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；该 SDK 为二次开发包工具，其提供的接口包含了我司生产研发的 HPS3D160 面阵相机的绝大部分操作指令，请详细阅读该使用手册；

二、将 SDK 集成到 IDE 中

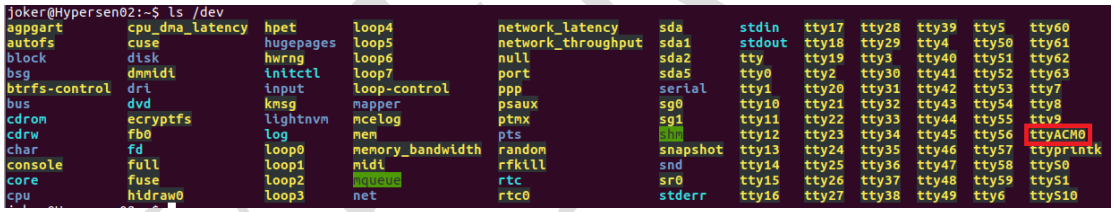
目前嵌入式程序的 IDE 环境大致可分为三种：Keil、IAR 和基于 Eclips 的 IDE（例如 TrueStudio、Simplicity Studio 等），每种的项目管理策略都不一致，但是差异不是很大,以下是不同平台下 SDK 的集成;

2.1 Linux 平台下环境配置及集成到 IDE 中

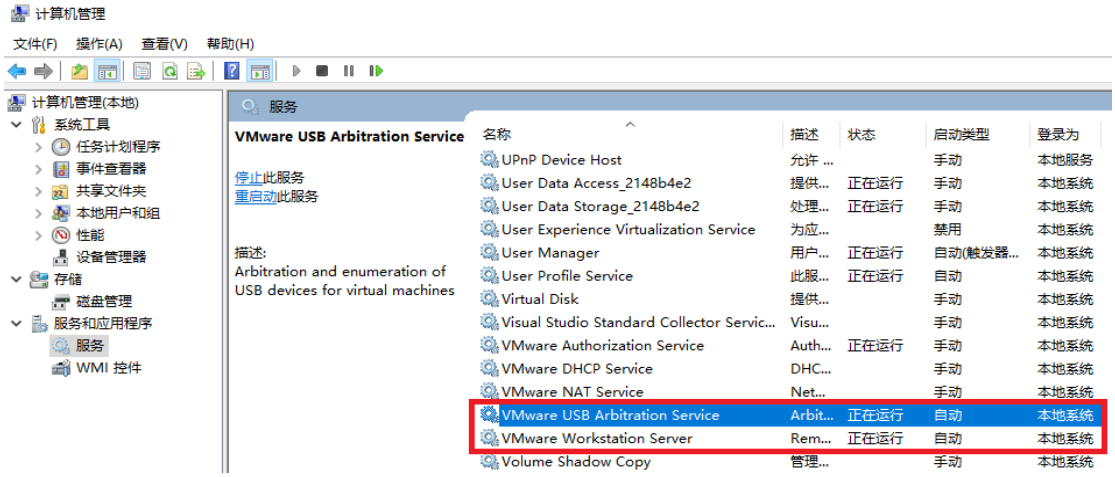
xxx.so 适合在 Linux 操作系统平台使用，这里以 Ubuntu 为例。本示例基于 API 版本号为 2018.12.04 V1.0.3 的 SDK 进行编写。

2.1.1 HPS3D160 设备连接

将 HPS3D160 设备连接到电脑上，打开终端输入 ls /dev，查看设备 ttyACM*，如下图所示：

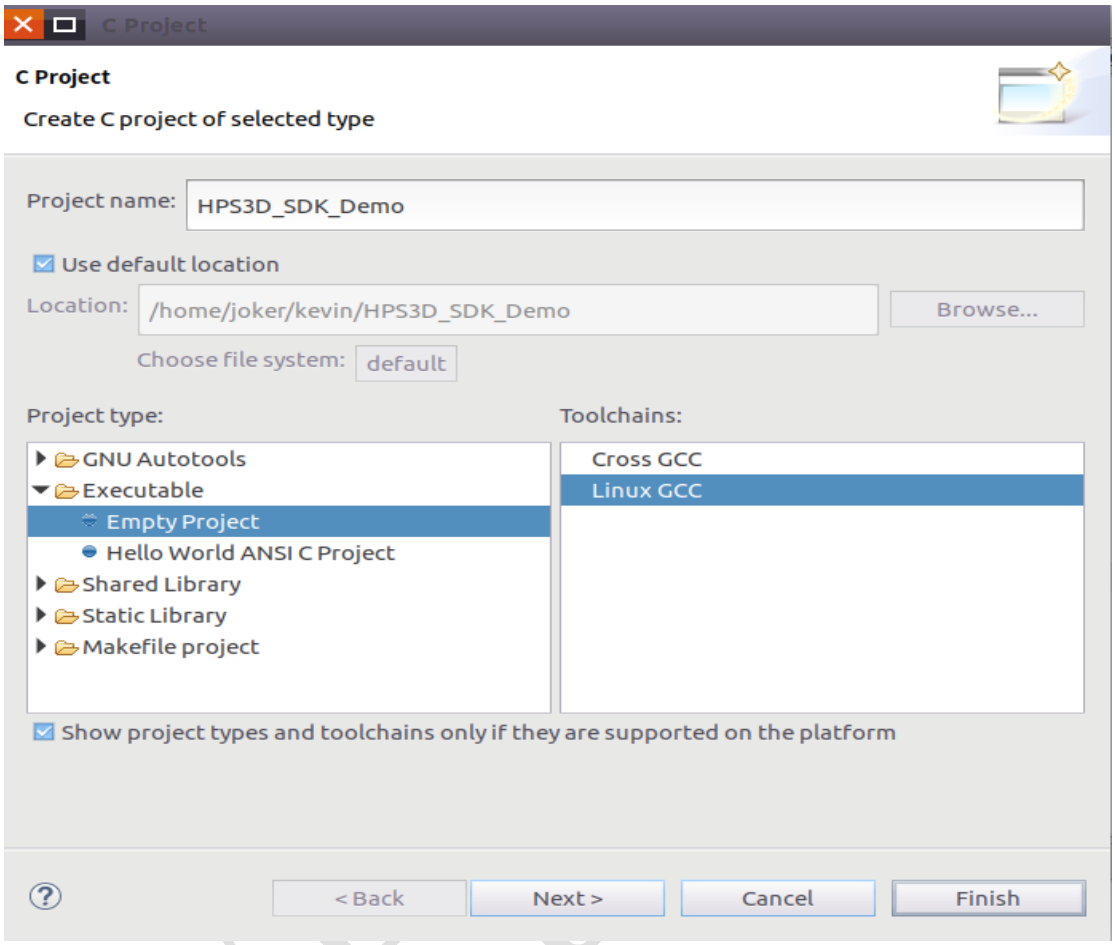


若没有查看到 ttyACM*设备名称，则需要重新拔插，再次查看，若还是没有，则到“计算机管理->服务和应用程序->服务”中查看 VMware USB Arbitration Service 是否正在运行，若禁用，则开启运行，之后重新拔插设备即可；若不想每次登陆虚拟机都启动一次 USB 设备服务，则可将 VMware Workstation Server 和 VMware USB Arbitration Service 均设置成运行和自动，重启计算机即可。

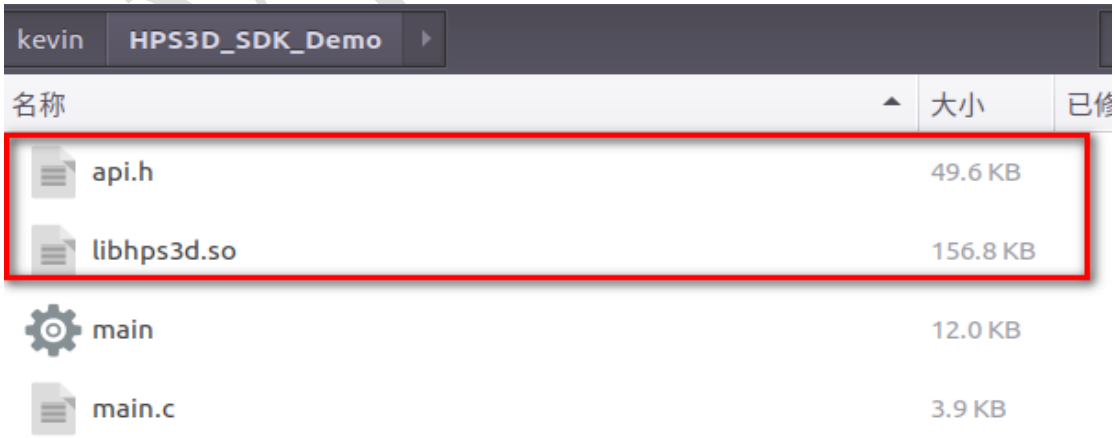


2.1.2 工程环境配置与集成

在 ubuntu 下的 eclipse 建立工程（其他 IDE 工具均可用），这里以 eclipse 为例，如下配置完后，点击 finish 完成工程搭建。



将 api.h 和 libhps3d.so 文件拷贝进工程目录；



打开终端，输入 sudo cp libhps3d.so /usr/local/lib/，将 libhps3d.so 拷贝到/usr/local/lib/目录下；之后执行 sudo ldconfig 进行加载。

```
@Hypersen02:~/kevin/HPS3D_SDK_Demo$ sudo cp libhps3d.so /usr/local/lib/
@Hypersen02:~/kevin/HPS3D_SDK_Demo$ sudo ldconfig
```

在 main.c 写入测试代码后，在终端输入：gcc main.c -L./ -lhps3d -o app，进行编译连接，再使用 sudo ./app 执行程序

```
@Hypersen02:~/kevin/HPS3D_SDK_Demo$
@Hypersen02:~/kevin/HPS3D_SDK_Demo$ gcc main.c -L./ -lhps3d -o app
@Hypersen02:~/kevin/HPS3D_SDK_Demo$
@Hypersen02:~/kevin/HPS3D_SDK_Demo$
@Hypersen02:~/kevin/HPS3D_SDK_Demo$
@Hypersen02:~/kevin/HPS3D_SDK_Demo$ sudo ./app
```

选择可连接的设备，初始化成功后即可正常输出测量结果：

```
当前可连接的设备(请选择):
0: /dev/ttyACM0
请输入对应的序号:
0
初始化成功
测量平均距离值为1706
测量平均距离值为1706
测量平均距离值为1734
测量平均距离值为1715
测量平均距离值为1689
测量平均距离值为1705
测量平均距离值为1724
测量平均距离值为1700
测量平均距离值为1693
测量平均距离值为1707
```

2.1.3 在用户项目中使用 SDK

1、在项目中包含头文件，示例代码：

```
#include "api.h"
```

2、调用设备连接函数，在此之前需要输入 handle->DeviceName 设备名路径，此函数得到 handle->DeviceFd 设备文件描述符和 handle->ConnectStatus 连接状态(true)。Handle->DeviceFd 设备文件描述符在所有的命令函数接口均需要使用。示例代码：

```
HPS3D_HandleTypeDef handle;
RET_StatusTypeDef ret = RET_OK;
ret = HPS3D_Connect(&handle);
if(RET_OK != ret)
{
    printf("设备打开失败! ret = %d\n",ret);
    break;
}
```

3、调用初始化配置函数，在此之前需要执行步骤 2，使用步骤 2 得到的 handle->DeviceFd 设备文件描述符，调用此函数后将对设备进行初始化配置，默认配置为待机模式以及异步通信方式，并创建异步通信线程；可得到 handle->DeviceAddr 设备地址、handle->RoiNumber 设备支持的 ROI 数量、handle->ThresholdNumber 设备 ROI 区域支持的阈值数量、handle->opticalEnable 光学使能参数(用于 3 维点云空间坐标转换),以及对 MeasureData 进行动态分配空间。示例代码：

```
/*设备初始化*/
ret = HPS3D_ConfigInit(&handle);
if(RET_OK != ret)
{
    printf("初始化失败:%d\n", ret);
    break;
```

```
}
printf("初始化成功\n");
```

注：

（1）步骤 2 连接设备函数返回的 handle->DeviceFd 设备文件描述符和步骤 3 配置初始化函数返回的 handle->DeviceAddr 设备地址在所有的命令函数接口中会使用到。

（2）请务必检查初始化函数的返回值，根据返回值判断是否初始化成功。SDK 的绝大多数 API 都提供了操作状态返回，建议用户对每个 API 的返回值进行检查，以确保可靠。

4、根据用户需要调用其他命令函数接口，对相机进行配置，配置完成后设置运行模式。在配置成连续测量命令后，该设置运行模式的命令返回值可能检测不到,此时可忽略该函数返回值；示例代码：

```
/*将运行模式设置成连续模式*/
handle.RunMode = RUN_CONTINUOUS;
HPS3D_SetRunMode(&handle);
```

