HPS-3D640
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 HPS-3D640 系列传感器应用程序接口，支持 Windows 、Linux 、ARM Linux 以及 ROS 平台；SDK 提供的接口包含了 HPS-3D640 的大部分操作指令，请详细阅读该使用手册进行环境配置；

二、 将 SDK 集成到 IDE 中

2.1 Visual Studio 平台下环境配置及集成到 IDE 中

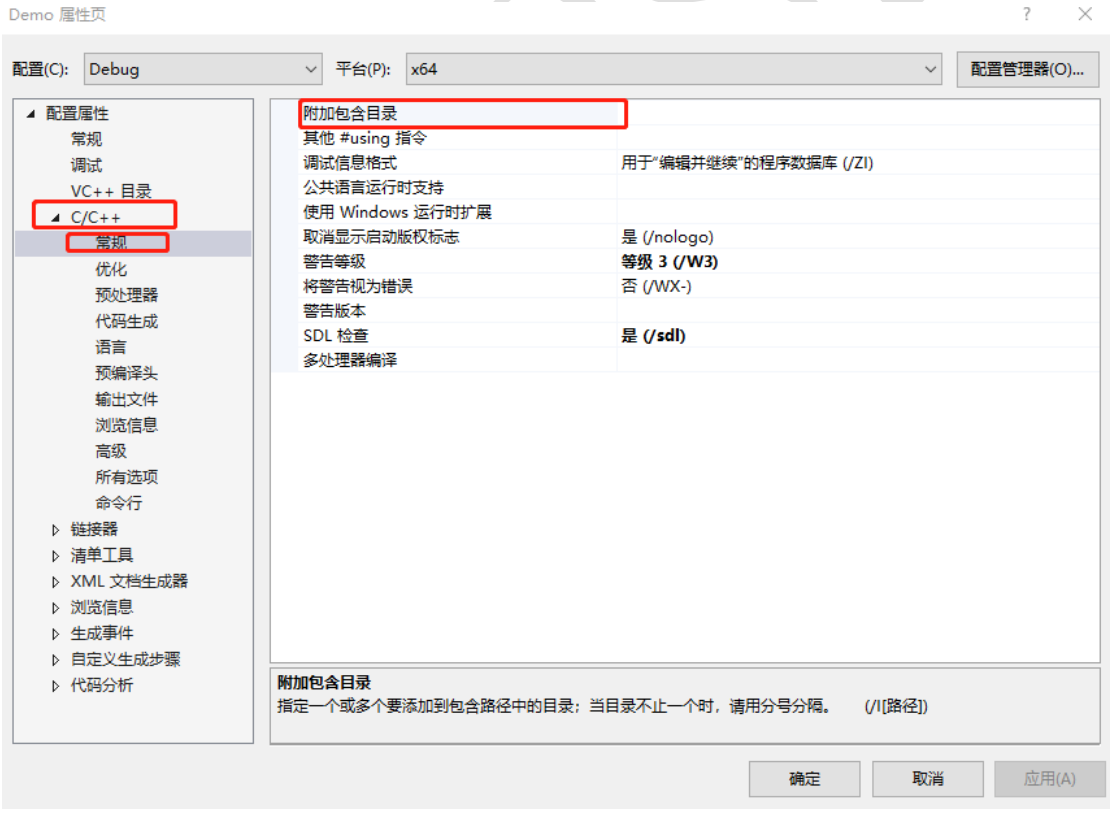
xxx.lib 和 xxx.dll 适合在 Windows 操作系统平台使用，这里以 VS2015 环境为例。

2.1.1 工程环境配置与集成

1、添加头文件包含路径

a、将 HPS3D_IF.h 和 HPS3D_DEFINE.h 拷贝到用户指定路径下。

b、点击项目 - 属性 - C/C++ -常规。在附加包含目录下指定头文件的路径。

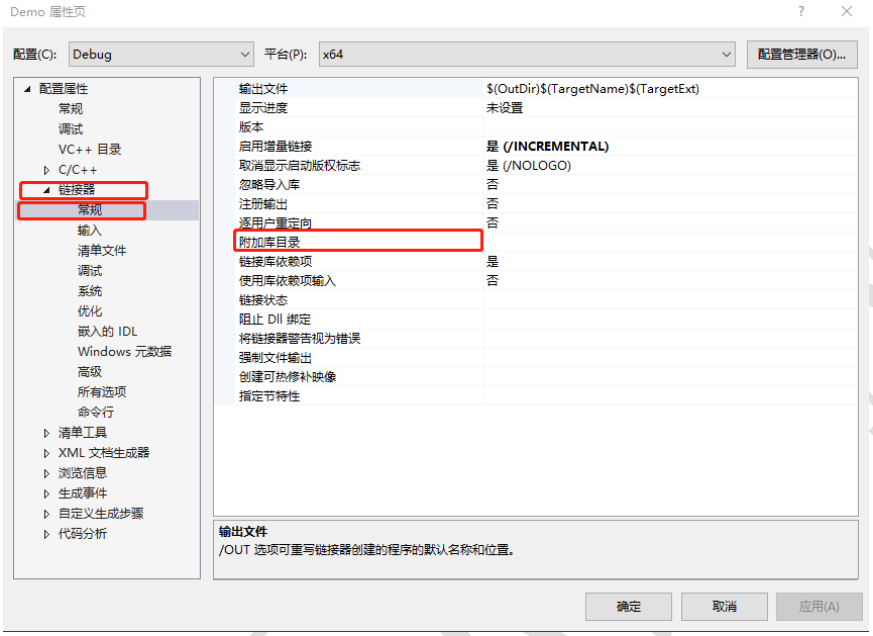


2、添加库引用和库的路径

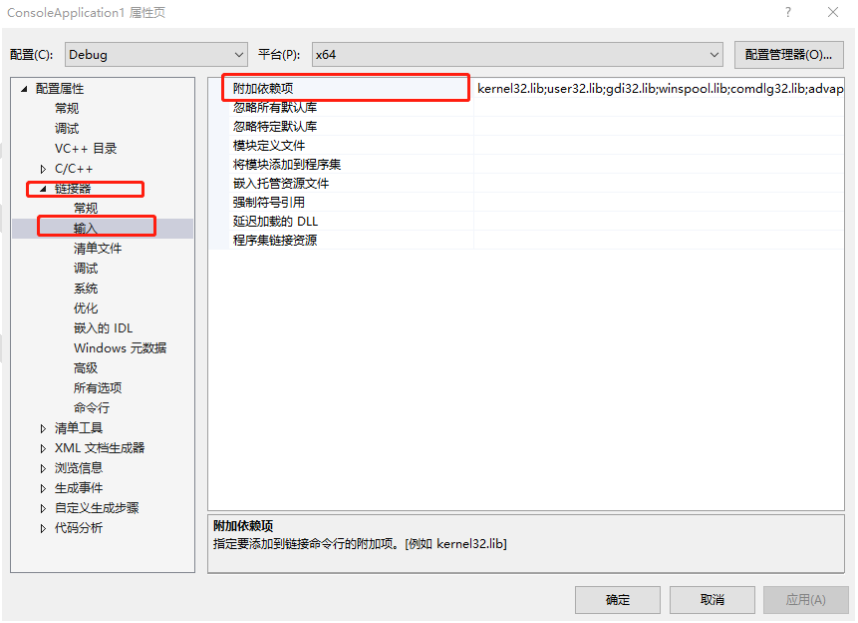
a、根据用户 Visual Studio 的平台，选择 x64 或 x86 目录下的 SDK 文件，这里以 x64 为例。

b、将 HPS3D_SDK_x64 目录下 HPS3D_SDK.dll 拷贝到程序运行的目录下；将 HPS3D_SDK.lib 拷贝到用户指定路径下。

c、点击项目 - 属性 - 链接器 - 常规。在附加库目录中指定 HPS3D_SDK.lib 的路径。



d、点击项目 - 属性 - 链接器 - 输入。在附加依赖选项中填入 HPS3D_SDK.lib。



2.1.2 在用户项目中使用 SDK

以下为获取传感器数据的简单步骤，更具体的请参考 HPS3D_DOME 中的示例代码。

1、参照用户手册中的连接方式，将 3D 传感器与 PC 端连接。

2、包含头文件：（必要）

```
#include "HPS3D_IF.h"
#include "HPS3D_DEFINE.h"
```

3、设置连接参数并连接设备：（必要）

```
/*设备连接之前可将设备 ID 设置为 99，连接成功后将自动重新分配 ID*/
uint8_t device_id = 99;
HPS3D_ConnectByEthernet((char*)"192.168.30.202", 12345, &device_id);
```

4、注册事件回调函数：（必要）

```
/* OutputEventFunc 为事件回调函数，第二个参数为执行回调函数的设备 ID*/
HPS3D_SetOutputCallBack(OutputEventFunc, device_id, NULL);
```

5、配置传感器：（首次使用配置即可）

注：用户可以使用客户端配置，配置完成后点击保存到用户配置表即可（保存到用户配置表中的数据，重新上电不会丢失）

```
    uint8_t ret = 0;
    uint32_t time_us = 0;
    //积分时间根据现场环境和被测物体信号反射强度进行调节（必要）
    ret = HPS3D_SetIntegTime(g_device_id, 200);
    if (ret != RET_OK)
    {
        printf("设置积分时间:.....失败!\n");
        printf("按回车键继续运行!\n");
        getchar();
        return RET_ERROR;
    }
    else
    {
        printf("设置积分时间:.....成功!\n");
    }

    //滤波器配置（可选）
    //1.平滑滤波器配置,根据需要自行选择滤波器类型,以下为关闭
    ret = HPS3D_SetSmoothFilter(g_device_id, SMOOTH_FILTER_DISABLE, 0);
    if (ret != RET_OK)
    {
        printf("设置平滑滤波器:.....失败!\n");
        printf("按回车键继续运行!\n");
        getchar();
```

```
    }
    else
    {
        printf("设置平滑滤波器:.....成功!\n");
    }

    //2.卡尔曼滤波器，根据需要自行选择滤波器参数,以下为关闭
    ret = HPS3D_SetSimpleKalmanFilter(g_device_id, false, NULL, NULL, NULL);
    if (ret != RET_OK)
    {
        printf("设置卡尔曼滤波器:.....失败!\n");
        printf("按回车键继续运行!\n");
        getchar();
    }
    else
    {
        printf("设置卡尔曼滤波器:.....成功!\n");
    }

    //距离偏移（可选），以下默认设置为0
    ret = HPS3D_SetOffset(g_device_id, 0);
    if (ret != RET_OK)
    {
        printf("设置距离偏移:.....失败!\n");
        printf("按回车键继续运行!\n");
        getchar();
    }
    else
    {
        printf("设置距离偏移:.....成功!\n");
    }

    //将设置保存到用户配置表
    ret = HPS3D_ProfileSaveToUser(g_device_id);
    if (ret != RET_OK)
    {
        printf("用户配置表保存:.....失败!\n");
        printf("按回车键继续运行!\n");
        getchar();
    }
    else
    {
        printf("用户配置表保存:.....成功!\n");
    }
```

