备配置参数

HPS3DAPI_ExportSettings

功能	导出当前设备配置参数
函数定义	int HPS3DAPI_ExportSettings(__IN int handle, __OUT uint8_t *settings);
参数	Handle ：设备句柄 ID settings ：返回指定设备参数,参数内容如下所示：

	/*设备当前参数*/ typedef struct { int user_id; /*用户自定义ID，默认为0*/ int max_resolution_X; /*X方向分辨率，默认160*/ int max_resolution_Y; /*Y方向分辨率，默认60*/ int max_roi_group_number; /*支持的最大ROI分组数，默认为16*/ int max_roi_number; /*支持的最大ROI数，默认为8*/ int max_threshold_number; /*支持的最大阈值报警数，默认为3*/ int max_multiCamera_code; /*支持的最大多机编号，默认为16*/ int dist_filter_enable; /*距离滤波器开启状态，默认为false*/ float dist_filter_K; /*距离滤波器比例系数*/ int smooth_filter_type; /*平滑滤波器类型*/ int smooth_filter_args; /*平滑滤波器参数*/ int cur_group_id; /*当前ROI分组*/ int cur_multiCamera_code; /*当前多机编号*/ int dist_offset; /*距离补偿值 */ int optical_path_calibration; /*光程补偿开启状态*/ }HPS3D_DeviceSettings_t;
返回	成功返回 1，失败返回错误描述码
注意	

例:

```
int ret = 0;
HPS3D_DeviceSettings_t settings;
ret = HPS3DAPI_ExportSettings(handle,(uint8_t *)&settings);
if (ret != 1)
{
    printf("ExportSettings failed, err:%d\r\n",ret);
    return;
}
```

3.14 保存配置参数至设备

HPS3DAPI_SaveSettings

功能	保存当前配置至传感器
函数定义	int HPS3DAPI_SaveSettings(__IN int handle);
参数	handle: 设备句柄 ID
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
int ret = 0;
ret =HPS3DAPI_SaveSettings(handle);
if(ret != 1)
{
    printf("SaveSettings failed, err:%d\r\n",ret);
    return;
}
```

3.15 设置用户自定义 ID

HPS3DAPI_SetDeviceUserID

功能	设置用户自定义 ID
函数定义	int HPS3DAPI_SetDeviceUserID (__IN int handle, __IN uint8_t userID);
参数	handle: 设备句柄 ID userID: 用户自定义 ID 范围 0-255
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
int ret = 0;
ret =HPS3DAPI_SetDeviceUserID (handle,1);
if(ret != 1)
{
    printf("SaveDeviceUserID failed, err:%d\r\n",ret);
    return;
}
```

3.16 设置敏感区域分组

HPS3DAPI_SetROIGroupID

功能	设置 ROI 组 ID
函数定义	int HPS3DAPI_SetROIGroupID (__IN int handle, __IN uint8_t groupID);
参数	handle: 设备句柄 ID group ID: ROI 组 ID 范围 0-15
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例：

```
int ret = 0;
ret =HPS3DAPI_SetROIGroupID (handle,1);
if(ret != 1)
{
    printf("SaveROIGroupID failed, err:%d\r\n",ret);
    return;
}
```

3.17 设置多机编号

HPS3DAPI_SetMultiCameraCode

功能	设置多机编号
函数定义	int HPS3DAPI_SetMultiCameraCode (__IN int handle, __IN uint8_t CameraCode);
参数	handle: 设备句柄 ID CameraCode: 多机编号 范围 0-15
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例:

```
int ret = 0;
ret =HPS3DAPI_SetMultiCameraCode (handle,1);
if(ret != 1)
{
    printf("SaveMultiCameraCode failed, err:%d\r\n",ret);
    return;
}
```

3.18 设置距离滤波器

HPS3DAPI_SetDistanceFilterConf

功能	设置距离滤波器参数
函数定义	int HPS3DAPI_SetDistanceFilterConf(__IN int handle, __IN int enable, __IN float K);
参数	handle: 设备句柄 ID enable: 开启或关闭距离滤波器 1 表示开启 0 表示关闭 K : 滤波系数，范围 0-1
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	K 值越小滤波效果越明显，距离滤波器可明显缓解距离波动情况，但帧率会受影响