注：

（1）在所有命令函数接口配置的时候，将会发送停止测量命令。所以在连续测量过程中，调用命令函数接口后，需要重新设置运行模式。

（2）在运行模式配置，可选择三种模式：

①RUN_IDLE：空闲模式，在此模式下，相机进入待机模式。

②RUN_SINGLE_SHOT：单次测量模式，在此模式下，相机进行一次测量，然后自动切换到 RUN_IDLE。有两种测量方式，一种是同步测量，一种是异步测量（默认是异步）。

③RUN_CONTINUOUS：连续测量模式，在此模式下，相机将自动进行连续测量，直到用户手动设定为 RUN_IDLE 或 RUN_SINGLE_SHOT 模式；测量数据通过结构体形式返回，参考 MeasureDataTypeDef 结构体，数据类型通过返回包类型枚举(RetPacketTypedef)进行区分。

5、在所有步骤配置成功后，有两个方式获得测量数据，一种是同步测量方式（只支持单次测量），另一种是异步方式（支持单次测量和连续测量）。

（1）同步测量（只支持单次测量），调用 api.h 中的 HPS3D_SingleMeasurement 函数进行同步单次测量。示例代码如下：

```
HPS3D_SingleMeasurement(&handle);
printf("测量平均距离值为%d\n",handle.MeasureData.full_depth_data->distance_average);
```

注：在 HPS3D_SingleMeasurement 函数内将通信模式 handle.SyncMode 设成同步模式 SYNC，如需设置成异步模式，则在函数后将 handle.SyncMode 设成异步模式 ASYNC。

（2）异步测量（支持单次测量和连续测量）

①设置运行模式，代码如下：

```
/*设置运行模式为连续模式*/
handle.RunMode = RUN_CONTINUOUS;
/*设置运行模式为单次模式*/
handle.RunMode = RUN_SINGLE_SHOT;
HPS3D_SetRunMode(&handle);
```

②编写观察者回调函数，代码如下：

```
/*观察者回调函数*/
void * User_Func(HPS3D_HandleTypedef *handle,AsyncIObserver_t *event)
{
    if(event->AsyncEvent == ISubject_Event_DataRecvd)
    {
```

```
        switch(event->RetPacketType)
        {
            case SIMPLE_ROI_PACKET:
                break;
            case FULL_ROI_PACKET:
                break;
            case FULL_DEPTH_PACKET:
                printf("测量平均距离值为%d
                \n",event->MeasureData.full_depth_data->distance_average);
            break;
            case SIMPLE_DEPTH_PACKET:
                break;
            case NULL_PACKET:
                break;
            default:
                printf("系统错误\n");
                break;
        }
    }
    return 0;
}
```

③观察者初始化，代码如下：

```
/*观察者回调函数与初始化*/
AsyncIObserver_t My_Observer; /*定义观察者*/
My_Observer.AsyncEvent = ISubject_Event_DataRecvd; /*观察者订阅事件为数据接收事件*/
My_Observer.NotifyEnable = true; /*使能观察者*/
My_Observer.ObserverID = 2; /*设置观察者 id，目前仅支持单一观察者*/
```

④添加异步观察者，代码如下：

```
/*添加异步观察者*/
HPS3D_AddObserver(&User_Func,&handle,&My_Observer); /*添加观察者*/
```

6、步骤 1-5 的配置就可以正常测量数据，默认得到的数据是完整的深度图数据包（含深度数据）。返回的数据包类型有四种：简单 ROI 数据包（不含深度数据）、完整 ROI 数据包（含深度数据）、简单深度数据包（不含深度数据）、完整深度数据包（含深度数据）。还可以设置转换成点云数据包(只有完整 ROI 数据包和完整深度数据包才能进行点云格式的转换)。

注：深度数据存储在一维数组，按顺序存放，若需要自行遍历。

（1）默认是完整的深度图数据包（含深度数据）输出，若要配置成简单数据包输出，则需调用函数 HPS3D_SetPacketType，进行设置。代码如下：

```
/*设置数据包类型*/
handle.PacketType = PACKET_SIMPLE;/*设置成简单数据包*/
HPS3D_SetPacketType(&handle);
```

（2）若是要配置成 ROI 数据包（简单和完整包同（1）配置），只需设定好 ROI 区域，并使能 ROI 即可，若需设置阈值自行配置，代码如下：

```
ROIConfTypeDef roi_conf;
/*设定 ROI 区域*/
```

```
roi_conf.roi_id = 0;
roi_conf.left_top_x = 10;
roi_conf.left_top_y = 10;
roi_conf.right_bottom_x = 30;
roi_conf.right_bottom_y = 20;
HPS3D_SetROIRegion(&handle, roi_conf);
HPS3D_SetROIEnable(&handle, roi_conf.roi_id, true);
```

（3）若要配置成点云数据输出，则需要在设置运行模式之前，调用光学参数使能函数 HPS3D_SetOpticalEnable 和使能点云数据输出的函数 HPS3D_SetPointCloudEn，在单次测量和连续测量下返回的数据就是点云数据，代码如下：

```
HPS3D_SetOpticalEnable(&handle, true);
HPS3D_SetPointCloudEn(true);
```

本 SDK 得到的点云数据是有序点云数据，点云数据格式是使用（x,y,z）空间坐标作为点云数据；提供结构体（在 api.h 中），代码如下：

```
/*每点的点云数据*/
typedef struct
{
    float32_t x;                    /*x,y,z 空间坐标*/
    float32_t y;
    float32_t z;
}PerPointCloudDataTypeDef;
/*有序点云数据*/
typedef struct
{
    PerPointCloudDataTypeDef point_data[MAX_PIX_NUM]; /*点云坐标 */
    uint16_t width;                                   /*宽度：一行，点的数目*/
    uint16_t height;                                  /*高度：行的总数*/
    uint32_t points;                                  /*点云图，点的总数*/
}PointCloudDataTypeDef;
```

注：

（1）在使能输出点云数据前需要开启光学使能，目的：为了得到垂直距离。

（2）在深度图的数据中，有无效点，在这里处理无效点也给出空间坐标，空间坐标(x,y,z)值为：z = distance[]（保留原有的无效数据值）；x,y 则是体现了 z 在 distance 中的位置（即无效点的位置）

（3）在返回数据包结构体 MeasureDataTypeDef 类型中，定义的点云数据包是结构体数组 PointCloudDataTypeDef，对于深度图数据转换的点云数据存储在数组的[0]中，对于 ROI 数据转换的点云数据则在数组按顺序存储。

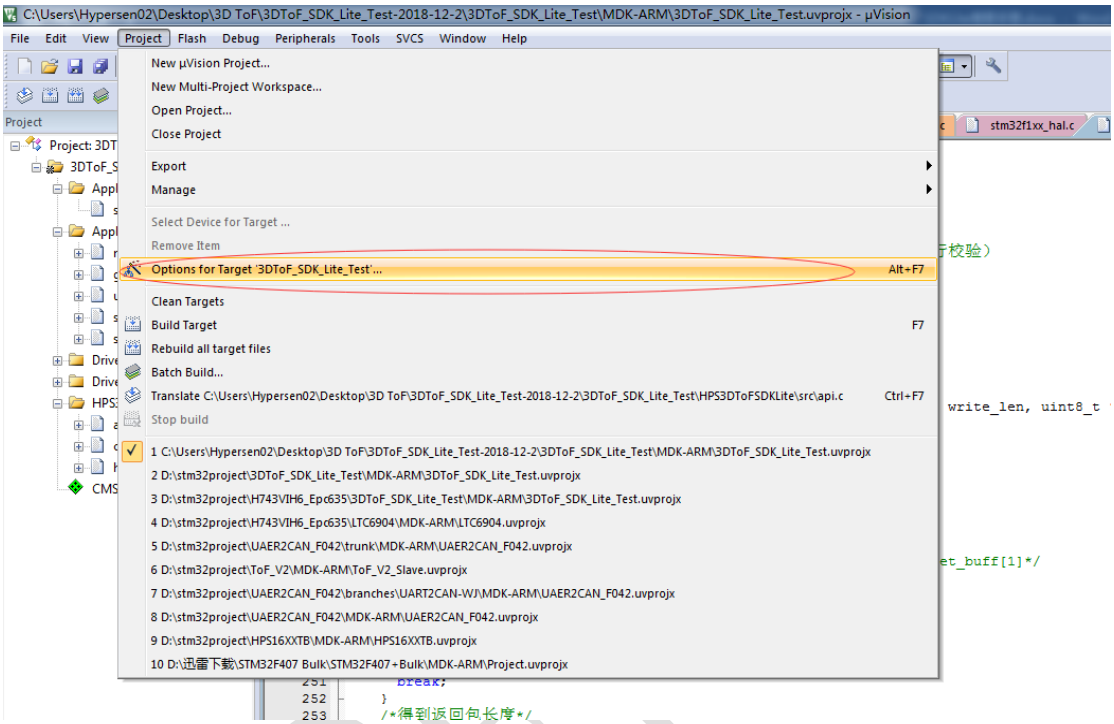
2.2 SDK Lite 集成到 IDE 中

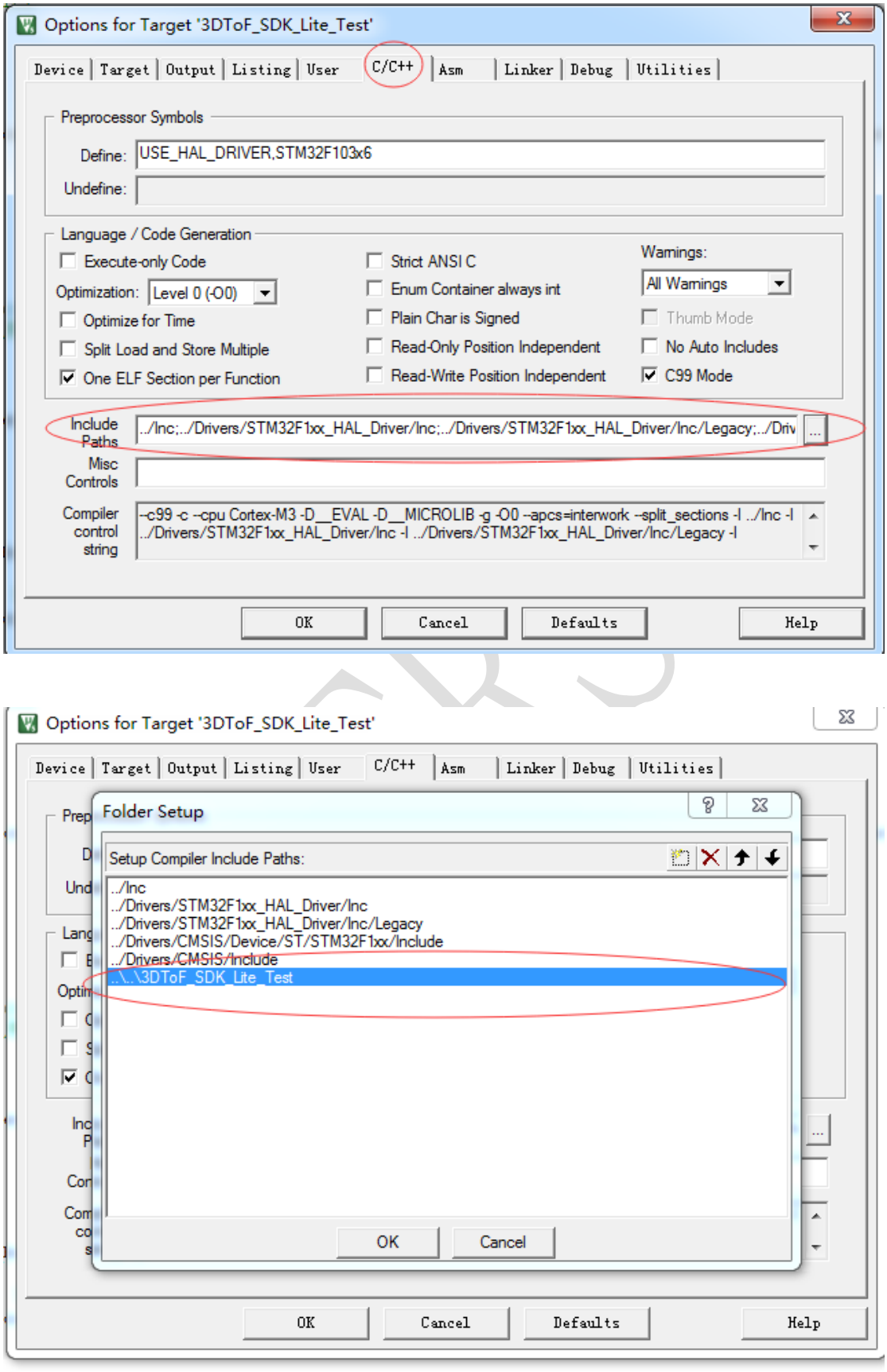
SDK Lite 提供了 HPS3D160 深度相机的轻量级应用程序接口，适合没有跑操作系统的大多数单片机平台使用，鉴于各种 MCU 平台的差异性，提供了源码级的 SDK，因此需要集成进用户的项目源码进行编译。SDK Lite 为轻量级二次开发包，仅提供基础的操作接口，由于深度图和完整的 ROI（敏感区域）数据占用内存较大（可能需要 1Mbyte 以上的 RAM），所