6、设置设备输出类型：（必要）

```
//选定一种需要输出的数据，配置完成后将通过回调函数 OutputEventFunc 通知对应事件。
//每次只能输出一种类型的数据

//1：简单深度事件 2：完整深度事件 3：幅值事件 4：完整点云事件
//5：ROI点云事件 6：ROI完整深度事件 7：ROI简单深度事件
uint8_t type = 1;
switch (type)
{
case 1: //设置简单深度事件，需要进行以下配置
    if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_SIMPLE) != RET_OK)
    {
        printf("设置简单深度事件:.....失败!\n");
        return RET_ERROR;
    }
    break;

case 2://设置完整深度事件，需要进行以下配置
    if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_FULL) != RET_OK)
    {
        printf("设置完整深度事件:.....失败!\n");
        return RET_ERROR;
    }
    //如果开启了点云转换，则需要关闭
    if (HPS3D_SetPointCloudMode(g_device_id, false, MIRROR_DISABLE) != RET_OK)
    {
        printf("设置完整深度事件:.....失败!\n");
        return RET_ERROR;
    }
    //如果开启了ROI需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
    for (int i = 0; i < 8; i++)
    {
        if (HPS3D_SetROIEnable(g_device_id, i, false) != RET_OK)
        {
            printf("设置完整深度事件:.....失败!\n");
            return RET_ERROR;
        }
    }
    break;

case 3://设置振幅事件，需要进行以下配置
    if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_AMPLITUDE) != RET_OK)
    {
```

```
            printf("设置振幅事件:.....失败!\n");
            return RET_ERROR;
        }
        break;

    case 4://设置完整点云事件，需要进行以下配置
        if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_FULL) != RET_OK)
        {
            printf("设置完整点云事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果没有开启点云转换，则需要开启
        if (HPS3D_SetPointCloudMode(g_device_id, true, MIRROR_DISABLE) != RET_OK)
        {
            printf("设置完整点云事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果开启了ROI需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
        for (int i = 0; i < 8; i++)
        {
            if (HPS3D_SetROIEnable(g_device_id, i, false) != RET_OK)
            {
                printf("设置完整点云事件:.....失败!\n");
                return RET_ERROR;
            }
        }
        break;

    case 5://设置ROI点云事件，需要进行以下配置
        if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_FULL) != RET_OK)
        {
            printf("设置ROI点云事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果没有开启点云转换，则需要开启
        if (HPS3D_SetPointCloudMode(g_device_id, true, MIRROR_DISABLE) != RET_OK)
        {
            printf("设置ROI点云事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
        //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
        if (HPS3D_SetROIEnable(g_device_id, 0, true) != RET_OK)
        {
```

```
            printf("设置ROI点云事件:.....失败!\n");
            return RET_ERROR;
        }
        break;

    case 6://设置ROI完整深度事件，需要进行以下配置
        if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_FULL) != RET_OK)
        {
            printf("设置ROI完整深度事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果开启了点云转换，则需要关闭
        if (HPS3D_SetPointCloudMode(g_device_id, false, MIRROR_DISABLE) != RET_OK)
        {
            printf("设置ROI完整深度事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
        //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
        if (HPS3D_SetROIEnable(g_device_id, 0, true) != RET_OK)
        {
            printf("设置ROI完整深度事件:.....失败!\n");
            return RET_ERROR;
        }
        break;

    case 7://设置ROI简单深度事件，需要进行以下配置
        if (HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_SIMPLE) != RET_OK)
        {
            printf("设置ROI简单深度事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果开启了点云转换，则需要关闭
        if (HPS3D_SetPointCloudMode(g_device_id, false, MIRROR_DISABLE) != RET_OK)
        {
            printf("设置ROI简单深度事件:.....失败!\n");
            return RET_ERROR;
        }
        //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
        //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
        if (HPS3D_SetROIEnable(g_device_id, 0, true) != RET_OK)
        {
            printf("设置ROI简单深度事件:.....失败!\n");
            return RET_ERROR;
```

```
        }
        break;

    default:
        break;
    }
```

7、启动测量数据：

```
    //1.输出一次数据
    HPS3D_SetRunMode(g_device_id, RUN_SINGLE_SHOT);
    printf("执行输出一次数据，按回车键继续运行!\n");
    getchar();
    g_fps = 0;
    //2.连续输出数据
    uint8_t tem = 0;
    HPS3D_SetRunMode(g_device_id, RUN_CONTINUOUS);
    g_start = clock();
    while (1)
    {
        g_end = clock();
        if ((g_end - g_start) >= 1000)
        {
            g_start = clock();
            g_fps = 0;
            tem++;
        }
       //两秒后退出
        if (tem == 2)
        {
            break;
        }
    }
```

8、停止输出数据：（连续输出数据后必须停止）

```
//停止输出数据
HPS3D_SetRunMode(g_device_id, RUN_IDLE);
```

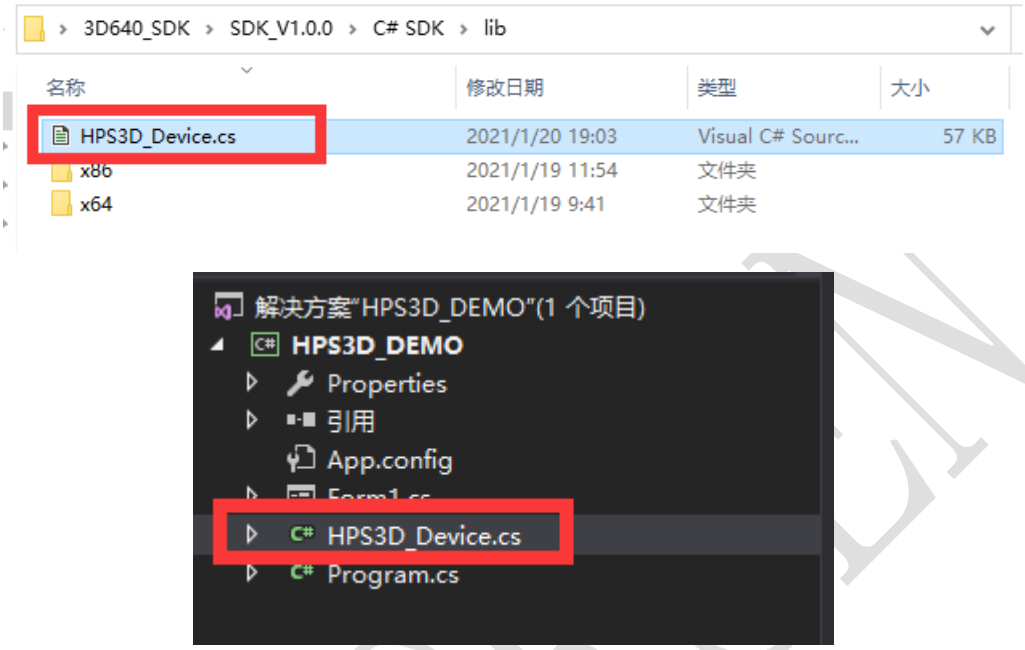
9、运行完成、断开设备连接：

```
//停止输出数据
HPS3D_DisConnect(g_device_id);
```

2.2 C# 开发环境配置及集成到 IDE 中

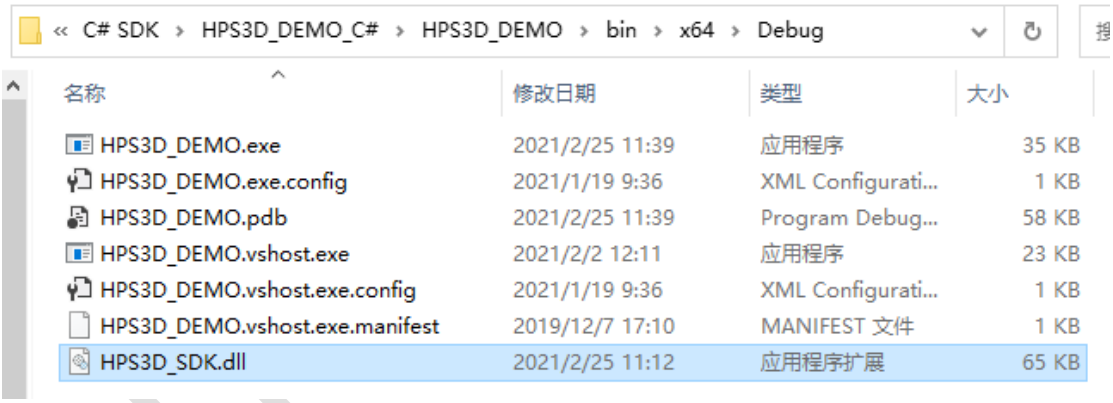
2.2.1 工程环境配置与集成

1、将 lib 目录下的 HPS3D_Device.cs 添加到用户工程中



2、根据用户 Visual Studio 的平台，选择 x64 或 x86 目录下的 dll 文件，这里以 x64 为例。

3、将 x64 目录下的 HPS3D_SDK.dll 拷贝到程序运行的目录下；



2.2.2 在用户项目中使用 SDK

以下为获取传感器数据的简单步骤，更具体的请参考 HPS3D_DOME 中的示例代码。在使用 Demo 中相关函数时，请参照 HPS3D_Device.cs 中对应函数的参数、返回值、和一些枚举类型定义等。

1、 参照用户手册中的接线方式，将激光位移传感器与 PC 端连接。

2、 引用 HPS3D_DeviceLib： （必要）

```
using HPS3D_DeviceLib;
```

3、设置连接参数并连接设备： （必要）

```
byte g_device_id = 99;
//设置 IP 及端口号
HPSLC_Device.ConnectByEthernet("192.168.30.202", 12345, ref g_device_id);
```

4、注册传感器传感器接收回调函数：（必要）

```
//注册传感器传感器接收回调函数
if (HPS3D_Device.SetOutputCallBack(new HPS3D_CALLBACK_OUTPUT(OutputEventFunc),
g_device_id, IntPtr.Zero) != RET_STATUS_ENUM.RET_OK)
{
    HPS3D_Device.DisConnect(g_device_id);
    MessageBox.Show("回调函数注册失败，请重试");
    return;
}
```

5、配置传感器：（首次使用配置即可）

注：用户可以使用客户端配置，配置完成后点击保存到用户配置表即可（保存到用户配置表中的数据，重新上电不会丢失）

```
//积分时间根据现场环境和被测物体信号反射强度进行调节（必要）
if(HPS3D_Device.SetIntegTime(g_device_id, 200) != RET_STATUS_ENUM.RET_OK)
{
    MessageBox.Show("设置积分时间失败!");
    return;
}

//滤波器配置（可选）
//1.平滑滤波器配置,根据需要自行选择滤波器类型,以下为关闭
if (HPS3D_Device.SetSmoothFilter(g_device_id,
SMOOTH_FILTER_TYPE_ENUM.SMOOTH_FILTER_DISABLE, 0) != RET_STATUS_ENUM.RET_OK)
{
    MessageBox.Show("设置平滑滤波器失败!");
    return;
}
//2.卡尔曼滤波器，根据需要自行选择滤波器参数,以下为关闭
if (HPS3D_Device.SetSimpleKalmanFilter(g_device_id, false, 0, 0, 0) !=
RET_STATUS_ENUM.RET_OK)
{
    MessageBox.Show("设置卡尔曼滤波器失败!");
    return;
}

//距离偏移（可选），以下默认设置为0
if (HPS3D_Device.SetOffset(g_device_id, 0) != RET_STATUS_ENUM.RET_OK)
 {
    MessageBox.Show("设置距离偏移失败!");
```

```
    return;
}

//将设置保存到用户配置表,保存后重新上电不会丢失
if (HPS3D_Device.ProfileSaveToUser(g_device_id) != RET_STATUS_ENUM.RET_OK)
{
     MessageBox.Show("用户配置表保存失败!");
     return;
}
    MessageBox.Show("设置成功!");
```