例：

```
int ret = 0;
ret = HPS3DAPI_SetDistanceFilterConf(handle,1,0.3);
if(ret != 1)
{
    printf("SetDistanceFilterConf failed, err:%d\r\n",ret);
    return;
}
```

3.19 设置平滑滤波器

HPS3DAPI_SetSmoothFilterConf

功能	设置平滑滤波器参数
函数定义	int HPS3DAPI_SetSmoothFilterConf(__IN int handle, __IN int type, __IN int args);
参数	handle：设备句柄 ID type：滤波器类型 0 表示关闭滤波器 1 表示平均滤波器 2 表示高斯滤波器 args ：滤波参数，参考值 2 或 3
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	平滑滤波器对平面噪声进行滤波，但帧率会受影响

例：

```
int ret = 0;
ret = HPS3DAPI_SetSmoothFilterConf (handle,1,2);
if(ret != 1)
{
    printf("SetSmoothFilterConf failed, err:%d\r\n",ret);
    return;
}
```

3.20 设置距离补偿值

HPS3DAPI_SetDistanceOffset

功能	设置距离补偿值
函数定义	int HPS3DAPI_SetDistanceOffset(__IN int handle, __IN int16_t offset);
参数	handle: 设备句柄 ID offset: 距离补偿值
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	

例:

```
int ret = 0;
ret = HPS3DAPI_SetDistanceOffset(handle,-100);
if(ret != 1)
{
    printf("SetDistanceOffset failed, err:%d\r\n",ret);
    return;
}
```

3.21 设置光程补偿

HPS3DAPI_SetOpticalPathCalibration

功能	设置光程补偿
函数定义	int HPS3DAPI_SetOpticalPathCalibration(__IN int handle, __IN int enbale);
参数	handle: 设备句柄 ID enable: 1 表示开启 0 表示关闭
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	开启光程补偿后将测量斜线距离转换为垂直传感器所在平面距离

例:

```
int ret = 0;
ret = HPS3DAPI_SetOpticalPathCalibration(handle,1);
if(ret != 1)
{
    printf("SetOpticalPathCalibration failed, err:%d\r\n",ret);
    return;
}
```

3.22 设置边缘滤波器

HPS3DAPI_SetEdgeFilterEnable

功能	设置光程补偿
函数定义	int HPS3DAPI_SetEdgeFilterEnable (__IN int handle, __IN int enbale);
参数	handle: 设备句柄 ID enable: 1 表示开启 0 表示关闭
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	开启边缘滤波器后,被测物与背景之间的过渡数据将可能被滤除

例:

```
int ret = 0;
ret = HPS3DAPI_SetEdgeFilterEnable (handle,1);
if(ret != 1)
{
    printf("SetEdgeFilterEnable failed, err:%d\r\n",ret);
    return;
}
```

3.23 设置设备网口重连

HPS3D_EthternetReconnection

功能	设置设备重连
函数定义	HPS3D_StatusTypeDef HPS3D_EthternetReconnection(__IN int handle);
参数	handle: 设备句柄 ID
返回	成功返回 1,失败返回错误描述码(错误码见 HPS3Duser_IF.h HPS3D_StatusTypeDef)
注意	连接设备的网线断开重新插上后,需要等待 10 秒左右才会继续测量

例:

```
int ret = 0;
ret = (HPS3D_StatusTypeDef)HPS3D_EthternetReconnection(handle);
if (ret == HPS3D_RET_OK)
{
    HPS3D_StartCapture(handle);
    printf("Reconnection Successful:Hardware %d\n", handle);
}
```

四、修订历史纪录

Date	Revision	Description
2021/06/17	V2.1	V1.8 SDK 初始版本
2021/12/03	V2.2	新增边缘滤波器设定接口
2022/06/18	V2.3	新增网口断开重连接口

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Hypersen Technologies Co., Ltd. reserve the right to make changes, corrections, enhancements, modifications, and improvements to Hypersen products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on Hypersen products before placing orders. Hypersen products are sold pursuant to Hypersen's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of Hypersen products and Hypersen assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by Hypersen herein.

Resale of Hypersen products with provisions different from the information set forth herein shall void any warranty granted by Hypersen for such product.

Hypersen and the Hypersen logo are trademarks of Hypersen. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.