以 SKD Lite 目前不支持深度图数据和完整的 ROI 数据，仅支持精简的数据包格式解析；该示例基于 SDK 版本号为 2018.12.03 V1.0.0 的 SDK 进行编写。

2.2.1 添加头文件搜索路径

SDKLite 中的头文件包含的路径使用的是相对于项目根目录的相对路径，要保证 SDK 能够正常编译，需要在 IDE 中将项目根目录路径包含进头文件搜索路径中，以 Keil 为例:

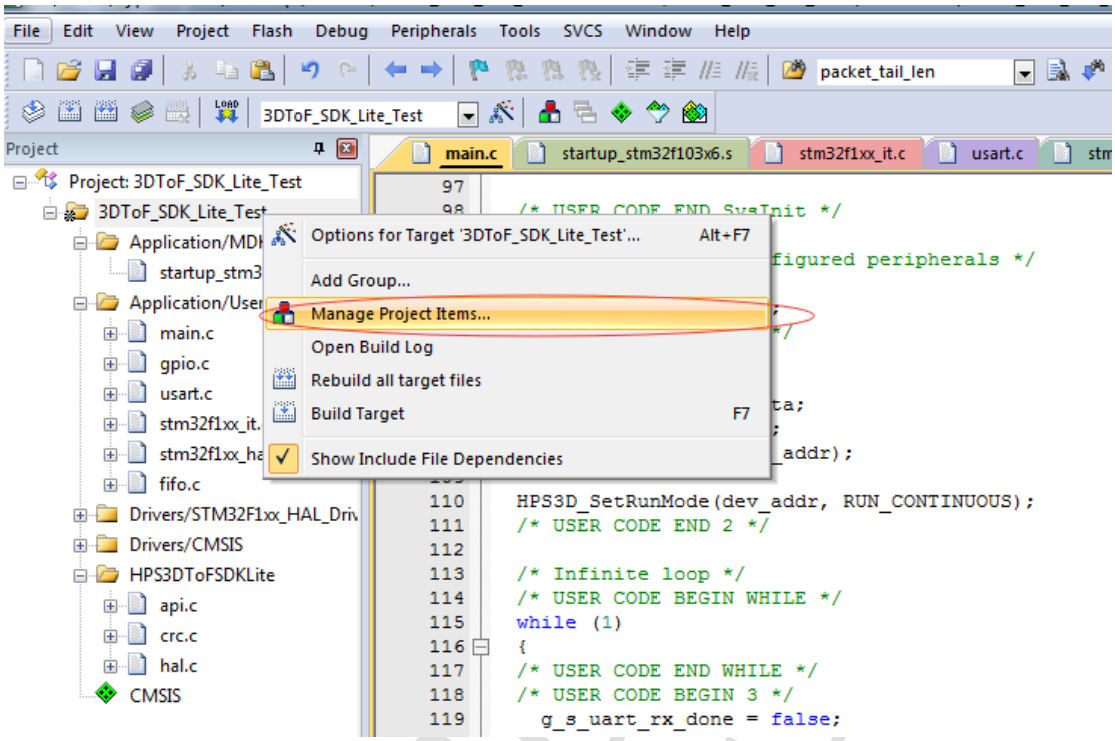


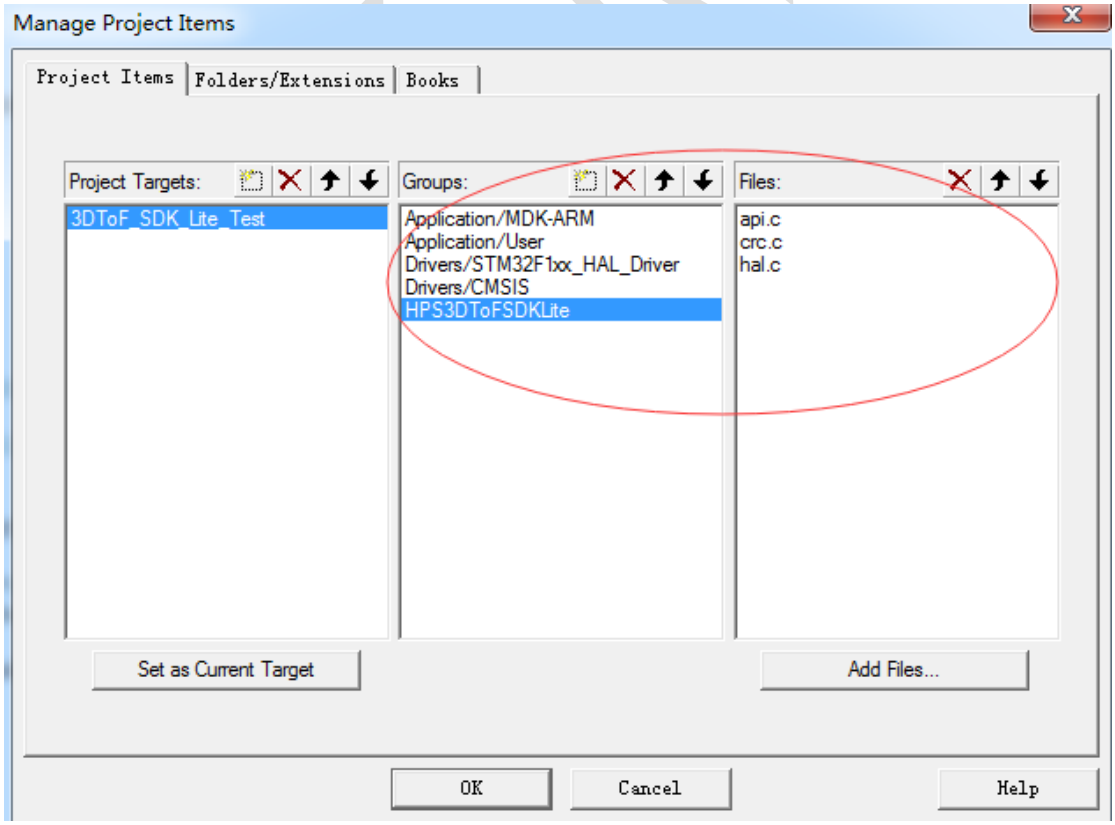


图上 3DToF_SDK_Lite_Test 为项目的根目录。

2.2.2 将 SDK 目录源码添加到项目中

以 Keil 为例，将 SDK 目录源文件添加到项目中：





2.2.3 移植到用户的平台

SDK Lite 仅支持 RS232 和 RS485 接口的 HPS3D160 深度相机，由于平台的差异性，需要

对通讯接口等底层进行移植。

1、 编辑 HPS3DToFSDKLite/src/hal.c 文件

2、 适配 Uart_Read 和 Uart_Write 等接口即可，以基于 HAL 库的 STM32 平台为例：

```
void Uart_Read(uint8_t *dest_buff, uint16_t length, uint32_t timeout_ms)
{
  HAL_UART_Receive(&huart1, dest_buff, length, timeout_ms);
}

void Uart_Write(uint8_t *from_buff, uint16_t length, uint32_t timeout_ms)
{
  HAL_UART_Transmit(&huart1, from_buff, length, timeout_ms);
}

void Delay_Ms(uint16_t ms)
{
  HAL_Delay(ms);
}
```

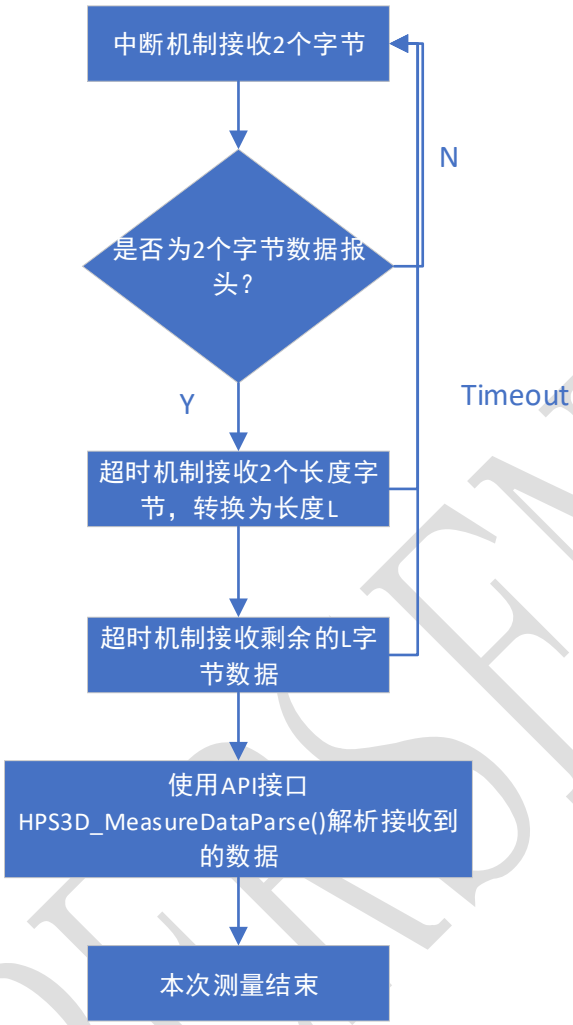
2.2.4 在用户项目中使用 SDK

1. 在源文件中包含头文件，示例代码：
   #include "HPS3DToFSDKLite/inc/api.h"
2. 调用初始化 API 对相机进行相应初始化，该函数会将相机的数据包格式设定为精简格式，并停止当前的连续测量，返回当前的相机的设备地址，这个地址在其他 API 中会使用到，请务必检查初始化函数的返回值，根据返回值判断是否初始化成功。SDK 的绝大多数 API 都提供了操作状态返回，建议用户对每个 API 的返回值进行检查，以确保可靠。示例代码：
   HPS3D_Initialize(&dev_addr);
3. 通过步骤 2 后便可以使用 SDK 中的 API 对数据包进行解析以及配置相机的参数。

2.2.5 使用 SDK 解析测量数据包

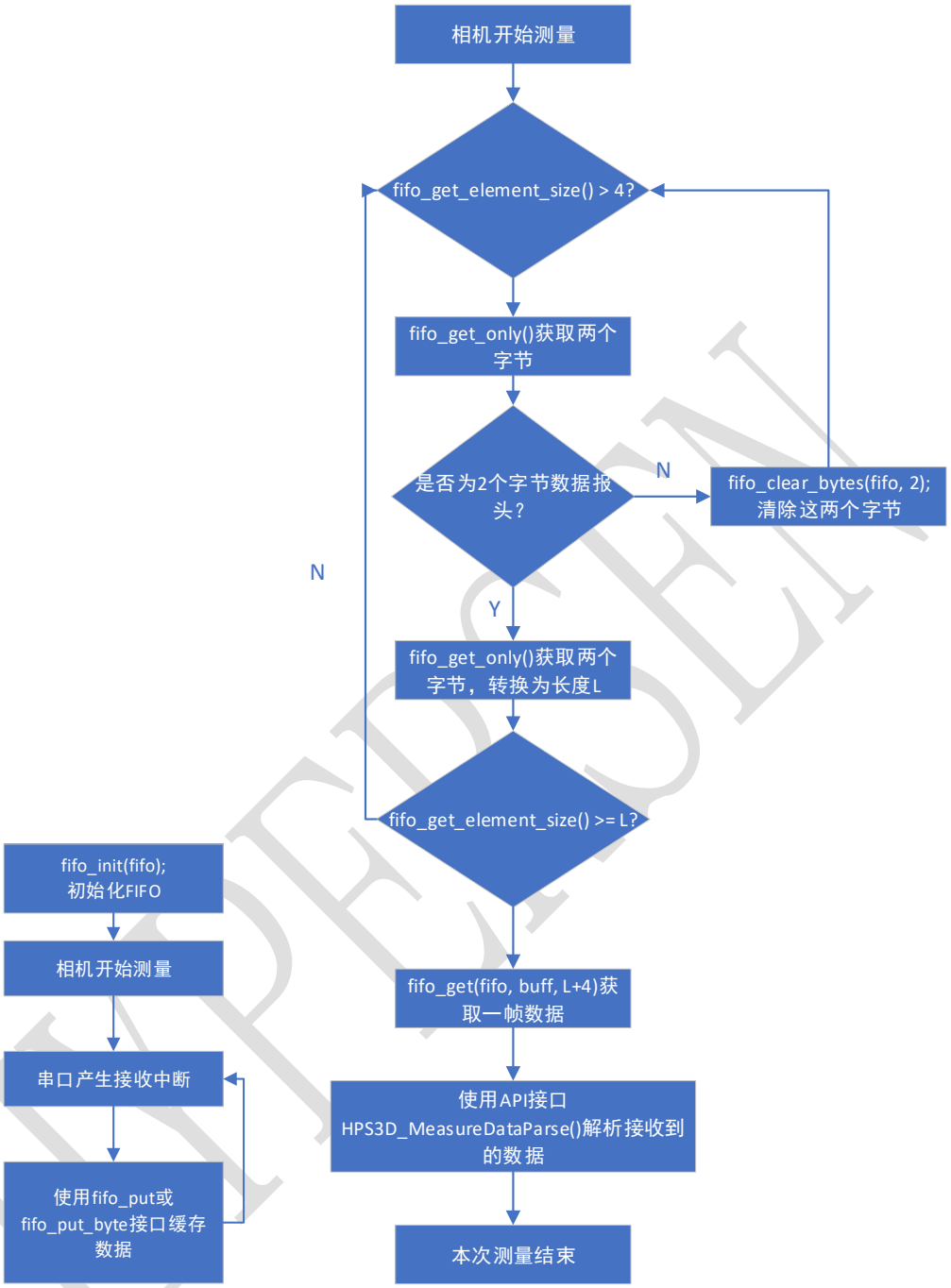
SDK 提供了数据解析 API，但不实现测量数据的接收，因为这与平台相关性太强。