6、设置设备输出类型：（必要）

```
switch (comboBox1.SelectedItem.ToString()) //获取选择的内容
{
    case "简单深度数据"://获取简单深度数据，需要进行以下配置
    if(HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_SIMPLE) != RET_STATUS_ENUM.RET_OK)
    {
        MessageBox.Show("设置简单深度事件失败");
        return;
     }
     //如果开启了ROI需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
     for (byte i = 0; i < 8; i++)
     {
          if (HPS3D_Device.SetROIEnable(g_device_id, i, false) !=
RET_STATUS_ENUM.RET_OK)
          {
              MessageBox.Show("设置简单深度事件失败");
              return;
          }
     }
     break;

     case "完整深度数据"://获取完整深度数据，需要进行以下配置
     if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_FULL) != RET_STATUS_ENUM.RET_OK)
     {
         MessageBox.Show("设置完整深度事件失败");
         return;
      }
      //如果开启了点云转换，则需要关闭
      if (HPS3D_Device.SetPointCloudMode(g_device_id,
false,POINTCLOUD_MIRROR_MODE_ENUM.MIRROR_DISABLE) != RET_STATUS_ENUM.RET_OK)
     {
```

```
                MessageBox.Show("设置完整深度事件失败");
                return;
            }
            //如果开启了ROI需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
            for (byte i = 0; i < 8; i++)
            {
                if (HPS3D_Device.SetROIEnable(g_device_id, i, false) !=
RET_STATUS_ENUM.RET_OK)
                {
                    MessageBox.Show("设置完整深度事件失败");
                    return;
                }
            }
            break;

            case "幅值数据"://获取幅值数据，需要进行以下配置
            if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_AMPLITUDE) != RET_STATUS_ENUM.RET_OK)
            {
                MessageBox.Show("设置幅值事件失败");
                return;
            }
            break;

            case "完整点云数据"://获取完整点云数据，需要进行以下配置
            if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_FULL) != RET_STATUS_ENUM.RET_OK)
            {
                MessageBox.Show("设置完整点云事件失败");
                return;
            }
            //如果没有开启点云转换，则需要开启
            if (HPS3D_Device.SetPointCloudMode(g_device_id, true,
POINTCLOUD_MIRROR_MODE_ENUM.MIRROR_DISABLE) != RET_STATUS_ENUM.RET_OK)
            {
                MessageBox.Show("设置完整点云事件失败");
                return;
            }
            //如果开启了ROI需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
            for (byte i = 0; i < 8; i++)
            {
                if (HPS3D_Device.SetROIEnable(g_device_id, i, false) !=
RET_STATUS_ENUM.RET_OK)
                {
```

```
                    MessageBox.Show("设置完整点云事件失败");
                    return;
                }
            }
            break;

            case "ROI点云数据"://获取ROI点云数据，需要进行以下配置
            if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_FULL) != RET_STATUS_ENUM.RET_OK)
            {
                MessageBox.Show("设置ROI点云事件失败");
                return;
             }
             //如果没有开启点云转换，则需要开启
             if (HPS3D_Device.SetPointCloudMode(g_device_id, true,
POINTCLOUD_MIRROR_MODE_ENUM.MIRROR_DISABLE) != RET_STATUS_ENUM.RET_OK)
             {
                 MessageBox.Show("设置ROI点云事件失败");
                 return;
              }
              //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
              //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
              if (HPS3D_Device.SetROIEnable(g_device_id, 0, true) !=
RET_STATUS_ENUM.RET_OK)
              {
                  MessageBox.Show("设置ROI点云事件失败");
                  return;
              }
              break;

              case "ROI完整深度数据"://获取ROI完整深度数据，需要进行以下配置
              if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_FULL) != RET_STATUS_ENUM.RET_OK)
              {
                  MessageBox.Show("设置ROI完整深度事件失败");
                  return;
               }
               //如果开启了点云转换，则需要关闭
               if (HPS3D_Device.SetPointCloudMode(g_device_id, false,
POINTCLOUD_MIRROR_MODE_ENUM.MIRROR_DISABLE) != RET_STATUS_ENUM.RET_OK)
               {
                   MessageBox.Show("设置ROI完整深度事件失败");
                   return;
               }
```

```
            //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
            //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
            if (HPS3D_Device.SetROIEnable(g_device_id, 0, true) !=
RET_STATUS_ENUM.RET_OK)
            {
                    MessageBox.Show("设置ROI完整深度事件失败");
                    return;
            }
            break;

            case "ROI简单深度数据"://获取ROI简单深度数据，需要进行以下配置
            if (HPS3D_Device.SetOutputDataType(g_device_id,
OUTPUT_TYPE_ENUM.OUTPUT_DISTANCE_SIMPLE) != RET_STATUS_ENUM.RET_OK)
            {
                   MessageBox.Show("设置ROI简单深度事件失败");
                   return;
            }
            //如果没有开启ROI需要开启（ROI配置需要使用客户端）SDK仅提供ROI使能开关
            //以下开启ROI 0,注意如果没有使用客户端绘制ROI区域，设置ROi使能将无效
            if (HPS3D_Device.SetROIEnable(g_device_id, 0, true) !=
RET_STATUS_ENUM.RET_OK)
            {
                    MessageBox.Show("设置ROI简单深度事件失败");
                    return;
            }
             break;
             }
```

7、启动测量数据：

```
//1.单次测量
//将运行模式设置为单次测量模式
if (HPS3D_Device.SetRunMode(g_device_id, RUN_MODE_ENUM.RUN_SINGLE_SHOT) !=
RET_STATUS_ENUM.RET_OK)
{
       MessageBox.Show("设置失败，请重试");
       return;
 }
//2.连续测量
//将运行模式设置为连续测量模式
if (HPS3D_Device.SetRunMode(g_device_id, RUN_MODE_ENUM.RUN_CONTINUOUS) !=
RET_STATUS_ENUM.RET_OK)
{
       MessageBox.Show("设置失败，请重试");
         return;
```

```
}
```

8、停止输出数据：（连续输出数据后必须停止）

```
//停止输出数据
if (HPS3D_Device.SetRunMode(g_device_id, RUN_MODE_ENUM.RUN_IDLE) !=
RET_STATUS_ENUM.RET_OK)
{
    MessageBox.Show("设置失败，请重试");
    return;
}
```

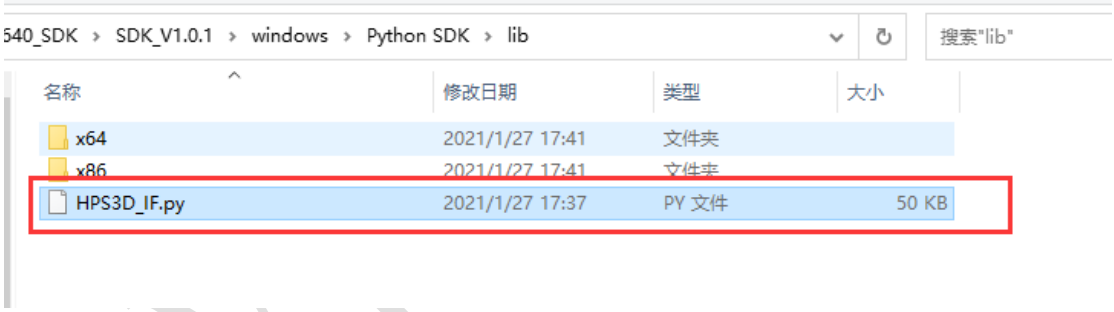
9、运行完成、断开设备连接：

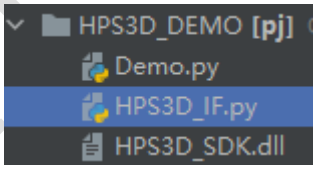
```
//停止输出数据
HPS3D_Device.DisConnect(g_device_id);
```

2.3 Python 开发环境配置及集成到 IDE 中

2.3.1 工程环境配置与集成到 PyCharm 中

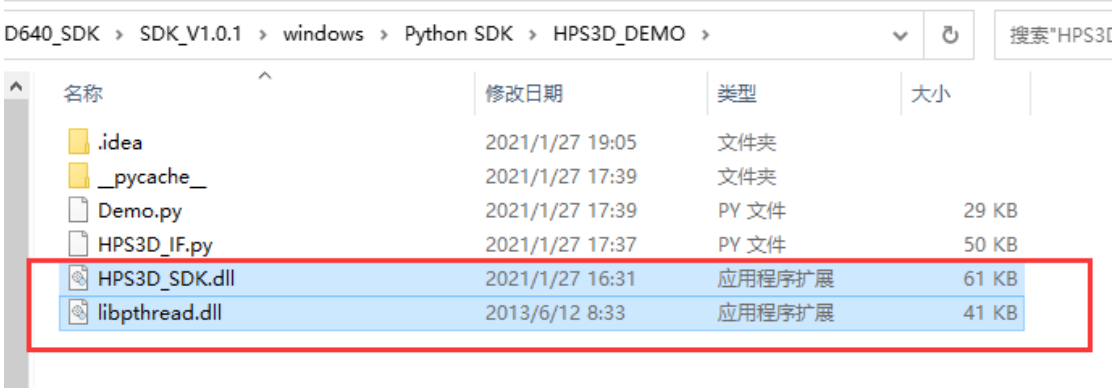
1、将 lib 目录下的 HPS3D_IF.py 添加到用户工程中





2、根据用户 PyCharm 的平台，选择 x64 或 x86 目录下的 dll 文件，这里以 x64 为例。

3、将 x64 目录下的 HPS3D_SDK.dll 拷贝到程序运行的目录下；



2.3.2 在用户项目中使用 SDK

以下为获取传感器数据的简单步骤，更具体的请参考 HPS3D_DOME 中的示例代码。在使用 Demo 中相关函数时，请参照 HPS3D_IF.py 中对应函数的参数、返回值、和一些常量类型定义等。

1、 参照用户手册中的接线方式，将激光位移传感器与 PC 端连接。

2、 导入 HPS3D_IF.py：（必要）

```
from HPS3D_IF import *
```

3、设置连接参数并连接设备：（必要）

```
# 申明全局变量
G_DEVICE_ID = 99
#以下为设备的默认 IP 和端口号
dict = connect_by_ethernet("192.168.30.202", 12345)
if dict['ret_value'] != RET_OK:
    print("设备连接失败，退出程序!")
    # 连接失败，则退出程序
    sys.exit()
else:
    print("设备连接成功!")
    # 连接失败后将设备 ID 取出
    G_DEVICE_ID = dict["device_id"]
```

4、注册传感器传感器接收回调函数：（必要）

```
dict = set_output_callback(POUTPUTEVENTFUNC, G_DEVICE_ID, None)
if dict['ret_value'] != RET_OK:
    print("输出事件回调函数注册失败!")
    # 输出事件回调函数注册失败，则退出程序
    sys.exit()
else:
    print("输出事件回调函数注册成功!")
```

5、配置传感器： （首次使用配置即可）

注：用户可以使用客户端配置，配置完成后点击保存到用户配置表即可（保存到用户配置表中的数据，重新上电不会丢失）

```
global G_DEVICE_ID
# 积分时间根据现场环境和被测物体信号反射强度进行调节（必要）
dict = set_integ_time(G_DEVICE_ID, 200)
if dict['ret_value'] != RET_OK:
    print("设置积分时间失败!")
else:
    print("设置积分时间成功!")
```

```
# 滤波器配置（可选）
# 1.平滑滤波器配置，根据需要自行选择滤波器类型，以下为关闭
dict = set_smooth_filter(G_DEVICE_ID, SMOOTH_FILTER_DISABLE, 0)
if dict['ret_value'] != RET_OK:
    print("设置平滑滤波器失败!")
else:
    print("设置平滑滤波器成功!")

# 2.卡尔曼滤波器，根据需要自行选择滤波器参数,以下为关闭
dict = set_simple_kalman_filter(G_DEVICE_ID, False, 0, 0, 0)
if dict['ret_value'] != RET_OK:
    print("设置卡尔曼滤波器失败!")
else:
    print("设置卡尔曼滤波器成功!")

# 距离偏移（可选），以下默认设置为 0
dict = set_offset(G_DEVICE_ID, 0)
if dict['ret_value'] != RET_OK:
    print("设置距离偏移失败!")
else:
    print("设置距离偏移成功!")

# 将设置保存到用户配置表，保存后重新上电不会丢失
dict = profile_save_to_user(G_DEVICE_ID)
if dict['ret_value'] != RET_OK:
    print("用户配置表保存失败!")
else:
    print("用户配置表保存成功!")
```