1. 设定相机的运行模式，可选三种模式：
   1） RUN_IDLE:空闲模式，在此模式下，相机进入待机模式，不进行任何测量。
   2） RUN_SINGLE_SHOT:单次测量模式，设定此模式后，相机进行一次测量，然后自动切换到 RUN_IDLE。
   3） RUN_CONTINUOUS:连续测量模式，设定此模式后，相机将自动进行连续测量，直到用户手动设定为 RUN_IDLE 或 RUN_SINGLE_SHOT 模式，在此模式下，数据会通过 RS485 或 RS232 连续输出测量数据。
2. 串口接收测量数据包，数据包格式请参考 HPS-3D160 规格书，请按照格式定义接收数据包，可参考以下的数据接收流程结合自身平台进行相应改进：
   1） 中断+超时机制接收测量数据
   这个接收机制的优势在于不需要额外的数据缓冲区，但是对于外设中断比较多的环境可能会出现丢帧现象，所以该机制比较适合负载低的环境或 RAM 吃紧的平台，具有较高的内存空间利用率，但牺牲了时间利用率。



2） 数据异步缓存机制

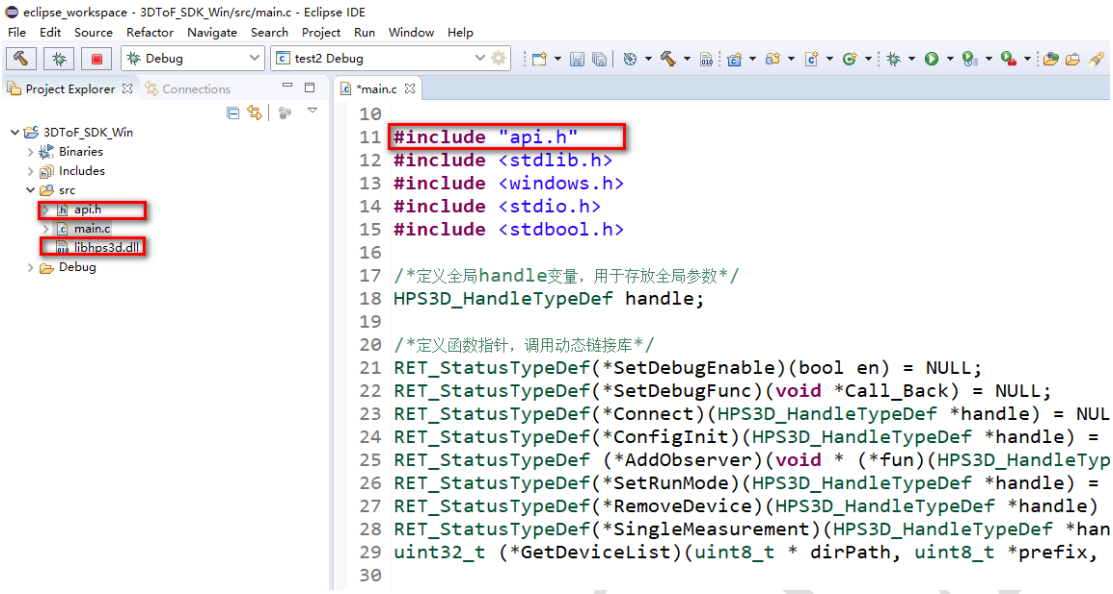
这个数据接收机制需要单独开辟一个数据缓冲区用于缓存数据，可使用循环FIFO（SDK 测试程序已提供 fifo 模块），对于负载相对较高的环境可极大缓解来不及接收数据引起的丢帧现象，该机制的实现需要开启串口中断，串口每次产生接收中断就将数据缓存到 FIFO 中，主程序只需要对 FIFO 的数据长度进行检查判断，可充分利用方式 1）的等待时间，具有更高的时间利用率，但牺牲了空间利用率。



2.3 Windows 平台下将 SDK 集成到 IDE 中

xxx.dll 适合在 Windows 操作系统平台使用，这里以 windows 下的 eclipse 为例。本示例基于 API 版本号为 2018.12.08 V1.0.0 的 SDK 进行编写。

2.3.1 添加 xxx.dll 动态链接库与 api.h 头文件到项目中



xxx.dll 动态链接库文件也可放置与其他路径下,调用时填写绝对路径即可;

2.3.2 在用户项目中使用 SDK

以下示例采用 windows 下的 LoadLibraryA 函数接口加载动态链接库：

1、在项目中包含 api.h 头文件,将 api.h 头文件包含在工程目录下

```
#include "api.h"
```

2、定义使用到的函数指针

```
/*定义函数指针，调用动态链接库*/
RET_StatusTypeDef(*SetDebugEnable)(bool en) = NULL;
RET_StatusTypeDef(*SetDebugFunc)(void *Call_Back) = NULL;
RET_StatusTypeDef(*Connect)(HPS3D_HandleTypeDef *handle) = NULL;
RET_StatusTypeDef(*ConfigInit)(HPS3D_HandleTypeDef *handle) = NULL;
RET_StatusTypeDef (*AddObserver)(void * (*fun)(HPS3D_HandleTypeDef *,
AsyncIObserver_t *), HPS3D_HandleTypeDef *handle, AsyncIObserver_t
*Observer_t) = NULL;
RET_StatusTypeDef(*SetRunMode)(HPS3D_HandleTypeDef *handle) = NULL;
RET_StatusTypeDef(*RemoveDevice)(HPS3D_HandleTypeDef *handle) = NULL;
RET_StatusTypeDef(*SingleMeasurement)(HPS3D_HandleTypeDef *handle,
MeasureDataTypeDef *MeasureData, RetPacketTypedef *PacketType) = NULL;
uint32_t (*GetDeviceList)(uint8_t * dirPath, uint8_t *prefix, uint8_t
fileName[DEV_NUM][DEV_NAME_SIZE]) = NULL;
```

3、加载动态库并获取相应函数地址

```
    /*加载动态链接库*/
HMODULE module = LoadLibraryA("libhps3d.dll");
    DWORD error_id = GetLastError();
    if (module == NULL)
    {
```

```
        system("error load");
        return 0;
    }
    SetDebugEnable = (RET_StatusTypeDef(*)(bool en))GetProcAddress(module,
"HPS3D_SetDebugEnable");
    SetDebugFunc = (RET_StatusTypeDef(*)(void
*Call_Back))GetProcAddress(module, "HPS3D_SetDebugFunc");
    Connect = (RET_StatusTypeDef(*)(HPS3D_HandleTypeDef
*handle))GetProcAddress(module, "HPS3D_Connect");
    ConfigInit = (RET_StatusTypeDef(*)(HPS3D_HandleTypeDef
*handle))GetProcAddress(module, "HPS3D_ConfigInit");
    AddObserver = (RET_StatusTypeDef(*)(void* (*fun)(HPS3D_HandleTypeDef *,
AsyncIObserver_t *), HPS3D_HandleTypeDef *, AsyncIObserver_t
*))GetProcAddress(module, "HPS3D_AddObserver");
    SetRunMode = (RET_StatusTypeDef(*)(HPS3D_HandleTypeDef
*handle))GetProcAddress(module, "HPS3D_SetRunMode");
    RemoveDevice = (RET_StatusTypeDef(*)(HPS3D_HandleTypeDef
*handle))GetProcAddress(module, "HPS3D_RemoveDevice");
    SingleMeasurement = (RET_StatusTypeDef(*)(HPS3D_HandleTypeDef *,
MeasureDataTypeDef *, RetPacketTypedef *))GetProcAddress(module,
"HPS3D_SingleMeasurement");
    GetDeviceList = (uint32_t(*)(uint8_t * , uint8_t *, uint8_t
fileName[DEV_NUM][DEV_NAME_SIZE]))GetProcAddress(module,
"HPS3D_GetDeviceList");
```

4 调用相应的接口函数进行设备连接，设备初始化配置等操作，详细示例请参考 Windows 平台下的 Demo 程序；

①设备连接

```
handle.DeviceName = "\\\\.\\COMxx";      /*设置设备连接的端口号*/
ret = Connect(&handle);  /*设备连接*/
if (ret != RET_OK)
{
    printf("设备连接失败! ret = %d\n", ret);
}
```

②设备初始化,默认配置为待机模式以及异步通信方式，主要实现创建异步线程，获取设备地址(其余命令均使用到设备地址),分配内存空间等

```
/*设备初始化*/
ret = ConfigInit(&handle);
if (RET_OK != ret)
    {
        printf("初始化失败:%d\n", ret);
    }
    printf("初始化成功\n");
```

③设置测量模式：单次测量或连续测量

```
/*连续测量*/
```

```
handle.RunMode = RUN_CONTINUOUS;
HPS3D_SetRunMode(&handle);

/*单次测量*/
Handle.RunMode = RUN_SINGLE_SHOT;
HPS3D_SetRunMode(&handle);
```

按照以上步骤配置完成后即可得到默认配置下测量输出的完整深度数据包；

其中，连续模式需要异步线程来连续输出，此时可使用观察者模式来监听测量返回数据，示例代码如下：

```
/*用户观察者回调函数*/
void* User_Fun(HPS3D_HandleTypeDef *handle,AsyncIObserver_t *event)
{
    if(event->AsyncEvent == ISubject_Event_DataRecvd)
    {
        switch(event->RetPacketType)
        {
            case SIMPLE_ROI_PACKET:
                printf("测量平均距离值为%d\n",event->MeasureData.simple_roi_data[0].distance_average);
                break;
            case FULL_ROI_PACKET:
                printf("测量平均距离值为%d\n",event->MeasureData.full_roi_data[0].distance_average);
                break;
            case FULL_DEPTH_PACKET:
                printf("测量平均距离值为%d\n",event->MeasureData.full_depth_data->distance_average);
                break;
            case SIMPLE_DEPTH_PACKET:
                printf("测量平均距离值为%d\n",event->MeasureData.simple_depth_data->distance_average);
                break;
            case NULL_PACKET:
                break;
            default:
                printf("系统错误\n");
                break;
        }
    }
}

/*定义观察者并初始化*/
AsyncIObserver_t My_Observer;
/*观察者订阅事件为数据接收事件,即有数据则立即通知观察者*/
```

```
My_Observer.AsyncEvent = ISubject_Event_DataRecvd;
My_Observer.NotifyEnable = true; /*使能观察者*/
My_Observer.ObserverID = 2;  /*观察者ID*/

/*添加观察者*/
ret = HPS3D_AddObserver(&User_Fun,&handle,&My_Observer);
if(RET_OK != ret)
{
    printf("观察者添加失败:%d\n", ret);
}
```

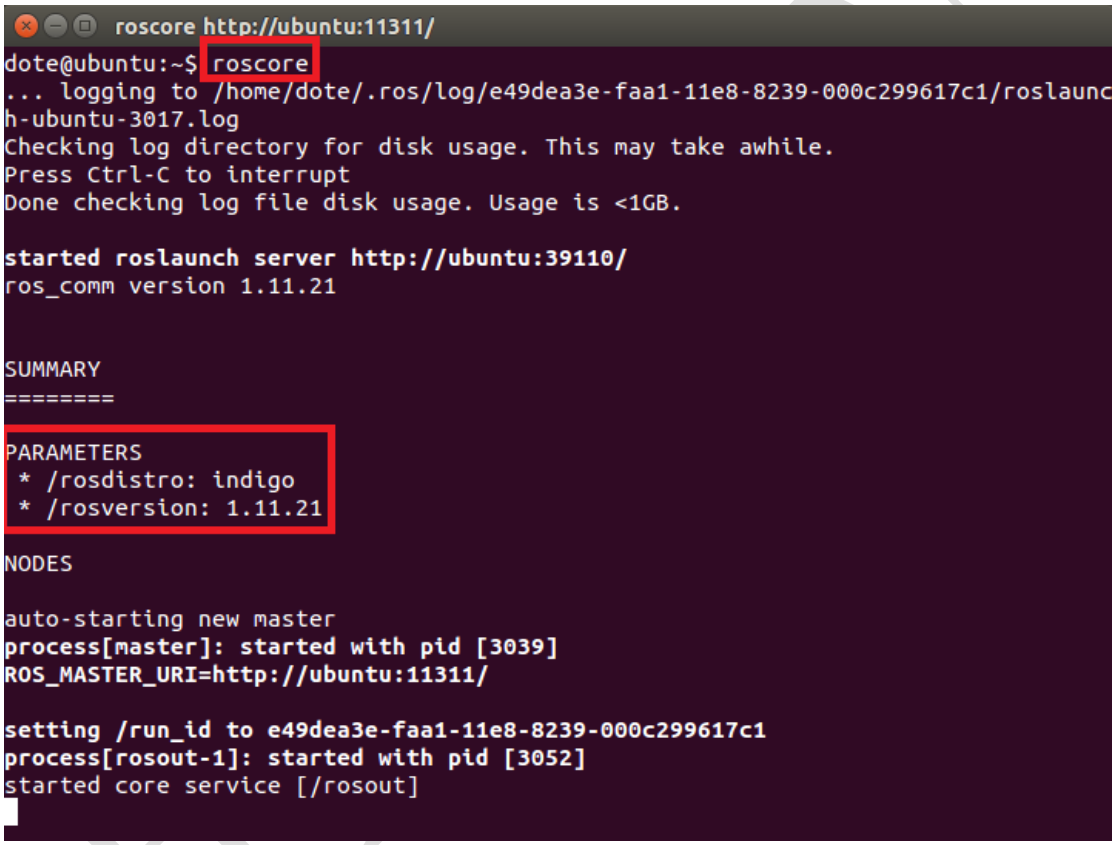
单次测量模式可支持同步也可支持异步方式，异步方式同上,建议使用同步方式；同步方式示例如下：注：同步单次测量函数会将 handle.SyncMode 设置为 SYNC 同步方式，如需切换至异步方式需将该参数设置为 ASYNC；

```
ret = HPS3D_SingleMeasurement(&handle);
if(ret == RET_OK)
{
    switch(handle.RetPacketType)
    {
        case SIMPLE_ROI_PACKET:
            printf("单次测量平均距离值为%d
\n",handle.MeasureData.simple_roi_data[0].distance_average);
            break;
        case FULL_ROI_PACKET:
            printf("单次测量平均距离值为%d
\n",handle.MeasureData.full_roi_data[0].distance_average);
            break;
        case FULL_DEPTH_PACKET:
            printf("单次测量平均距离值为%d
\n",handle.MeasureData.full_depth_data->distance_average);
            break;
        case SIMPLE_DEPTH_PACKET:
            printf("单次测量平均距离值为%d
\n",handle.MeasureData.simple_depth_data->distance_average);
            break;
        case NULL_PACKET:
            printf("无测量数据包\n");
            break;
        default:
            printf("系统错误\n");
            break;
    }
}
```

2.4 ROS 平台下环境配置及将 SDK 集成到 IDE 中

SDK ROS 提供了 HPS3D160 深度相机的应用程序接口，生成的 32 位/64 位的.so 动态链接库适合 Linux 操作系统上 ROS 平台使用，通过的.so 和 api.h 接口可集成进用户的项目源码进行编译。SDK 为二次开发包，仅提供基础的操作接口。可以获得深度图和完整 ROI（敏感区域）数据，若需要转换为点云数据，则在获取数据前，使能转换点云数据的接口即可。本文档基于 API 版本号为 2018.12.10 V1.0.0 的 SDK 进行编写。

SDK ROS 适合在 Linux 操作系统上 ROS 平台使用，这里以 Ubuntu14.04 为例。因为安装的是 Ubuntu14.04 的 Linux 操作系统，所以安装与之对应的 ROS 版本为发行版本 indigo，这里安装 1.11.21 版本。在终端输入 rosrun，运行 ROS 总线，可查看运行 ROS 的版本，如下图所示：

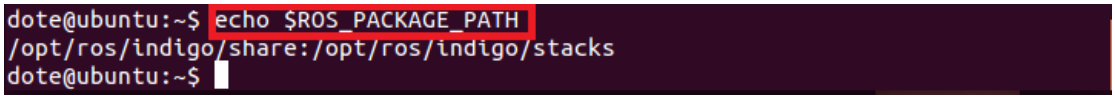


2.4.1 HPS3D160 设备连接

与本文档的“2.1.1 HPS3D160 设备连接”的配置相同，请参照本文档的“2.1.1 HPS3D160 设备连接”的内容。

2.4.2 创建一个工作空间

在创建一个工作空间前，首先查看环境变量，在终端输入 echo $ROS_PACKAGE_PATH，查看 Linux 上的环境变量，如下图所示：



然后要检查是否安装 catkin 工具，若没有安装的话，请先安装 catkin 工具。默认情况下，是有 catkin 工具的，catkin 是 ROS 的一个官方的编译构建系统，是原本的 ROS 的编译构建系统 rosbuild 的后继者。

（1）生效 ROS 系统的环境变量，在终端输入 source /opt/ros/indigo/setup.bash。

（2）这里在家目录下创建一个工作空间，输入 mkdir -p ~/HPS3D_SDK_ROS_Demo/src。

（3）到 src 目录下，输入 cd ~/HPS3D_SDK_ROS_Demo/src，执行初始化空间，输入 catkin_init_workspace，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ cd ~/HPS3D_SDK_ROS_Demo/src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ catkin_init_workspace
Creating symlink "/home/dote/HPS3D_SDK_ROS_Demo/src/CMakeLists.txt" pointing to
"/opt/ros/indigo/share/catkin/cmake/toplevel.cmake"
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$
```

（4）通过上面配置，就可以创建一个工作空间，并转到已创建好的工作空间之下，输入 cd ~/HPS3D_SDK_ROS_Demo/；尽管这个空间是空的，我们仍然可以构建（build）它，输入 catkin_make，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ cd src/
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ cd ~/HPS3D_SDK_ROS_Demo/src
```

（5）输入 ls 查看当前的工作目录，会发现里面多了两个文件夹“build”和“devel”。在 devel 文件夹下，您可以看到很多 setup.*sh 文件。输入 source devel/setup.bash，配置您的工作空间，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls devel/
env.sh  lib  setup.bash  setup.sh  setup_util.py  setup.zsh
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$
```

注：任何的源文件、python 库、脚本，以及其他的静态文件，将会留在源空间 src 中。然而所有产生的文件，像库文件、可执行文件以及产生的代码都被放置在 devel 中。

2.4.3 创建一个 ROS 包（catkin 包）

（1）创建一个名字为 hps_camera 的包，它直接依赖于以下三个包：std_msgs,rospy 以及 roscpp，在终端输入 cd ~/HPS3D_SDK_ROS_Demo/src/进入 src 目录下，再输入 catkin_create_pkg hps_camera std_msgs rospy roscpp，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ cd ~/HPS3D_SDK_ROS_Demo/src/
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ catkin_create_pkg hps_camera std_msgs rosp
y roscpp
Created file hps_camera/CMakeLists.txt
Created file hps_camera/package.xml
Created folder hps_camera/include/hps_camera
Created folder hps_camera/src
Successfully created files in /home/dote/HPS3D_SDK_ROS_Demo/src/hps_camera. Plea
se adjust the values in package.xml.
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$
```

（2）在终端输入 rospack depends hps_camera，可以查看到包可以有很多依赖关系，可看到本身加的依赖的三个包：std_msgs、rospy 和 roscpp，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ rospack depends hps_camera
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
gencpp
genlisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
rospack
roslib
rospy
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```

注：如果在执行 rospack depends hps_camera 时，出现如下错误（**在使用 ros*命令时，出现错误，有可能就是工作空间失效了，重新生效即可**，也可将其写入 ros 环境变量中），则回到工作空间目录，再次执行 source devel/setup.bash，生效工作空间即可，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ rospack depends hps_camera
[rospack] Error: no such package hps_camera
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ cd ..
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
```

2.4.4 创建 ROS 消息 msg 和服务 srv

msg 文件是描述 ROS 中消息的域的简单的文本文件，它用来为消息产生不同语言的源代码。

srv 文件描述一个服务，它由两部分组成，请求和服务。

提示：一个 msg 文件或一个 srv 文件相当于一个结构体，所以可以对照提供的 api.h，在深度相机数据返回包内有五种数据类型，每一种数据类型都是一个结构体，即在写 msg 文件时，可以进行嵌套。

1、创建 ROS 消息 msg

（1）在创建的包中，创建一个消息 msg 目录，存放 msg 文件。在终端输入 cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/，到包目录；输入 mkdir msg，创建 msg 目录；输入 echo "uint16 distance_average" > msg/distance.msg，创建 distance.msg 文件，并写入平均距离的变量。如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src$ cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt  include  package.xml  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ mkdir msg
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt  include  msg  package.xml  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ echo "uint16 distance_average"
> msg/distance.msg
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo cat msg/distance.msg
uint16 distance_average
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```

（2）在终端输入 sudo gedit package.xml，对 package.xml 进行配置，在文件中添加以下两句代码，如下图所示：

```
<build_depend> message_generation </build_depend>
<exec_depend> message_runtime </exec_depend>
```

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<build_depend> message_generation </build_depend>
<exec_depend> message_runtime </exec_depend>
```

到这里 package.xml 配置完成，保存并退出。

（3）在终端输入 sudo gedit CMakeLists.txt，对 CMakeLists.txt 进行配置，找到相应位置进行修改。

①添加 message_generation 到如下代码片，添加后结果如下图所示：

```
## 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
)
```

②添加 CATKIN_DEPENDS message_runtime 到如下代码片，添加后结果如下图所示：

```
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES hps_camera
#  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system lib
   CATKIN_DEPENDS message_runtime
)
```

③找到如下图代码片：

```
## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

将其添加刚创建的 distance.msg 消息文件，如下图所示：

```
## Generate messages in the 'msg' folder
add_message_files(
  FILES
  distance.msg
)
```

④找到如下图代码片：

```
## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

将其修改为如下图所示：

```
## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

到这里 CMakeLists.txt 配置完成，保存并退出。

（4）查看刚刚创建的 msg 消息，在终端输入 rosmsg show distance，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ rosmsg show distance
[hps_camera/distance]:
uint16 distance_average

dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```

2、创建 ROS 服务 srv

（1）在创建的包中，创建一个服务 srv 目录，存放 srv 文件。在终端输入 cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/，到包目录；输入 mkdir srv，创建 srv 目录；输入 sudo gedit srv/camera.srv，创建 camera.srv 文件，并输入如下代码，如下图所示：

```
string client_node_name
---
string control_cmd
```

注：”string client_node_name”是请求，存储客户端节点的名称发送给服务器，”---”是分隔请求和响应，”string control_cmd”是响应，存储服务器发送给客户端的控制命令。

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt  CMakeLists.txt~  include  msg  package.xml  package.xml~  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ mkdir srv
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt  CMakeLists.txt~  include  msg  package.xml  package.xml~  src  srv
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo gedit srv/camera.srv
```

（2）在终端输入 sudo gedit CMakeLists.txt，对 CMakeLists.txt 进行配置，找到相应位置进行修改。找到如下图代码段：

```
## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )
```

将其添加刚刚创建的 camera.srv 服务文件，如下图所示：

```
## Generate services in the 'srv' folder
add_service_files(
  FILES
  carema.srv
)
```

到这里 CMakeLists.txt 配置完成，保存并退出。

（3）查看刚刚创建的 srv 服务，在终端输入 rossrv show camera，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera/srv$ rossrv show camera
[hps_camera/camera]:
string client_node_name
---
string control_cmd

dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera/srv$
```

3、重新构建 ROS 包

在工作目录中，输入 catkin_make，进行构建，将会看到如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ cd ../..
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ catkin_make
```

```
#### Running command: "make -j1 -l1" in "/home/dote/HPS3D_SDK_ROS_Demo/build"
####
Scanning dependencies of target _hps_camera_generate_messages_check_deps_distanc
[  0%] Built target _hps_camera_generate_messages_check_deps_distance
Scanning dependencies of target _hps_camera_generate_messages_check_deps_camera
[  0%] Built target _hps_camera_generate_messages_check_deps_camera
Scanning dependencies of target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_py
Scanning dependencies of target hps_camera_generate_messages_py
[ 12%] Generating Python from MSG hps_camera/distance
[ 25%] Generating Python code from SRV hps_camera/camera
[ 37%] Generating Python msg __init__.py for hps_camera
[ 50%] Generating Python srv __init__.py for hps_camera
[ 50%] Built target hps_camera_generate_messages_py
Scanning dependencies of target std_msgs_generate_messages_lisp
[ 50%] Built target std_msgs_generate_messages_lisp
Scanning dependencies of target hps_camera_generate_messages_lisp
[ 62%] Generating Lisp code from hps_camera/distance.msg
[ 75%] Generating Lisp code from hps_camera/camera.srv
[ 75%] Built target hps_camera_generate_messages_lisp
Scanning dependencies of target std_msgs_generate_messages_cpp
[ 75%] Built target std_msgs_generate_messages_cpp
Scanning dependencies of target hps_camera_generate_messages_cpp
[ 87%] Generating C++ code from hps_camera/distance.msg
[100%] Generating C++ code from hps_camera/camera.srv
[100%] Built target hps_camera_generate_messages_cpp
Scanning dependencies of target hps_camera_generate_messages
[100%] Built target hps_camera_generate_messages
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$
```