6、设置设备输出类型：（必要）

```
def set_output_type(type_mode):
    """选定一种需要输出的数据，配置完成后将通过回调函数 output_event_func 通知对应事件

        注意:
            1.调用前需要确保设备已经正常连接

            2.执行该命令前，需要将运行模式设置为 RUN_IDLE
```

```
            3.每次只能输出一种类型的数据
        参数:
            type_mode: (int)输出数据类型,例如:
                type_mode = 1: 简单深度事件

                type_mode = 2: 完整深度事件

                type_mode = 3: 幅值事件

                type_mode = 4: 完整点云事件

                type_mode = 5: ROI 点云事件

                type_mode = 6: ROI 完整深度事件

                type_mode = 7: ROI 简单深度事件

        """
    global G_DEVICE_ID
    if type_mode == 1:  # 设置简单深度事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_SIMPLE)['ret_value'] != RET_OK:
            print("设置简单深度事件失败!");
            return

        # 如果开启了ROI 需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
        for i in range(0, 8):
            if set_roi_enable(G_DEVICE_ID, i, False)['ret_value'] !=
RET_OK:
                print("设置简单深度事件失败!")
                return

    elif type_mode == 2:  # 设置完整深度事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_FULL)['ret_value'] != RET_OK:
            print("设置完整深度事件失败!");
            return

        # 如果开启了点云转换，则需要关闭
        if set_point_cloud_mode(G_DEVICE_ID, False,
MIRROR_DISABLE)['ret_value'] != RET_OK:
            print("设置完整深度事件失败!");
```

```
            return

        # 如果开启了 ROI 需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
        for i in range(0, 8):
            if set_roi_enable(G_DEVICE_ID, i, False)['ret_value'] !=
RET_OK:
                print("设置完整深度事件失败!")
                return

    elif type_mode == 3:  # 设置幅值事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_AMPLITUDE)['ret_value'] != RET_OK:
            print("设置幅值事件失败!");
            return

    elif type_mode == 4:  # 设置完整点云事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_FULL)['ret_value'] != RET_OK:
            print("设置完整点云事件失败!");
            return

        # 如果没有开启点云转换，则需要开启
        if set_point_cloud_mode(G_DEVICE_ID, True,
MIRROR_DISABLE)['ret_value'] != RET_OK:
            print("设置完整点云事件失败!");
            return

        # 如果开启了 ROI 需要关闭，开启了哪个就关闭哪个，以下示例为全部关闭
        for i in range(0, 8):
            if set_roi_enable(G_DEVICE_ID, i, False)['ret_value'] !=
RET_OK:
                print("设置完整点云事件失败!")
                return

    elif type_mode == 5:  # 设置 ROI 点云事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_FULL)['ret_value'] != RET_OK:
            print("设置 ROI 点云事件失败!");
            return

        # 如果没有开启点云转换，则需要开启
```

```
        if set_point_cloud_mode(G_DEVICE_ID, True,
MIRROR_DISABLE)['ret_value'] != RET_OK:
            print("设置 ROI 点云事件失败!");
            return

        # 如果没有开启 ROI 需要开启（ROI 配置需要使用客户端）SDK 仅提供 ROI 使能
开关
        print('以下开启 ROI0，注意如果没有使用客户端绘制 ROI 区域，设置 ROi 使
能将无效')
        if set_roi_enable(G_DEVICE_ID, 0, True)['ret_value'] !=
RET_OK:
            print("设置 ROI 点云事件失败!")
            return

    elif type_mode == 6:  # 设置 ROI 完整深度事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_FULL)['ret_value'] != RET_OK:
            print("设置 ROI 完整深度事件失败!");
            return

        # 如果开启了点云转换，则需要关闭
        if set_point_cloud_mode(G_DEVICE_ID, False,
MIRROR_DISABLE)['ret_value'] != RET_OK:
            print("设置 ROI 完整深度事件失败!");
            return

        # 如果没有开启 ROI 需要开启（ROI 配置需要使用客户端）SDK 仅提供 ROI 使能
开关
        print('以下开启 ROI0，注意如果没有使用客户端绘制 ROI 区域，设置 ROi 使
能将无效')
        if set_roi_enable(G_DEVICE_ID, 0, True)['ret_value'] !=
RET_OK:
            print("设置 ROI 完整深度事件失败!")
            return

    elif type_mode == 7:  # 设置 ROI 简单深度事件，需要进行以下配置
        # 设置输出数据类型
        if set_output_data_type(G_DEVICE_ID,
OUTPUT_DISTANCE_SIMPLE)['ret_value'] != RET_OK:
            print("设置 ROI 完整深度事件失败!");
            return

        # 如果没有开启 ROI 需要开启（ROI 配置需要使用客户端）SDK 仅提供 ROI 使能
```

```
开关
        print('以下开启 ROI0，注意如果没有使用客户端绘制 ROI 区域，设置 ROi 使能将无效')
        if set_roi_enable(G_DEVICE_ID, 0, True)['ret_value'] != RET_OK:
            print("设置 ROI 完整深度事件失败!")
            return

    else:
        print("请参照注释，正确输入参数")
        return

    print("设置输出类型配置成功!")
```

7、启动测量数据

```
SINGLE = True
if SINGLE == True:  # 单次测量
    set_run_mode(G_DEVICE_ID, RUN_SINGLE_SHOT)

else:  # 连续测量
    set_run_mode(G_DEVICE_ID, RUN_CONTINUOUS)
```