2.4.5 创建 ROS 的深度相机客户端节点和服务器

在这部分会提供例程，用户可根据需要自行修改或编写自己的程序，本文档给出的深度相机客户端节点和服务器的例程的流程是：

（1）客户端节点，配置好用户需采集的数据（或通过服务器发送的命令进行配置）；

（2）在客户端登陆后，连接可选设备文件，连接成功之后，向服务器发送自身的客户端名称（自定义名称）；

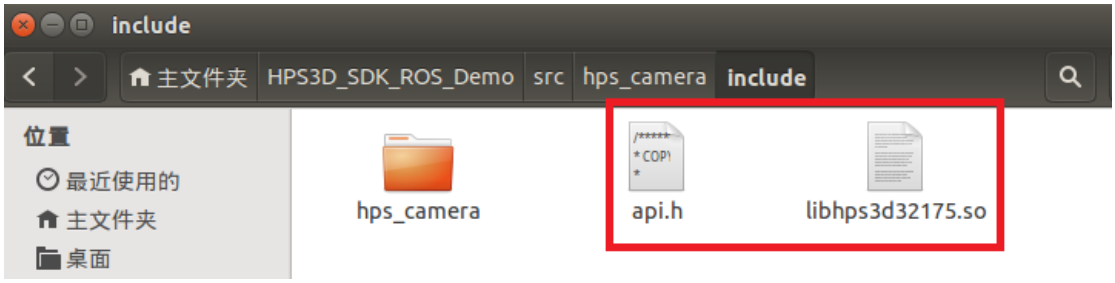
（3）在服务器接收到深度相机客户端发送的消息（客户端名称）之后，进行名称判断，是不是深度相机客户端，若是则向客户端发送开始命令（自定义命令），否则继续等待客户端连接，发送的消息；

（4）在客户端接收到服务器发送过来的命令时，判断是什么命令，若是开始命令，则开使采集数据，并向服务器发布消息；

（5）服务器接收客户端节点发送的消息，可进行进一步匹配，设置等。

1、将 api.h 和.so 文件集成进工程

（1）将 api.h 和 lib*.so 复制进工作空间中包的 include 目录下，如下图所示：



（2）将 lib*.so 复制到/usr/local/lib 中，在包目录中的终端输入 sudo mv include/libhps3d32175.so /usr/local/lib/，拷贝完成后，输入 sudo ldconfig，进行加载配置，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt   include  package.xml   src
CMakeLists.txt~  msg      package.xml~  srv
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo mv include/libhps3d32175.s
o /usr/local/lib/
[sudo] password for dote:
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo ldconfig
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```

（3）在包目录中的终端输入 sudo gedit CMakeLists.txt，配置 CMakeLists.txt 文件，找到如下图代码片：

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

将其修改为如下图所示：

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

到这里 CMakeLists.txt 配置完成，保存并退出。

2、创建 ROS 的深度相机客户端节点和服务器

（1）在包目录中的 src 目录中，创建 ros_camera_client.cpp 和 ros_camera_server.cpp，在包目录中的终端输入 sudo touch src/ros_camera_client.cpp src/ros_camera_server.cpp，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ ls
CMakeLists.txt   include  package.xml   src
CMakeLists.txt~  msg      package.xml~  srv
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo touch src/ros_camera_clien
t.cpp src/ros_camera_server.cpp
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo ls src/
ros_camera_client.cpp  ros_camera_server.cpp
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```

（4）在包目录中的终端输入 sudo gedit CMakeLists.txt，配置 CMakeLists.txt 文件，添加如下代码片到 CMakeLists.txt 文件中，如下图所示：

```
add_executable(ros_camera_client src/ros_camera_client.cpp)
target_link_libraries(ros_camera_client ${catkinhps_camera_LIBRARIES} hps3d32175)
add_dependencies(ros_camera_client hps_camera_generate_messages_cpp)

add_executable(ros_camera_server src/ros_camera_server.cpp)
target_link_libraries(ros_camera_server ${catkin_LIBRARIES})
add_dependencies(ros_camera_server hps_camera_generate_messages_cpp)
```

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

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/hps_camera.cpp
# )

add_executable(ros_camera_client src/ros_camera_client.cpp)
target_link_libraries(ros_camera_client ${catkin_LIBRARIES} hps3d32175)
add_dependencies(ros_camera_client hps_camera_generate_messages_cpp)

add_executable(ros_camera_server src/ros_camera_server.cpp)
target_link_libraries(ros_camera_server ${catkin_LIBRARIES})
add_dependencies(ros_camera_server hps_camera_generate_messages_cpp)
```

3、编写 ROS 的深度相机客户端节点和服务器

（1）编写服务器程序

①添加头文件，代码如下：

```
#include "ros/ros.h"//ros
#include "hps_camera/distance.h"//msg
#include "hps_camera/camera.h"//srv
```

②在主函数中进行 ros 初始化，节点创建，话题创建等，代码如下：

```
ros::init(argc, argv, "ros_camera_server");
ros::NodeHandle n;
ros::ServiceServer service = n.advertiseService("client_login", send_cmd);
ros::Subscriber sub = n.subscribe("camera", 1000, chatterCallback);
ros::spin();
```

③服务函数的编写，代码如下：

```
bool send_cmd(hps_camera::camera::Request  &req, hps_camera::camera::Response
&res)
{
    std::stringstream scmd;
    printf("client_name: %s\n", req.client_node_name.c_str() );
    if( strcmp(req.client_node_name.c_str(), "camera_client" ) == 0 )
    {
        scmd<< "start";
        res.control_cmd = scmd.str();
```

```
        printf("send_cmd: %s\n", res.control_cmd.c_str() );
    }
    return true;
}
```

④订阅话题的回调函数，代码如下：

```
void chatterCallback(const hps_camera::distance& msg)
{
    printf("distance_average = %d\n", msg.distance_average);
}
```

（2）编写深度相机客户端节点程序

在客户端节点中，深度相机 api 接口的使用与本文档的“2.1.3 在用户项目中使用 SDK”配置一样，详情请参照本文档的“2.1.3 在用户项目中使用 SDK”的内容或给出的示例代码。

①添加头文件，代码如下：

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "hps_camera/distance.h"//msg
#include "hps_camera/camera.h"//srv
#include "../include/api.h"
#include <sstream>
```

②在主函数中，ros 初始化，节点创建，话题创建以及深度相机 api 接口配置，代码如下：

```
ros::Publisher camera_pub;//全局变量，因为在观察者回调函数需要使用
int main(int argc, char **argv)
{
    ros::init(argc, argv, "ros_camera_client");
    ros::NodeHandle n;
    ......
    std::stringstream sclient_name;
    ros::ServiceClient client =
n.serviceClient<hps_camera::camera>("client_login");
    hps_camera::camera srv;
    sclient_name<<"camera_client";
    printf("send name = %s\n",sclient_name.str().c_str());
    srv.request.client_node_name = sclient_name.str();
    camera_pub = n.advertise<hps_camera::distance>("camera", 1000);
    ......
    if (client.call(srv))
    {
        while(ros::ok())
        {
            printf("rev cmd = %s\n",srv.response.control_cmd.c_str());
            if( strcmp(srv.response.control_cmd.c_str(), "start" ) == 0 )
            {
                break;
            }
```

```
        }
    }
    else
    {
        break;
    }
    ......
    while(1)
    {
        ros::spinOnce();
    }
    return 0;
}
```

③在本文档的“2.1.3 在用户项目中使用 SDK”中，第 5 点（2）异步测量的②编写观察者回调函数，需要在测量得到数值时，将数值赋值给 msg 消息结构体，并发布。代码如下：

```
/*观察者回调函数*/
void *User_Func(HPS3D_HandleTypeDef *handle, AsyncIObserver_t *event)
{
    hps_camera::distance msg;
    if(event->AsyncEvent == ISubject_Event_DataRecvd)
    {
        switch(event->RetPacketType)
        {
            case SIMPLE_ROI_PACKET:
                printf("distance = %d  event->RetPacketType = %d\n",
event->MeasureData.simple_roi_data[0].distance_average,event->RetPacketType);
                break;
            case FULL_ROI_PACKET:
                msg.distance_average =
event->MeasureData.full_roi_data[0].distance_average;
                printf("distance = %d\n",msg.distance_average);
                camera_pub.publish(msg);
                break;
            case FULL_DEPTH_PACKET:
                printf("distance = %d  event->RetPacketType = %d\n",
event->MeasureData.full_depth_data->distance_average,event->RetPacketType);
                break;
            case SIMPLE_DEPTH_PACKET:
                printf("distance = %d  event->RetPacketType = %d\n",
event->MeasureData.simple_depth_data->distance_average,event->RetPacketType);
                break;
            case NULL_PACKET:
                /*返回数据包类型为空*/
                break;
```

```
            default:
                printf("system error!\n");
                break;
        }
        event->RetPacketType  = NULL_PACKET;
    }
    return 0;
}
```

4、测试 ROS 的深度相机客户端节点和服务器

（1）在工作空间目录的终端输入 catkin_make，执行编译链接，在/devel/lib/hps_camera 文件夹下可以看到执行文件，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ cd devel/lib/hps_camera/
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/devel/lib/hps_camera$ ls
ros_camera_client  ros_camera_server
dote@ubuntu:~/HPS3D_SDK_ROS_Demo/devel/lib/hps_camera$
```

（2）在终端输入 cd /dev，查看设备文件，输入 ll ttyACM*查看设备文件的详细信息，找到 ttyACM*设备，输入 sudo chmod 777 ttyACM0，修改设备文件权限，如下图所示：

```
dote@ubuntu:~$ cd /dev/
dote@ubuntu:/dev$ ll ttyACM*
crw-rw---- 1 root dialout 166, 0 12月  10 20:40 ttyACM0
dote@ubuntu:/dev$ sudo chmod 777 ttyACM0
[sudo] password for dote:
dote@ubuntu:/dev$ ll ttyACM*
crwxrwxrwx 1 root dialout 166, 0 12月  10 20:40 ttyACM0
dote@ubuntu:/dev$
```

（3）新打开一个终端输入 roscore，运行 ros 总线，如下图所示：

```
dote@ubuntu:~$ roscore
... logging to /home/dote/.ros/log/5f389700-fcf3-11e8-9823-000c299617c1/roslaunch-ubuntu-14026.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:37024/
ros_comm version 1.11.21
```

（4）新打开两个终端，都输入 cd ~/HPS3D_SDK_ROS_Demo/，进入到工作空间，都输入 source devel/setup.bash，生效工作空间。

①在其中一个终端运行服务器，输入 hps_camera ros_camera_server，启动服务器，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_server
waiting client login
```

②在另一个终端运行客户端节点，输入 hps_camera ros_camera_client，启动客户端，如下图所示：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_client
send name = camera_client
当前可连接的设备(请选择)：
0: /dev/ttyACM0
请输入对应的序号：
```

③在客户端终端选择可连接设备，输入 0，得到如下结果：

客户端终端：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_client
send name = camera_client
当前可连接的设备(请选择)：
0: /dev/ttyACM0
请输入对应的序号：
0
初始化成功
rev cmd = start
distance = 1806
distance = 1797
distance = 1817
distance = 1798
distance = 1802
distance = 1793
```

服务器终端：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_server
waiting client login
client_name: camera_client
send_cmd: start
distance_average = 1806
distance_average = 1797
distance_average = 1817
distance_average = 1798
distance_average = 1802
distance_average = 1793
```

三、命令函数接口

命令函数接口是 HPS3D 深度相机中基本的命令；每个命令函数接口包含命令数据包打包，以及对命令返回包解析。若 api.h 中集成的函数接口不够用户使用时，用户可根据命令函数接口自行组合配置新的接口。（注：在发送完命令后，需要重新设置运行模式！！！）

3.1 设置运行模式

3.1.1 示例代码

```
HPS3D_HandleTypeDef handle;
int main(int argc, char *argv[])
{
    ......
    handle.DeviceName = "/dev/ttyACM0"; // handle.DeviceName = "\\\\.\\COM13"
    /*HPS3D 连接*/
    ret = HPS3D_Connect(&handle);
```

```
    /*HPS3D 配置初始化，获得 handle.DeviceAddr*/
    ret = HPS3D_ConfigInit(&handle);
    /*设置运行模式为连续模式*/
    handle.RunMode = RUN_CONTINUOUS;
    ret = HPS3D_SetRunMode(&handle);
    ......
}
```

3.2 获取/设置设备地址

3.2.1 示例代码

```
/*获取设备地址*/
ret = HPS3D_GetDevAddr(&handle);
printf("1handle.DeviceAddr = %#x\n",handle.DeviceAddr);

/*设置设备地址*/
ret = HPS3D_SetDevAddr(&handle, 0x01);
ret = HPS3D_GetDevAddr(&handle);
printf("2handle.DeviceAddr = %#x\n",handle.DeviceAddr);
```

3.2.2 运行结果

```
1handle.DeviceAddr = 0
2handle.DeviceAddr = 0x1
```

3.3 获取设备版本信息

3.3.1 示例代码

```
/*获取设备版本信息*/
Version_t version_t;
HPS3D_GetDeviceVersion(&handle, &version_t);
printf("version_t.year = %d\n", version_t.year);
printf("version_t.month = %d\n", version_t.month);
printf("version_t.day = %d\n", version_t.day);
printf("version_t.major = %d\n", version_t.major);
printf("version_t.minor = %d\n", version_t.minor);
printf("version_t.rev = %d\n", version_t.rev);
```

3.3.2 运行结果

```
version_t.year = 18
version_t.month = 11
version_t.day = 15
version_t.major = 1
version_t.minor = 7
version_t.rev = 9
```

3.4 获取/设置数据包类型

3.4.1 示例代码

```
/*获取数据包类型*/
HPS3D_GetPacketType(&handle);
printf("1handle.PacketType = %d\n", handle.PacketType);//PACKET_FULL = 0

/*设置数据包类型*/
handle.PacketType = PACKET_SIMPLE;/*设置成简单数据包*/
HPS3D_SetPacketType(&handle);
HPS3D_GetPacketType(&handle);
printf("2handle.PacketType = %d\n", handle.PacketType);
```

3.4.2 运行结果

```
1handle.PacketType = 0
2handle.PacketType = 1
```

3.5 保存/清除/恢复出厂设置

3.5.1 示例代码

```
HPS3D_ProfileSaveToCurrent(&handle);
HPS3D_ProfileClearCurrent(&handle);
HPS3D_ProfileRestoreFactory(&handle);
```

3.6 获得传输类型

3.6.1 示例代码

```
/*获取传输类型*/
TransportTypeDef transport_type;
HPS3D_GetTransportType(&handle, &transport_type);
printf("transport_type = %d\n", transport_type);//TRANSPORT_USB = 0
```

3.6.2 运行结果

```
transport_type = 0
```

3.7 选择 ROI 组/获得当前 ROI 组 ID

3.7.1 示例代码

```
uint8_t group_id = 0;
/*选择组 ID 为 3*/
HPS3D_SelectROIGroup(&handle, 3);

/*获得组 ID*/
```

```
HPS3D_GetROIGroupID(&handle, &group_id);
printf("group_id = %d\n",group_id);
```

3.7.2 运行结果

```
group_id = 3
```

3.8 ROI 相关配置

3.8.1 示例代码

```
ROIConfTypeDef roi_conf1, roi_conf2;
HysteresisSingleConfTypeDef hysteresis_conf1, hysteresis_conf2;

/*选择组 0，roi_id = 0，threshold_id = 2，GPIO 警报，ROI 区域的距离最小值作为参考值
*/
/*选择组 0，roi_id = 2，threshold_id = 1，GPIO 警报关闭，ROI 区域的有效幅值平均值作
为参考值*/
/*设置 ROI 警报类型*/
HPS3D_SetROIAlarmType(&handle, 0, 2, ROI_ALARM_GPIO);
HPS3D_SetROIAlarmType(&handle, 2, 1, ROI_ALARM_DISABLE);

/*设置 ROI 的参考值类型*/
HPS3D_SetROIReferenceType(&handle, 0, 2, ROI_REF_DIST_MIN);
HPS3D_SetROIReferenceType(&handle, 2, 1, ROI_REF_VAILD_AMPLITUDE);

/*设定 ROI 区域*/
roi_conf1.roi_id = 0;
roi_conf1.left_top_x = 10;
roi_conf1.left_top_y = 10;
roi_conf1.right_bottom_x = 30;
roi_conf1.right_bottom_y = 20;
HPS3D_SetROIRegion(&handle, roi_conf1);

roi_conf2.roi_id = 2;
roi_conf2.left_top_x = 40;
roi_conf2.left_top_y = 30;
roi_conf2.right_bottom_x = 80;
roi_conf2.right_bottom_y = 50;
HPS3D_SetROIRegion(&handle, roi_conf2);

/*设置 ROI 使能*/
HPS3D_SetROIEnable(&handle, 0, true);
HPS3D_SetROIEnable(&handle, 2, true);

/*设置 ROI 阈值使能*/
```

```
HPS3D_SetROIThresholdEnable(&handle, 0, 2, true);
HPS3D_SetROIThresholdEnable(&handle, 2, 1, true);

/*设置 ROI 阈值配置*/
hysteresis_conf1.threshold_value = 20;
hysteresis_conf1.hysteresis = 100;
hysteresis_conf1.positive = true;
HPS3D_SetROIThresholdConf(&handle, 0, 2, 60, hysteresis_conf1);

hysteresis_conf2.threshold_value = 30;
hysteresis_conf2.hysteresis = 200;
hysteresis_conf2.positive = false;
HPS3D_SetROIThresholdConf(&handle, 2, 1, 70, hysteresis_conf2);

/*获取指定的 ROI 配置*/
HPS3D_GetROIConfById(&handle, 0, &roi_conf1);
HPS3D_GetROIConfById(&handle, 2, &roi_conf2);

/*打印 roi_id = 0, threshold_id = 2*/
printf("roi_conf1.roi_id = %d\n", roi_conf1.roi_id);
printf("roi_conf1.enable = %d\n", roi_conf1.enable);
printf("roi_conf1.left_top_x = %d\n", roi_conf1.left_top_x);
printf("roi_conf1.left_top_y = %d\n", roi_conf1.left_top_y);
printf("roi_conf1.right_bottom_x = %d\n", roi_conf1.right_bottom_x);
printf("roi_conf1.right_bottom_y = %d\n", roi_conf1.right_bottom_y);
printf("roi_conf1.alarm_type[2] = %d\n", roi_conf1.alarm_type[2]);
printf("roi_conf1.roi_conf1.ref_type[2] = %d\n", roi_conf1.ref_type[2]);
printf("roi_conf1.roi_conf1.pixel_number_threshold[2] = %d\n",
roi_conf1.pixel_number_threshold[2]);
printf("roi_conf1.hysteresis_conf[2].enable = %d\n",
roi_conf1.hysteresis_conf[2].enable);
printf("roi_conf1.hysteresis_conf[2].threshold_value = %d\n",
roi_conf1.hysteresis_conf[2].threshold_value);
printf("roi_conf1.hysteresis_conf[2].positive = %d\n",
roi_conf1.hysteresis_conf[2].positive);
printf("roi_conf1.hysteresis_conf[2].hysteresis = %d\n",
roi_conf1.hysteresis_conf[2].hysteresis);

printf("\n");

/*打印 roi_id = 2, threshold_id = 1*/
printf("roi_conf2.roi_id = %d\n", roi_conf2.roi_id);
printf("roi_conf2.enable = %d\n", roi_conf2.enable);
printf("roi_conf2.left_top_x = %d\n", roi_conf2.left_top_x);
```

```
printf("roi_conf2.left_top_y = %d\n", roi_conf2.left_top_y);
printf("roi_conf2.right_bottom_x = %d\n", roi_conf2.right_bottom_x);
printf("roi_conf2.right_bottom_y = %d\n", roi_conf2.right_bottom_y);
printf("roi_conf2.alarm_type[1] = %d\n", roi_conf2.alarm_type[1]);
printf("roi_conf2.roi_conf1.ref_type[1] = %d\n", roi_conf2.ref_type[1]);
printf("roi_conf2.roi_conf1.pixel_number_threshold[1] = %d\n",
roi_conf2.pixel_number_threshold[1]);
printf("roi_conf2.hysteresis_conf[1].enable = %d\n",
roi_conf2.hysteresis_conf[1].enable);
printf("roi_conf2.hysteresis_conf[1].threshold_value = %d\n",
roi_conf2.hysteresis_conf[1].threshold_value);
printf("roi_conf2.hysteresis_conf[1].positive = %d\n",
roi_conf2.hysteresis_conf[1].positive);
printf("roi_conf2.hysteresis_conf[1].hysteresis = %d\n",
roi_conf2.hysteresis_conf[1].hysteresis);
```

3.8.2 运行结果

```
roi_conf1.roi_id = 0
roi_conf1.enable = 1
roi_conf1.left_top_x = 10
roi_conf1.left_top_y = 10
roi_conf1.right_bottom_x = 30
roi_conf1.right_bottom_y = 20
roi_conf1.alarm_type[2] = 1
roi_conf1.roi_conf1.ref_type[2] = 2
roi_conf1.roi_conf1.pixel_number_threshold[2] = 60
roi_conf1.hysteresis_conf[2].enable = 1
roi_conf1.hysteresis_conf[2].threshold_value = 20
roi_conf1.hysteresis_conf[2].positive = 1
roi_conf1.hysteresis_conf[2].hysteresis = 100

roi_conf2.roi_id = 2
roi_conf2.enable = 1
roi_conf2.left_top_x = 40
roi_conf2.left_top_y = 30
roi_conf2.right_bottom_x = 80
roi_conf2.right_bottom_y = 50
roi_conf2.alarm_type[1] = 0
roi_conf2.roi_conf1.ref_type[1] = 6
roi_conf2.hysteresis_conf[1].threshold_id = 2
roi_conf2.hysteresis_conf[1].enable = 1
roi_conf2.hysteresis_conf[1].threshold_value = 30
roi_conf2.hysteresis_conf[1].positive = 0
roi_conf2.hysteresis_conf[1].hysteresis = 200
```

3.9 获取当前设备支持的 ROI 数量和阈值数量

3.9.1 示例代码

```
/*获取当前设备支持的 ROI 数量和阈值数量*/
uint8_t roi_number, threshold_number;
HPS3D_GetNumberOfROI(&handle, &roi_number, &threshold_number);

printf("roi_number = %d\n", roi_number);
printf("threshold_number = %d\n", threshold_number);
```

3.9.2 运行结果

```
roi_number = 20
threshold number = 3
```

3.10 设置/获取输出/输入配置

3.10.1 示例代码

```
/*获取 GPOUT 的配置*/
GPIOOutConfTypeDef gpio_out_conf;
gpio_out_conf.gpio = GPOUT_1;/*指定 GPIO 口，必须是输出口*/
HPS3D_GetGPIOOutConf(&handle, &gpio_out_conf);
printf("1gpio_out_conf.function = %d\n",gpio_out_conf.function);
printf("1gpio_out_conf.polarity = %d\n",gpio_out_conf.polarity);

/*获取 GPIN 的配置*/
GPIOInConfTypeDef gpio_in_conf;
gpio_in_conf.gpio = GPIN_1;/*指定 GPIO 口，必须是输入口*/
HPS3D_GetGPIOInConf(&handle, &gpio_in_conf);
printf("1gpio_in_conf.function = %d\n",gpio_in_conf.function);
printf("1gpio_in_conf.polarity = %d\n",gpio_in_conf.polarity);

/*设置 GPOUT 的配置*/
gpio_out_conf.gpio = GPOUT_1;/*指定 GPIO 口，必须是输出口*/
gpio_out_conf.function = 1;
gpio_out_conf.polarity = 1;
HPS3D_SetGPIOOut(&handle, gpio_out_conf);
HPS3D_GetGPIOOutConf(&handle, &gpio_out_conf);
printf("2gpio_out_conf.function = %d\n",gpio_out_conf.function);
printf("2gpio_out_conf.polarity = %d\n",gpio_out_conf.polarity);

/*设置 GPIN 的配置*/
gpio_in_conf.gpio = GPIN_1;/*指定 GPIO 口，必须是输入口*/
gpio_in_conf.function = 0;/*注：这里功能如果设置为 1,则不再受命令控制，直接受 IO 口
控制！！！*/
gpio_in_conf.polarity = 1;
HPS3D_SetGPIOIn(&handle, gpio_in_conf);
HPS3D_GetGPIOInConf(&handle, &gpio_in_conf);
printf("2gpio_in_conf.function = %d\n",gpio_in_conf.function);
printf("2gpio_in_conf.polarity = %d\n",gpio_in_conf.polarity);
```

3.10.2 运行结果

```
1gpio_out_conf.function = 0
1gpio_out_conf.polarity = 0
1gpio_in_conf.function = 0
1gpio_in_conf.polarity = 0
2gpio_out_conf.function = 1
2gpio_out_conf.polarity = 1
2gpio_in_conf.function = 0
2gpio_in_conf.polarity = 1
```

3.11 设置 HDR 模式

3.11.1 示例代码

```
/*获得 HDR 模式*/
HDRConf hdr_conf;
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("1hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);

/*设置成 AUTO_HDR*/
HPS3D_SetHDRMode(&handle, AUTO_HDR);/*AUTO_HDR = 1*/
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("2hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);
```

3.11.2 运行结果

```
1hdr_conf.hdr_mode = 3
2hdr_conf.hdr_mode = 1
```

3.12 设置/获取 HDR 配置

3.12.1 示例代码

```
/*获取距离滤波器配置*/
HDRConf hdr_conf, set_conf;
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("1hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);//1
printf("1hdr_conf.hdr_disable_integration_time
= %d\n",hdr_conf.hdr_disable_integration_time);//7500
printf("1hdr_conf.qualtity_overexposed = %f\n",hdr_conf.qualtity_overexposed);//500
printf("1hdr_conf.qualtity_overexposed_serious
= %f\n",hdr_conf.qualtity_overexposed_serious);//800
printf("1hdr_conf.qualtity_weak = %f\n",hdr_conf.qualtity_weak);//90
printf("1hdr_conf.qualtity_weak_serious
= %f\n",hdr_conf.qualtity_weak_serious);//50
printf("1hdr_conf.simple_hdr_max_integration
= %d\n",hdr_conf.simple_hdr_max_integration);//2000
printf("1hdr_conf.simple_hdr_min_integration
```