8、停止输出数据：（连续输出数据后必须停止）

```
# 停止输出数据
set_run_mode(G_DEVICE_ID, RUN_IDLE)
```

9、运行完成、断开设备连接：

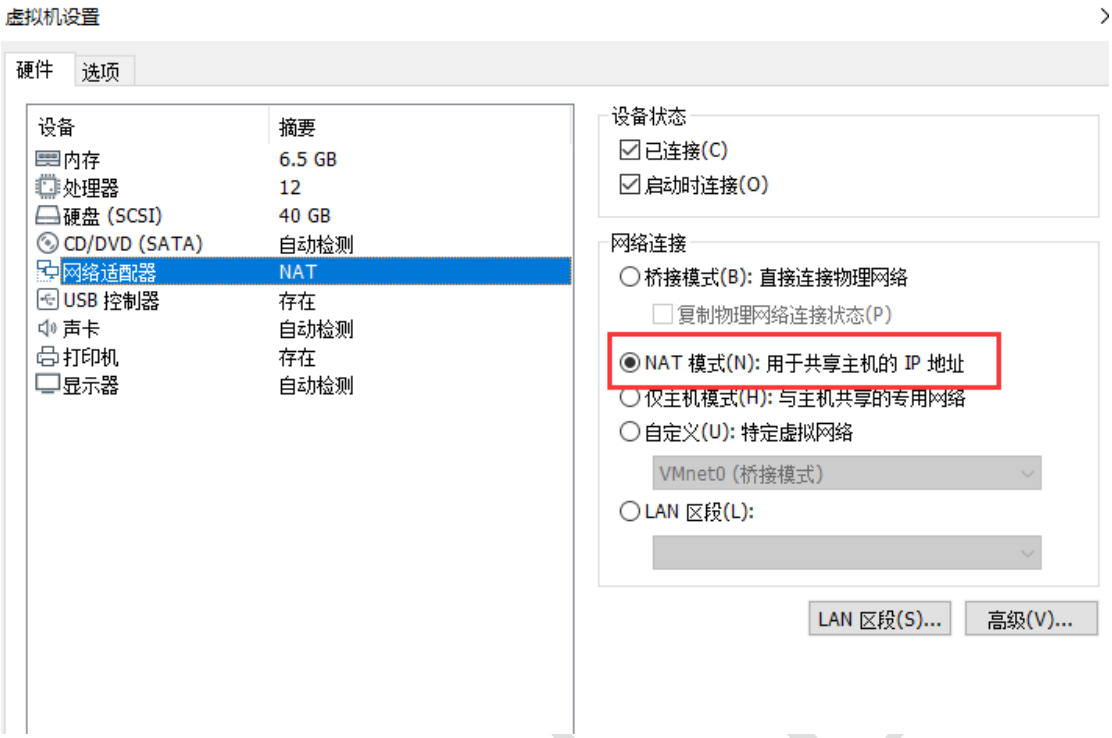
```
disconnect(G_DEVICE_ID)
```

2.4 Linux 开发环境配置及集成到 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 2021.02.05 V1.0.1 的 SDK 进行编写

2.4.1 HPS3D640 设备连接

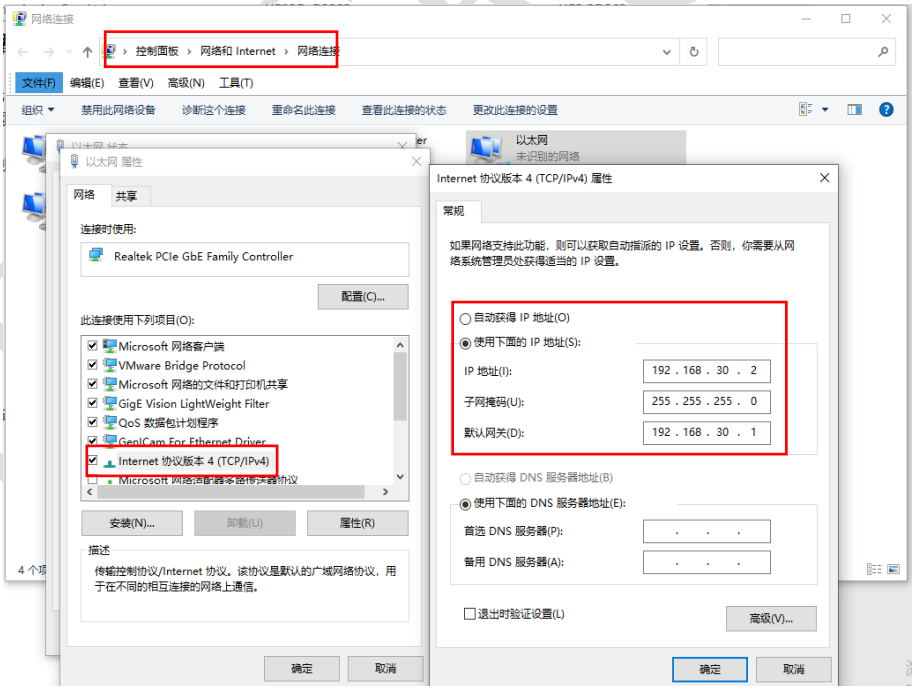
将 WM 虚拟机网络配置为 NAT 模式



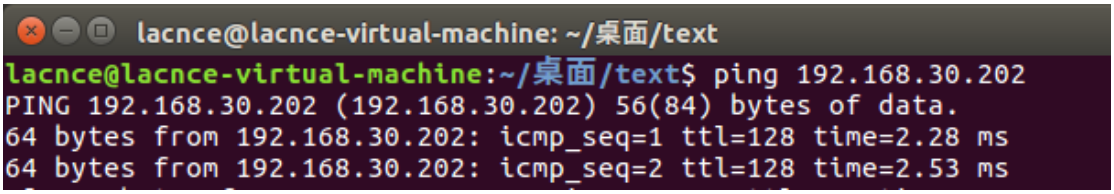
HPS-3D640 通过 LAN 接口与主机通讯。

传感器默认 IP 地址是 192.168.30.202， 端口是 12345。

传感器网线接入主机的以太网口后，需要在主机配置 ip 地址和子网掩码，如下图。

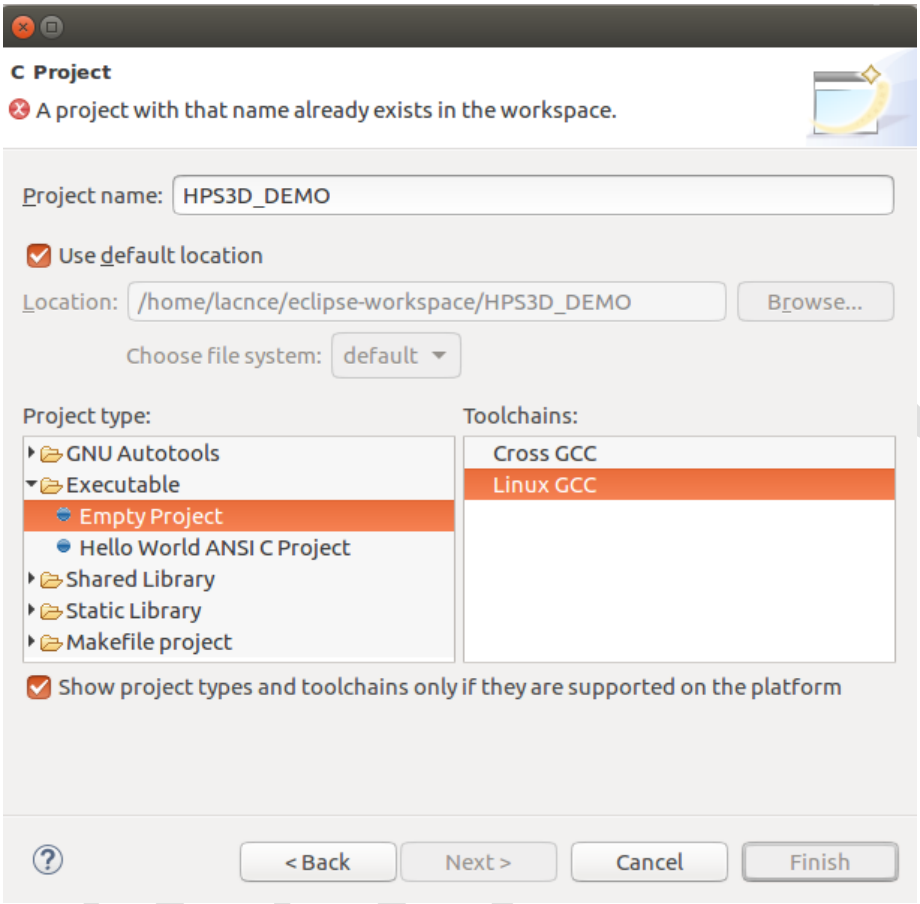


3）打开终端，输入 Ping 192.168.30.202 若出现以下现象表示连接成功

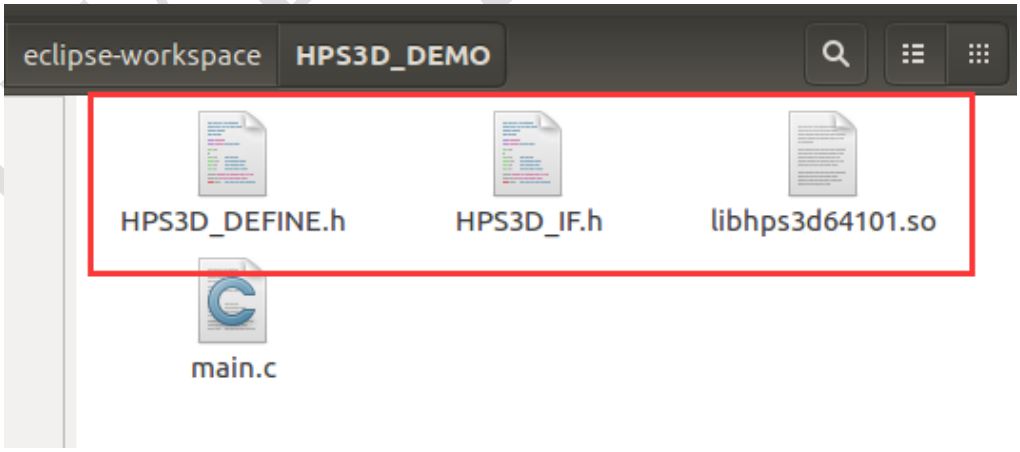


2.4.2 工程环境配置与集成

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 HPS3D_DEFINE.h、HPS3D_IF.h 和 libhps3d64101.so 文件拷贝进项目工程



方法一：

打开终端，输入 sudo cp libhps3d64101.so /usr/local/lib/，将 libhps3d64101.so 拷贝到 /usr/local/lib/目录下；之后执行 sudo ldconfig 进行加载。

```
lacnce@lacnce-virtual-machine: ~/eclipse-workspace/HPS3D_DEMO
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$ sudo cp libhps3d64
101.so /usr/local/lib/
[sudo] password for lacnce:
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$ sudo ldconfig
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$
```

方法二:

main.c 写入测试代码后，编译时加入指定动态库链接地址如 gcc main.c -Wl,-rpath=./ -L./ -lhps3d64101 -o app

```
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$ gcc main.c -Wl,-rp
ath=./ -L./ -lhps3d64101 -o app
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$
```

使用 ./app 执行程序

```
lacnce@lacnce-virtual-machine:~/eclipse-workspace/HPS3D_DEMO$ ./app
DEBUG信息回调函数注册:..........成功!
ip = 192.168.30.202,port = 12345
ConfigInit :1
设备连接:..........成功!
输出事件回调函数注册:..........成功!
看门狗关闭:..........成功!
设置积分时间:..........成功!
设置平滑滤波器:..........成功!
设置卡尔曼滤波器:..........成功!
设置距离偏移:..........成功!
用户配置表保存:..........成功!
设置输出类型配置:..........成功!
设备初始化:..........成功!
执行输出一次数据，按回车键继续运行!
aver_distance:2198  max_distance:2675  min_distance:1558
valid_aver_amp:17  all_aver_amp:13  fps:1
distance_data:65530
```

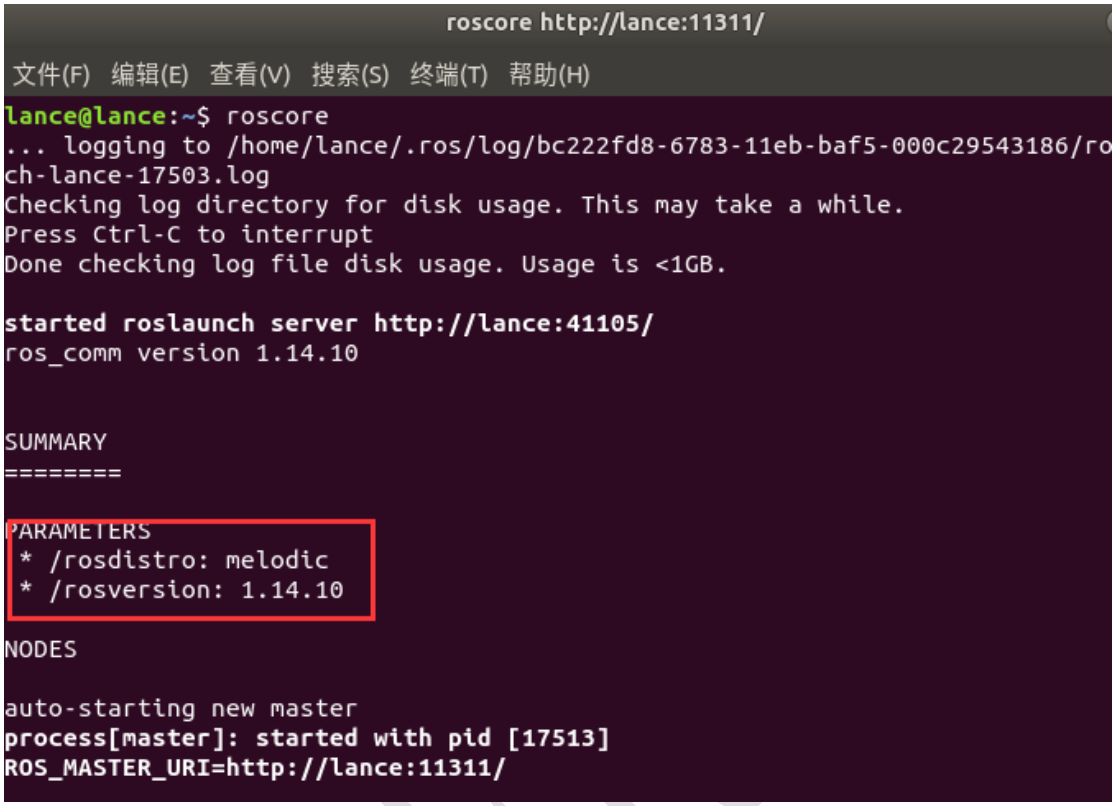
2.4.3 在用户项目中使用 SDK

使用方式 VS 平台下类似。2.1.2 在用户项目中使用 SDK

2.5 ROS 开发环境配置及集成到 IDE 中

SDK 适合在 Linux 操作系统上 ROS 平台使用，这里以 Ubuntu18.04 为例。因为安装

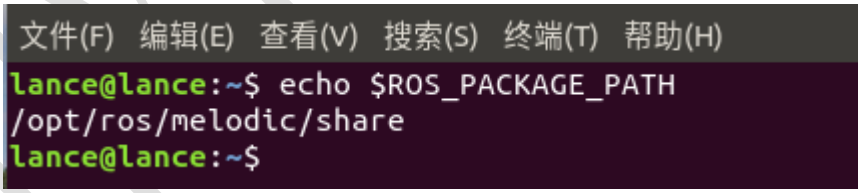
的是 Ubuntu18.04 的 Linux 操作系统，所以安装与之对应的 ROS 版本为发行版本 melodic，这里安装 1.14.10 版本。在终端输入 rosrun，运行 ROS 总线，可查看运行 ROS 的版本，如下图所示：



设备连接请参照本文档的“2.4.1 HPS3D640 设备连接”的内容。

2.5.1 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/melodic/setup.bash。

（2）创建一个工作空间，输入 mkdir -p ~/HPS3D_SDK_ROS_Demo/src。

（3）输入 cd HPS3D_SDK_ROS_Demo/src 进入 src 目录

```
lance@lance:~$ cd HPS3D_SDK_ROS_Demo/src/
lance@lance:~/HPS3D_SDK_ROS_Demo/src$
```

（4）初始化工作空间，在 src 目录下执行 catkin_init_workspace，如下图所示：

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

（5）输入 cd ../ 进入工作空间根目录，在再输入 catkin_make，如下图所示：

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src$ cd ../
lance@lance:~/HPS3D_SDK_ROS_Demo$ catkin_make
Base path: /home/lance/HPS3D_SDK_ROS_Demo
Source space: /home/lance/HPS3D_SDK_ROS_Demo/src
Build space: /home/lance/HPS3D_SDK_ROS_Demo/build
Devel space: /home/lance/HPS3D_SDK_ROS_Demo/devel
Install space: /home/lance/HPS3D_SDK_ROS_Demo/install
####
#### Running command: "cmake /home/lance/HPS3D_SDK_ROS_Demo/src -D
REFIX=/home/lance/HPS3D_SDK_ROS_Demo/devel -DCMAKE_INSTALL_PREFIX=
S3D_SDK_ROS_Demo/install -G Unix Makefiles" in "/home/lance/HPS3D_
uild"
####
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
```

（6）此时工作空间下生成了 “build”和“devel”两个文件夹。在 devel 文件夹下，您可以看到很多 setup.*sh 文件。输入 source devel/setup.bash，配置工作空间，如下图所示：

```
lance@lance:~/HPS3D_SDK_ROS_Demo$ ls
build  devel  src
lance@lance:~/HPS3D_SDK_ROS_Demo$ ls devel/
cmake.lock  lib              local_setup.sh   setup.bash  _setup_util.py
env.sh      local_setup.bash  local_setup.zsh  setup.sh    setup.zsh
lance@lance:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
lance@lance:~/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，如库文件、可执行文件以及产生的代码都被放置在 devel 中。

2.5.2 创建一个 ROS 包（catkin 包）

（1）ROS 包它直接依赖于以下三个包： rospy，roscpp，pcl_conversion.，输入 cd src 进入 src 目录，再输入 catkin_create_pkg hps_camera rospy roscpp pcl_conversions，如下图所示:

```
lance@lance:~/HPS3D_ROS_Demo$ cd src/
lance@lance:~/HPS3D_ROS_Demo/src$ catkin_create_pkg hps_camera std_msgs rospy ro
scpp pcl_conversions
Created file hps_camera/package.xml
Created file hps_camera/CMakeLists.txt
Created folder hps_camera/include/hps_camera
Created folder hps_camera/src
Successfully created files in /home/lance/HPS3D_ROS_Demo/src/hps_camera. Please
adjust the values in package.xml.
```

（2）在终端输入 rospack depends hps_camera，可以查看到 ROS 包的依赖关系，可看到本身加的依赖的三个包： rospy、roscpp 以及 pcl_conversions，如下图所示：

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

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

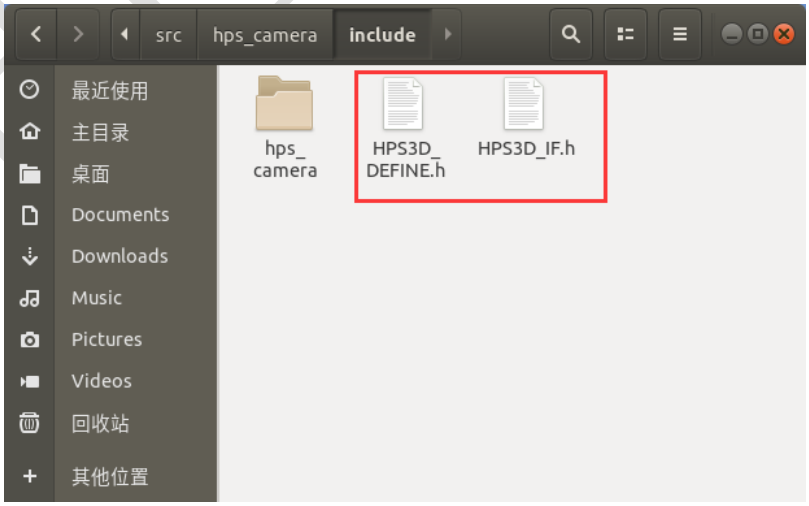
2.5.3 创建 ROS 的深度相机客户端节点

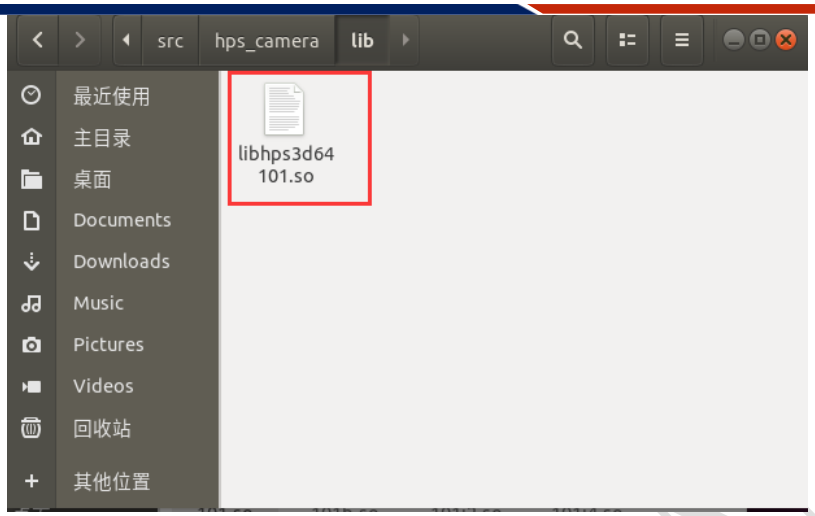
在这部分会提供例程，用户可根据需要自行修改或编写自己的程序

1、将 HPS3D_DEFINE.h、HPS3D_DEVICE.h 和.so 文件集成到工程

在 ROS 包目录下创建 include 目录（该目录通常会自动生成）以及 lib 目录，将 HPS3D_DEFINE.h 和 HPS3D_IF.h 拷贝至 include 目录下，xxx.so 拷贝至 lib 目录下

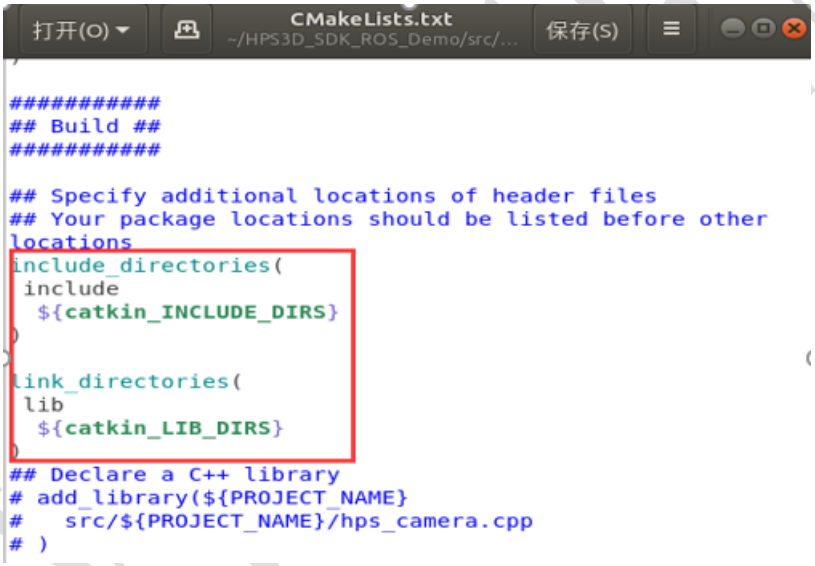
```
lance@lance:~/HPS3D_SDK_ROS_Demo/src$ cd hps_camera/
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ mkdir lib
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$
```





并修改 CmakeLists.txt 文件，添加如下代码：注意库名称应与 lib 目录下库名匹配;

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ mkdir lib
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo gedit CMakeLists.txt
[sudo] lance 的密码：
```



2、创建 ROS 的深度相机客户端节点

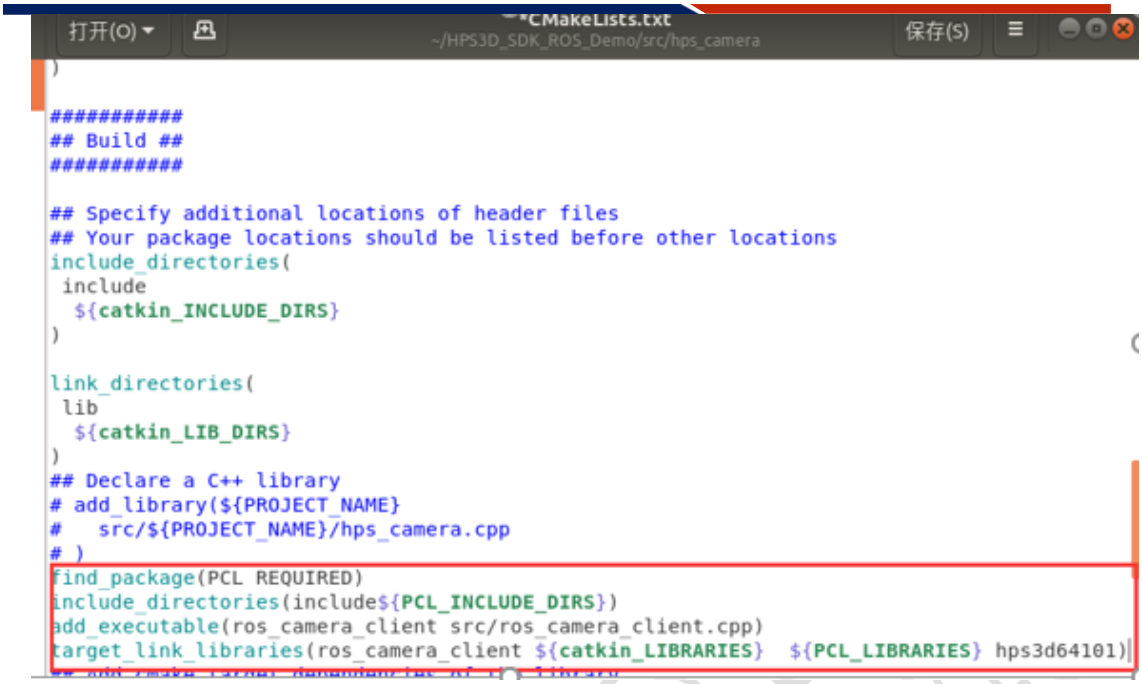
（1）在 src 目录中，创建 ros_camera_client.cpp 如下图所示：

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo touch src/ros_camera_client.cpp
```

（2）修改 CMakeLists.txt 文件，添加如下代码，如下图所示：

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo gedit CMakeLists.txt
```

```
find_package(PCL REQUIRED)
include_directories(include${PCL_INCLUDE_DIRS})
add_executable(ros_camera_client src/ros_camera_client.cpp)
target_link_libraries(ros_camera_client ${catkin_LIBRARIES}  ${PCL_LIBRARIES} hps3d64101)
```



3、编写 ROS 的深度相机客户端节点

输入 sudo gedit src/ros_camera_client.cpp 打开 ros_camera_client.cpp 文件

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ sudo gedit src/ros_camera_client.cpp
```

内容编写具体请参照 DEMO 中的示例

1）添加头文件，代码如下：

```
#include "ros/ros.h"//ros
#include <sstream>
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include "HPS3D_IF.h"
#include <time.h>
#include <sys/time.h>
// PCL specific includes
#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
```

2）在主函数中，ros 初始化，SDK 接口配置等，代码如下：

```
ros::Publisher camera_pub;//Global variable, because the observer callback function
needs to be used
int main(int argc, char **argv)
{
    ros::init(argc, argv, "ros_camera_client");//ros init
    ros::NodeHandle n;//Create a node
```

```
    uint8_t ret = RET_OK;
    //Install the signal
    if (signal(SIGINT, signal_handler) == SIG_ERR || signal(SIGTSTP, signal_handler)
== SIG_ERR)
    {
        printf("sigint error");
    }
    //Create a topic
    camera_pub = n.advertise<sensor_msgs::PointCloud2>("output", 1);
    //set debug enable and install printf log callback function
    HPS3D_SetDebugCallBack(my_printf,NULL);
    do
    {
        //Device Connection
        ret = HPS3D_ConnectByEthernet((char*)"192.168.30.202", 12345, &g_device_id);
        if (ret != RET_OK)
        {
            printf("HPS3D_ConnectByEthernet faild!\n");
            break;
        }
        else
        {
            printf("HPS3D_ConnectByEthernet succeed!\n");
        }
        /*set OutputCallBack Func*/
        ret = HPS3D_SetOutputCallBack(User_Func, g_device_id, NULL);
        if (ret != RET_OK)
        {
            printf("HPS3D_SetOutputCallBack faild!\n");
            break;
        }
        else
        {
            printf("HPS3D_SetOutputCallBack succeed!\n");
        }
        /*set watch dog enable*/
        ret = HPS3D_SetWatchDogEnable(g_device_id, false);
        if (ret != RET_OK)
        {
            printf("HPS3D_SetWatchDogEnable faild!\n");

        }
        else
        {
```

```
            printf("HPS3D_SetWatchDogEnable succeed!\n");
        }
        /*set convert point cloud data enable*/
        HPS3D_SetOutputDataType(g_device_id, OUTPUT_DISTANCE_FULL);
        HPS3D_SetPointCloudMode(g_device_id, true, MIRROR_DISABLE);
        //Set running mode
        HPS3D_SetRunMode(g_device_id,RUN_CONTINUOUS);

    } while (0);
    if (ret != RET_OK)
    {
        //Remove device and disconnect
        HPS3D_DisConnect(g_device_id);
        printf("Initialization failed, Remove device\n");
        return 0;
    }
    printf("Program running...\n");
    printf("Use Ctrl + C can exit the program!\n");
    while (1)
    {
        usleep(1000000);
        printf("FPS:%d\n", g_fps);
        g_fps = 0;
    }
    return 0;
}
```