```
= %d\n",hdr_conf.simple_hdr_min_integration);//100
printf("1hdr_conf.super_hdr_frame_number
= %d\n",hdr_conf.super_hdr_frame_number);//4
printf("1hdr_conf.super_hdr_max_integration
= %d\n",hdr_conf.super_hdr_max_integration);//30000

printf("\n");
/*1、设置 HDR,HDR-DISABLE*/
set_conf.hdr_mode = HDR_DISABLE;
set_conf.hdr_disable_integration_time = 1000;
HPS3D_SetHDRConfig(&handle, set_conf);
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("2hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);
printf("2hdr_conf.hdr_disable_integration_time
= %d\n",hdr_conf.hdr_disable_integration_time);

/*2、设置 HDR,AUTO-HDR*/
set_conf.hdr_mode = AUTO_HDR;
set_conf.qualtity_overexposed = 600;/*曝光幅值一定要小于过度曝光赋值，否则设置失败
*/
set_conf.qualtity_overexposed_serious = 900;
set_conf.qualtity_weak = 80;/*弱幅值一定要大于极弱幅值，否则设置失败*/
set_conf.qualtity_weak_serious = 60;
HPS3D_SetHDRConfig(&handle, set_conf);
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("3hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);
printf("3hdr_conf.qualtity_overexposed = %f\n",hdr_conf.qualtity_overexposed);
printf("3hdr_conf.qualtity_overexposed_serious
= %f\n",hdr_conf.qualtity_overexposed_serious);
printf("3hdr_conf.qualtity_weak = %f\n",hdr_conf.qualtity_weak);
printf("3hdr_conf.qualtity_weak_serious = %f\n",hdr_conf.qualtity_weak_serious);

/*3、设置 HDR,SIMPLE-HDR*/
set_conf.hdr_mode = SIMPLE_HDR;
set_conf.simple_hdr_max_integration = 500;/*必须大于最小积分时间，否则设置失败*/
set_conf.simple_hdr_min_integration = 400;
HPS3D_SetHDRConfig(&handle, set_conf);
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("4hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);
printf("4hdr_conf.simple_hdr_max_integration
= %d\n",hdr_conf.simple_hdr_max_integration);
printf("4hdr_conf.simple_hdr_min_integration
= %d\n",hdr_conf.simple_hdr_min_integration);
```

```
/*4、设置 HDR,SUPER-HDR*/
set_conf.hdr_mode = SUPER_HDR;
set_conf.super_hdr_frame_number = 2;
set_conf.super_hdr_max_integration = 15000;
HPS3D_SetHDRConfig(&handle, set_conf);
HPS3D_GetHDRConfig(&handle, &hdr_conf);
printf("5hdr_conf.hdr_mode = %d\n",hdr_conf.hdr_mode);
printf("5hdr_conf.super_hdr_frame_number = %d\n",hdr_conf.super_hdr_frame_number);
printf("5hdr_conf.super_hdr_max_integration
= %d\n",hdr_conf.super_hdr_max_integration);
```

3.12.2 运行结果

```
1hdr_conf.hdr_mode = 1
1hdr_conf.hdr_disable_integration_time = 400
1hdr_conf.qualtity_overexposed = 500.000000
1hdr_conf.qualtity_overexposed_serious = 800.000000
1hdr_conf.qualtity_weak = 90.000000
1hdr_conf.qualtity_weak_serious = 50.000000
1hdr_conf.simple_hdr_max_integration = 2000
1hdr_conf.simple_hdr_min_integration = 100
1hdr_conf.super_hdr_frame_number = 4
1hdr_conf.super_hdr_max_integration = 30000

2hdr_conf.hdr_mode = 0
2hdr_conf.hdr_disable_integration_time = 1000
3hdr_conf.hdr_mode = 1
3hdr_conf.qualtity_overexposed = 600.000000
3hdr_conf.qualtity_overexposed_serious = 900.000000
3hdr_conf.qualtity_weak = 80.000000
3hdr_conf.qualtity_weak_serious = 60.000000
4hdr_conf.hdr_mode = 3
4hdr_conf.simple_hdr_max_integration = 500
4hdr_conf.simple_hdr_min_integration = 400
5hdr_conf.hdr_mode = 2
5hdr_conf.super_hdr_frame_number = 2
5hdr_conf.super_hdr_max_integration = 15000
```

3.13 设置/获取距离滤波器配置

3.13.1 示例代码

```
/*获取距离滤波器配置*/
DistanceFilterConfTypeDef distance_filter_conf, set_conf;
HPS3D_GetDistanceFilterConf(&handle, &distance_filter_conf);
printf("1distance_filter_conf.filter_type
= %d\n",distance_filter_conf.filter_type);//0
printf("1distance_filter_conf.kalman_K = %f\n",distance_filter_conf.kalman_K);//0.1
printf("1distance_filter_conf.kalman_threshold
= %d\n",distance_filter_conf.kalman_threshold);//100
printf("1distance_filter_conf.num_check = %d\n",distance_filter_conf.num_check);//2

/*设置距离滤波器类型*/
HPS3D_SetDistanceFilterType(&handle, DISTANCE_FILTER_SIMPLE_KALMAN);/*简单的卡尔曼
滤波器*/
```

```
HPS3D_GetDistanceFilterConf(&handle, &distance_filter_conf);
printf("2distance_filter_conf.filter_type
= %d\n",distance_filter_conf.filter_type);

/*配置距离滤波器*/
set_conf.kalman_K = 0.3;
set_conf.kalman_threshold = 200;
set_conf.num_check = 3;
HPS3D_SetSimpleKalman(&handle, set_conf);
HPS3D_GetDistanceFilterConf(&handle, &distance_filter_conf);
printf("2distance_filter_conf.kalman_K = %f\n",distance_filter_conf.kalman_K);
printf("2distance_filter_conf.kalman_threshold
= %d\n",distance_filter_conf.kalman_threshold);
printf("2distance_filter_conf.num_check = %d\n",distance_filter_conf.num_check);
```

3.13.2 运行结果

```
1distance_filter_conf.filter_type = 0
1distance_filter_conf.kalman_K = 0.100000
1distance_filter_conf.kalman_threshold = 100
1distance_filter_conf.num_check = 2
2distance_filter_conf.filter_type = 1
2distance_filter_conf.kalman_K = 0.300000
2distance_filter_conf.kalman_threshold = 200
2distance_filter_conf.num_check = 3
```

3.14 设置/获取平滑滤波器配置

3.14.1 示例代码

```
/*获取平滑滤波器的配置*/
SmoothFilterConfTypeDef smooth_filter_conf, set_conf;
HPS3D_GetSmoothFilterConf(&handle, &smooth_filter_conf);
printf("1smooth_filter_conf.type
= %d\n",smooth_filter_conf.type);/*SMOOTH_FILTER_DISABLE = 0*/
printf("1smooth_filter_conf.arg1 = %d\n",smooth_filter_conf.arg1);/*0*/

/*设置平滑滤波器*/
set_conf.type = SMOOTH_FILTER_AVERAGE;/*均值滤波器*/
set_conf.arg1 = 200;
HPS3D_SetSmoothFilter(&handle, set_conf);
HPS3D_GetSmoothFilterConf(&handle, &smooth_filter_conf);
printf("2smooth_filter_conf.type = %d\n",smooth_filter_conf.type);
printf("2smooth_filter_conf.arg1 = %d\n",smooth_filter_conf.arg1);
```

3.14.2 运行结果

```
1smooth_filter_conf.type = 0
1smooth_filter_conf.arg1 = 0
2smooth_filter_conf.type = 1
2smooth_filter_conf.arg1 = 200
```

3.15 设定光学参数使能/获取光学参数

3.15.1 示例代码

```
/*获得光学参数*/
OpticalParamConfTypeDef optical_param_conf;
HPS3D_GetOpticalParamConf(&handle, &optical_param_conf);
printf("1optical_param_conf.enable = %d\n",optical_param_conf.enable);
printf("1optical_param_conf.illum_angle_horiz
= %d\n",optical_param_conf.illum_angle_horiz);
printf("1optical_param_conf.illum_angle_vertical
= %d\n",optical_param_conf.illum_angle_vertical);
printf("1optical_param_conf.viewing_angle_horiz
= %d\n",optical_param_conf.viewing_angle_horiz);
printf("1optical_param_conf.viewing_angle_vertical
= %d\n",optical_param_conf.viewing_angle_vertical);

/*光学参数使能关闭设置*/
HPS3D_SetOpticalEnable(&handle, false);
HPS3D_GetOpticalParamConf(&handle, &optical_param_conf);
printf("2optical_param_conf.enable = %d\n",optical_param_conf.enable);
```

3.15.2 运行结果

```
1optical_param_conf.enable = 1
1optical_param_conf.illum_angle_horiz = 82
1optical_param_conf.illum_angle_vertical = 36
1optical_param_conf.viewing_angle_horiz = 76
1optical_param_conf.viewing_angle_vertical = 32
2optical_param_conf.enable = 0
```

3.16 设置/获取距离补偿

3.16.1 示例代码

```
/*获得距离补偿*/
int16_t offset;
HPS3D_GetDistanceOffset(&handle, &offset);
printf("1offset = %d\n",offset);

/*设置距离补偿*/
HPS3D_SetDistanceOffset(&handle, 20);

HPS3D_GetDistanceOffset(&handle, &offset);
printf("2offset = %d\n",offset);
```

3.16.2 运行结果

```
1offset = 0
2offset = 20
```

3.17 设定/获取多机干扰参数

3.17.1 示例代码

```
/*获得多机干扰的配置*/
InterferenceDetectConfTypeDef interference_detect_conf;
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("1interference_detect_conf.enable
= %d\n",interference_detect_conf.enable);//0
printf("1interference_detect_conf.integ_time
= %d\n",interference_detect_conf.integ_time);//250
printf("1interference_detect_conf.amplitude_threshold
= %d\n",interference_detect_conf.amplitude_threshold);//6
printf("1interference_detect_conf.capture_num
= %d\n",interference_detect_conf.capture_num);//2
printf("1interference_detect_conf.number_check
= %d\n",interference_detect_conf.number_check);//1

/*设置多机干扰检测使能*/
HPS3D_SetInterferenceDetectEn(&handle, true);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.enable = %d\n",interference_detect_conf.enable);

/*设置多机干扰检测积分时间*/
HPS3D_SetInterferenceDetectIntegTime(&handle, 200);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.integ_time
= %d\n",interference_detect_conf.integ_time);

/*设置多机干扰检测阈值*/
HPS3D_SetInterferenceDetectAmplitudeThreshold(&handle, 5);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.amplitude_threshold
= %d\n",interference_detect_conf.amplitude_threshold);

/*设置多机干扰检测采样次数*/
HPS3D_SetInterferenceDetectCaptureNumber(&handle, 6);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.capture_num
= %d\n",interference_detect_conf.capture_num);

/*设置多机干扰检测采样次数检查*/
HPS3D_SetInterferenceDetectNumberCheck(&handle, 3);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
```

```
printf("2interference_detect_conf.number_check
= %d\n",interference_detect_conf.number_check);
```

3.17.2 运行结果

```
1interference_detect_conf.enable = 0
1interference_detect_conf.integ_time = 250
1interference_detect_conf.amplitude_threshold = 6
1interference_detect_conf.capture_num = 2
1interference_detect_conf.number_check = 1
2interference_detect_conf.enable = 1
2interference_detect_conf.integ_time = 200
2interference_detect_conf.amplitude_threshold = 5
2interference_detect_conf.capture_num = 6
2interference_detect_conf.number_check = 3
```

3.18 设定/获取安装角度参数

3.18.1 示例代码

```
/*获取安装角度变换参数*/
MountingAngleParamTypeDef mounting_angle_param_conf, set_conf;
HPS3D_GetMountingParamConf(&handle, &mounting_angle_param_conf);
printf("1mounting_angle_param_conf.enable
= %d\n",mounting_angle_param_conf.enable);
printf("1mounting_angle_param_conf.angle_vertical
= %d\n",mounting_angle_param_conf.angle_vertical);

/*设定安装角度变换使能*/
HPS3D_SetMountingAngleEnable(&handle, true);

/*设定安装角度变换参数*/
set_conf.angle_vertical = 50;/*50 度*/
HPS3D_SetMountingAngleParamConf(&handle, set_conf);

HPS3D_GetMountingParamConf(&handle, &mounting_angle_param_conf);
printf("2mounting_angle_param_conf.enable
= %d\n",mounting_angle_param_conf.enable);
printf("2mounting_angle_param_conf.angle_vertical
= %d\n",mounting_angle_param_conf.angle_vertical);
```

3.18.2 运行结果

```
1mounting_angle_param_conf.enable = 0
1mounting_angle_param_conf.angle_vertical = 0
2mounting_angle_param_conf.enable = 1
2mounting_angle_param_conf.angle_vertical = 50
```

四、修订历史纪录

Date	Revision	Description
2018/12/11	1.0.0	初始版本。

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.