3）输出数据回调函数，代码如下：

```
//回调函数，请不要在回调函数中进行耗时操作
void User_Func(uint8_t out_id, uint8_t event, void *contex)
{
    pcl::PointCloud<pcl::PointXYZ> cloud;
    sensor_msgs::PointCloud2 output;
    g_fps++;
    if (event == EVENT_SIMPLEDISTANCERECVD) //简单深度通知事件
    {
        //简单深度通知事件只能获取数学统计信息
        HPS3D_GetAverageDistance_CB(out_id, &g_aver_distance); //获取平均距离
        HPS3D_GetMaxDistance_CB(out_id, &g_max_distance);        //获取最大距离
        HPS3D_GetMinDistance_CB(out_id, &g_min_distance);        //获取最小距离
        HPS3D_GetValidAverAmplitude_CB(out_id, &g_valid_aver_amp);    //有效平均幅值，计算不包含0幅值
        HPS3D_GetAllAverAmplitude_CB(out_id, &g_all_aver_amp);    //平均幅值，计算包含0幅值
```

```
                                                                    //以下为数据显
示
        printf("aver_distance:%d  max_distance:%d  min_distance:%d\n",
g_aver_distance, g_max_distance, g_min_distance);
        printf("valid_aver_amp:%d  all_aver_amp:%d  fps:%d\n", g_valid_aver_amp,
g_all_aver_amp, g_fps);
    }
    else if (event == EVENT_FULLDISTANCERECVD) //完整深度通知事件
    {
        //完整深度通知事件可以获取数学统计信息和每个像素点的深度值

        //获取每个像素点的深度数据
        HPS3D_GetDistanceData_CB(out_id, g_distance_data);

        //获取深度数据的数学统计信息
        HPS3D_GetAverageDistance_CB(out_id, &g_aver_distance); //获取平均距离
        HPS3D_GetMaxDistance_CB(out_id, &g_max_distance);        //获取最大距离
        HPS3D_GetMinDistance_CB(out_id, &g_min_distance);        //获取最小距离
        HPS3D_GetValidAverAmplitude_CB(out_id, &g_valid_aver_amp);    //有效平均幅
值，计算不包含0幅值
        HPS3D_GetAllAverAmplitude_CB(out_id, &g_all_aver_amp);    //平均幅值，计算包
含0幅值

                                                                    //以下为数据显
示
        printf("aver_distance:%d  max_distance:%d  min_distance:%d\n",
g_aver_distance, g_max_distance, g_min_distance);
        printf("valid_aver_amp:%d  all_aver_amp:%d  fps:%d\n", g_valid_aver_amp,
g_all_aver_amp, g_fps);
        printf("distance_data:%d\n", g_distance_data[1]);
    }
    else if (event == EVENT_AMPLITUDE) //幅值通知事件
    {
        //振幅通知事件可以获取每个像素点的灰度数据或者幅值数据，以及幅值相关的数学统
计信息

        //获取每个像素点的灰度数据，灰度数据由振幅数据归一化处理得到
        HPS3D_GetGrayscaleData_CB(out_id, g_grayscale_data);

        //获取每个像素点的幅值数据
        HPS3D_GetAmplitudeData_CB(out_id, g_amplitude_data);

        //获取振幅相关的数学统计信息
        HPS3D_GetValidAverAmplitude_CB(out_id, &g_valid_aver_amp); // 有效平均幅值
```

```
        HPS3D_GetMaxAmplitudeData_CB(out_id, &g_max_amp); //最大幅值
        HPS3D_GetMinAmplitudeData_CB(out_id, &g_min_amp); //最小幅值

                                                          //以下为数据显示
        printf("valid_aver_amp:%d  max_amp:%d  min_amp:%d\n", g_valid_aver_amp,
g_max_amp, g_min_amp);
        printf("grayscale_data:%d  amplitude_data:%d  fps:%d\n",
g_grayscale_data[1], g_amplitude_data[1], g_fps);
    }
    else if (event == EVENT_FULLPOINTCLOUDRECVD) //完整点云通知事件
    {
        //完整点云图，能够获取每个像素的点云数据，和深度数据的数学统信息，点云数据自
动过滤了无效值


        //获取每个像素的点云数据
        HPS3D_GetPointCloudData_CB(out_id, g_point_data, &g_point_count); //获取点云
数据

        for (size_t i = 0; i < g_point_count; i++)
        {
                pcl::PointXYZ temp;
                temp.x = g_point_data[i][0];
                temp.y = g_point_data[i][1];
                temp.z = g_point_data[i][2];
                cloud.points.push_back(temp);
        }

        pcl::toROSMsg(cloud, output);
        output.header.frame_id = "hps";
        camera_pub.publish(output);
    }
    else if (event == EVENT_ROIPOINTCLOUDRECVD) //ROI点云通知事件
    {
        //ROI点云事件可以获取ROI点云信息和每个ROI的深度数学统计信息
        uint32_t count = 0;
        //其中获取点云数据有两种方式
        //第一种，通过点云数据获取接口直接获取点云信息，自动将多个ROI点云数据拼接成
        HPS3D_GetPointCloudData_CB(out_id, g_point_data, &g_point_count); //获取点云
数据

        cloud.points.clear();
        for (size_t i = 0; i < g_point_count; i++)
        {
```

```
            pcl::PointXYZ temp;
            temp.x = g_point_data[i][0];
            temp.y = g_point_data[i][1];
            temp.z = g_point_data[i][2];
            cloud.points.push_back(temp);
        }

        pcl::toROSMsg(cloud, output);
        output.header.frame_id = "hps";
        camera_pub.publish(output);

    }
    else if (event == EVENT_ROIFULLDISTANCERECVD) //ROI完整深度通知事件
    {

        //ROI完整深度事件，可以获取每个ROI的深度数据和每个ROI的深度数学统计信息

        uint8_t group_id = 0;
        uint8_t roi_number = 0;

        //获取指定ROI的深度信息前，需要通过ROI参数接口，获取当前输出的ROI信息，（有几个ROI，ROI的ID等）
        //获取ROI参数
        HPS3D_GetOutRoiParam_CB(out_id, &group_id, &roi_number, g_roi_id);
        int i = 0;
        for (i = 0; i < roi_number; i++)
        {
            //获取ROI距离值
            HPS3D_GetOutRoiDistanceData_CB(out_id, g_roi_id[i],
g_roi_data[g_roi_id[i]], g_roi_area[g_roi_id[i]]);

            //ROI深度数据的数据量需要通过ROI区域计算
            g_roi_data_count = (g_roi_area[g_roi_id[i]][2] -
g_roi_area[g_roi_id[i]][0] + 1) * (g_roi_area[g_roi_id[i]][3] -
g_roi_area[g_roi_id[i]][1] + 1);
            printf("roi_data_count:%d roi_data:%d\n", g_roi_data_count,
g_roi_data[g_roi_id[i]][g_roi_data_count - 1]);

            //获取ROI数学统计信息
            HPS3D_GetOutRoiAverageDistance_CB(out_id, g_roi_id[i],
&g_roi_aver_distance);
            HPS3D_GetOutRoiMaxDistance_CB(out_id, g_roi_id[i],
&g_roi_max_distance);
            HPS3D_GetOutRoiMinDistance_CB(out_id, g_roi_id[i],
```

```
&g_roi_min_distance);
            HPS3D_GetOutRoiAllAverAmplitude_CB(out_id, g_roi_id[i],
&g_roi_all_aver_amp);
            HPS3D_GetOutRoiValidAverAmplitude_CB(out_id, g_roi_id[i],
&g_roi_valid_aver_amp);
            printf("roi_aver_distance:%d  roi_max_distance:%d
roi_min_distance:%d\n", g_roi_aver_distance, g_roi_max_distance, g_roi_min_distance);
            printf("roi_all_aver_amp:%d  roi_valid_aver_amp:%d\n",
g_roi_all_aver_amp, g_roi_valid_aver_amp);
        }
    }
    else if (event == EVENT_ROISIMPLEDISTANCERECVD) //ROI简单深度通知事件
    {

        //简单深度事件，可以获取每个ROI的深度数学统计信息
        uint8_t group_id = 0;
        uint8_t roi_number = 0;

        //获取指定ROI的深度信息前，需要通过ROI参数接口，获取当前输出的ROI信息，（有
几个ROI，ROI的ID等）
        //获取ROI参数
        HPS3D_GetOutRoiParam_CB(out_id, &group_id, &roi_number, g_roi_id);
        int i = 0;
        for (i = 0; i < roi_number; i++)
        {
            //获取ROI数学统计信息
            HPS3D_GetOutRoiAverageDistance_CB(out_id, g_roi_id[i],
&g_roi_aver_distance);
            HPS3D_GetOutRoiMaxDistance_CB(out_id, g_roi_id[i],
&g_roi_max_distance);
            HPS3D_GetOutRoiMinDistance_CB(out_id, g_roi_id[i],
&g_roi_min_distance);
            HPS3D_GetOutRoiAllAverAmplitude_CB(out_id, g_roi_id[i],
&g_roi_all_aver_amp);
            HPS3D_GetOutRoiValidAverAmplitude_CB(out_id, g_roi_id[i],
&g_roi_valid_aver_amp);
            printf("roi_aver_distance:%d  roi_max_distance:%d
roi_min_distance:%d\n", g_roi_aver_distance, g_roi_max_distance, g_roi_min_distance);
            printf("roi_all_aver_amp:%d  roi_valid_aver_amp:%d\n",
g_roi_all_aver_amp, g_roi_valid_aver_amp);
        }
    }
    else if (event == EVENT_DEVDISCONNECT) //断开连接
    {
```

```
        //断开连接，回收资源
        HPS3D_DisConnect(out_id);
    }
    else
    {
        printf("system error!\n");
    }
}
```

4）退出事件回调函数

```
//check ctrl+c signal
void signal_handler(int signo)
{
    if (HPS3D_DisConnect(g_device_id) != RET_OK)
    {
        printf("HPS3D_DisConnect faild\n");
    }
    else
    {
        printf("HPS3D_DisConnect succeed\n");
    }
    exit(0);
}
```

2.5.4 测试 ROS 的深度相机客户端节点

1）输入 cd ~/HPS3D_SDK_ROS_Demo/ 进入项目根目录

```
lance@lance:~/HPS3D_SDK_ROS_Demo/src/hps_camera$ cd ~/HPS3D_SDK_ROS_Demo/
lance@lance:~/HPS3D_SDK_ROS_Demo$
```

2）输入 catkin_make 编译工程

```
lance@lance:~/HPS3D_SDK_ROS_Demo$ catkin_make
Base path: /home/lance/HPS3D_SDK_ROS_Demo
Source space: /home/lance/HPS3D_SDK_ROS_Demo/src
Build space: /home/lance/HPS3D_SDK_ROS_Demo/build
Devel space: /home/lance/HPS3D_SDK_ROS_Demo/devel
Install space: /home/lance/HPS3D_SDK_ROS_Demo/install
####
```

等待编译成功

```
Scanning dependencies of target ros_camera_client
[ 50%] Building CXX object hps_camera/CMakeFiles/ros_camera_client.dir/src/ros_camera_client.cpp.o
[100%] Linking CXX executable /home/lance/HPS3D_SDK_ROS_Demo/devel/lib/hps_camera/ros_camera_client
[100%] Built target ros_camera_client
lance@lance:~/HPS3D_SDK_ROS_Demo$
```

（3）打开一个新终端输入 roscore，运行 ros 总线，如下图所示：

```
lance@lance:~$ roscore
... logging to /home/lance/.ros/log/1edd9a1a-6791-11eb-baf5-000c29543186/roslaun
ch-lance-18651.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://lance:44051/
ros_comm version 1.14.10


SUMMARY
========

PARAMETERS
```

（4）打开第二个新终端，输入 cd HPS3D_SDK_ROS_Demo/ 后输入 source devel/setup.bash，再输入 rosrun hps_camera ros_camera_client 启动程序，可以看见程序已经被正常启动。

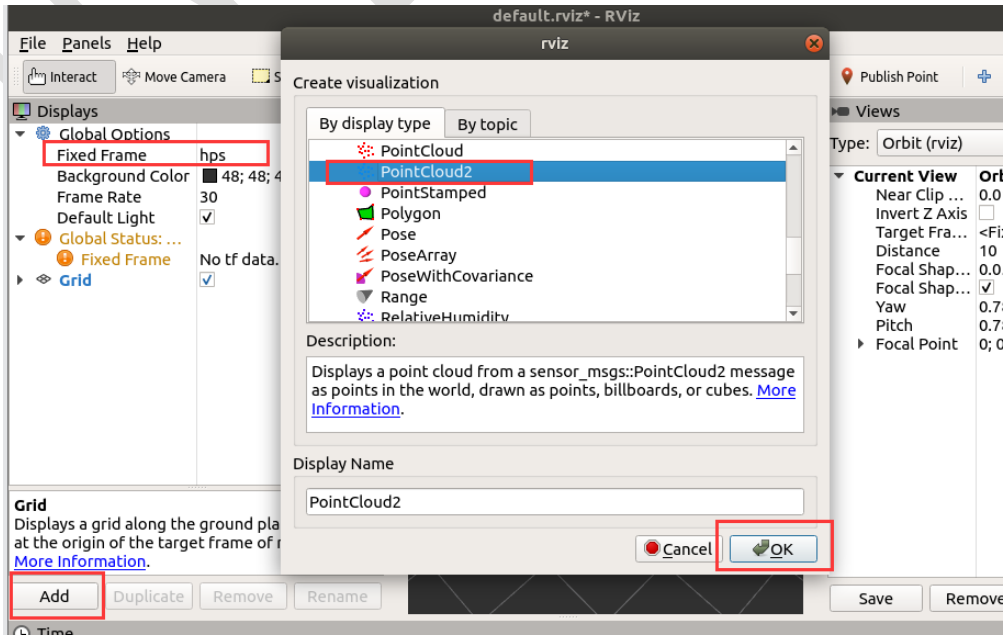
```
lance@lance:~$ cd HPS3D_SDK_ROS_Demo/
lance@lance:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
lance@lance:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_client
ip = 192.168.30.202,port = 12345
ConfigInit :1
HPS3D_ConnectByEthernet succeed!
HPS3D_SetOutputCallBack succeed!
HPS3D_SetWatchDogEnable succeed!
Program running...
Use Ctrl + C can exit the program!
FPS:32
FPS:34
FPS:35
FPS:35
FPS:35
```

（5）显示点云数据

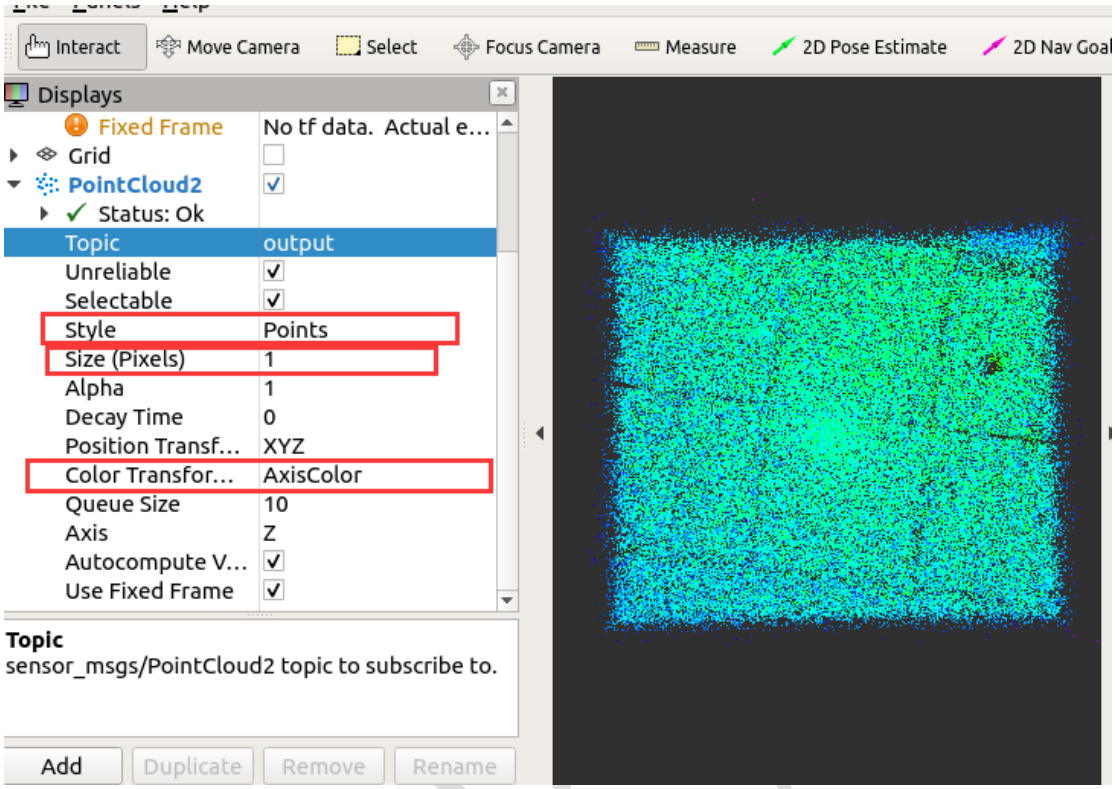
①开启第三个新终端，输入输入 cd HPS3D_SDK_ROS_Demo/ 后输入 source devel/setup.bash，再输入 rviz 启动 rviz 程序

```
lance@lance:~$ cd HPS3D_SDK_ROS_Demo/
lance@lance:~/HPS3D_SDK_ROS_Demo$ source devel/setup.bash
lance@lance:~/HPS3D_SDK_ROS_Demo$ rviz
[ INFO] [1612516442.674139600]: rviz version 1.13.15
[ INFO] [1612516442.674241422]: compiled against Qt version 5.9.5
[ INFO] [1612516442.674267020]: compiled against OGRE version 1.9.0 (Ghadamon)
[ INFO] [1612516442.677852490]: Forcing OpenGl version 0.
[ INFO] [1612516443.584272919]: Stereo is NOT SUPPORTED
[ INFO] [1612516443.584382455]: OpenGl version: 3.3 (GLSL 3.3).
```

②将 rviz 程序进行如下配置，添加 PointCloud2



③将 PointCloud2 进行如下配置，即可完成点云成像



三、API 函数接口

3.1 通过以太网连接设备

HPSLC_ConnectByEthernet

功能	通过以太网连接设备
函数定义	uint8_t __stdcall HPSLC_ConnectByEthernet(__IN char *server_ip, __IN uint16_t server_port, __OUT uint8_t* device_id);
参数	server_ip 设备 IP，ASCII 编码 server_port 设备端口号 device_id 设备 ID 号，设置失败时，ID 返回值为 99
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.必须与固件上设置的连接方式对应。 2.设备 IP 字符串采用 ASCII 编码格式

例：

```
uint8_t g_device_id = 99;
uint8_t ret = RET_OK;
//设置 IP 和端口号
ret = HPS3D_ConnectByEthernet((char*)"192.168.30.202", 12345, &g_device_id);
if (ret != RET_OK)
{
    printf("设备连接失败!\n");
    return RET_ERROR;
}
```

3.2 扫描设备的 ID 列表

HPS3D_ScanDeviceList

功能	扫描设备的 ID 列表
函数定义	uint8_t __stdcall HPS3D_ScanDeviceList(__OUT uint8_t* device_list, __OUT uint8_t* device_count);
参数	device_count 已经连接设备的数量 device_list 数组最少开辟 4 个空间，device_list[4]
返回	成功返回 RET_OK,失败返回错误描述码
注意	当只有一台设备连接时，之后的设备ID返回值为99。 例如：当只有一台设备连接时 device_count 返回值为1，device_list[0] = 当前已连接的设备ID device_list[1] = device_list[2] = device_list[3] = 99

例：

```
uint8_t device_list[4] = { 0 };
uint8_t device_count = 0;
uint8_t ret = RET_OK;
//扫描设备的 ID 列表
ret = HPS3D_ScanDeviceList(device_list, &device_count);
if (ret != RET_OK)
{
    printf("扫描设备ID列表失败!\n");
    return RET_ERROR;
}
```

3.3 断开设备

HPS3D_DisConnect

功能	断开设备
函数定义	uint8_t __stdcall HPS3D_DisConnect(__IN uint8_t device_id);
参数	device_id　　已经连接成功的设备ID
返回	成功返回 RET_OK,失败返回错误描述码
注意	程序运行完成后需要调用此接口断开连接，释放资源

例：

```
uint8_t ret = RET_OK;
//断开设备
ret = HPS3D_DisConnect(g_device_id);
if (ret != RET_OK)
{
    printf("断开设备失败!\n");
    return RET_ERROR;
}
```

3.4 Debug 信息回调函数原型

HPS3D_CALLBACK_DEBUG

功能	Debug 信息回调函数原型
函数定义	typedef void(__cdecl *HPS3D_CALLBACK_DEBUG)(char *str, void * contex);
参数	str　　输出的 DEBUG 信息 contex　　用户自定义参数，回调上下文，不需要时则置为 NULL

3.5 输出事件回调函数原型

HPS3D_CALLBACK_OUTPUT

功能	输出事件回调函数原型
函数定义	typedef void(__cdecl *HPS3D_CALLBACK_OUTPUT)(uint8_t out_id, uint8_t event, void *contex);
参数	out_id 输出事件的设备ID event 输出的事件 contex 用户自定义参数，回调上下文，不需要时则置为 NULL

3.6 设置 Debug 回调函数

HPS3D_SetDebugCallBack

功能	设置输出事件回调函数
函数定义	uint8_t __stdcall HPS3D_SetDebugCallBack(__IN HPS3D_CALLBACK_DEBUG call_back, __IN void *contex);
参数	call_back call_back设置为NULL时将停止该功能 contex 用户自定义参数，回调上下文，不需要时则置为NULL
返回	成功返回 RET_OK,失败返回错误描述码

例：

```
uint8_t ret = RET_OK;
ret =  HPS3D_SetDebugCallBack(debugFunc, NULL);
if (ret != RET_OK)
{
    printf("DEBUG信息回调函数注册失败!\n");
}
```

3.7 设置输出事件回调函数

HPS3D_SetOutputCallBack

功能	设置输出事件回调函数
函数定义	uint8_t __stdcall HPS3D_SetOutputCallBack(__IN HPS3D_CALLBACK_OUTPUT call_back, __IN uint8_t device_id, __IN void *contex);
参数	call_back call_back设置为NULL时将停止该功能 device_id 设备ID contex 用户自定义参数，回调上下文，不需要时则置为NULL
返回	成功返回 RET_OK,失败返回错误描述码
注意	

例：

```
uint8_t ret = RET_OK;
ret = HPS3D_SetOutputCallBack(OutputEventFunc, g_device_id, NULL);
if (ret != RET_OK)
{
    printf("输出事件回调函数注册失败!\n");
    return RET_ERROR;
}
```

3.8 获取平均距离（回调函数中使用）

HPS3D_GetAverageDistance_CB

功能	获取平均距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetAverageDistance_CB(__IN uint8_t device_id, __OUT uint16_t *aver_distance);
参数	device_id　　设备ID aver_distance　　返回平均距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_SIMPLEDISTANCERECVD 或者 EVENT_FULLDISTANCERECVD 或者 EVENT_FULLPOINTCLOUDRECVD 通知时调用

3.9 获取最大距离（回调函数中使用）

HPS3D_GetMaxDistance_CB

功能	获取最大距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetMaxDistance_CB(__IN uint8_t device_id, __OUT uint16_t *max_distance);
参数	device_id　　设备ID max_distance　　返回最大距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_SIMPLEDISTANCERECVD 或者 EVENT_FULLDISTANCERECVD 或者 EVENT_FULLPOINTCLOUDRECVD 通知时调用

3.10 获取最小距离（回调函数中使用）

HPS3D_GetMinDistance_CB

功能	获取最小距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetMinDistance_CB(__IN uint8_t device_id, __OUT uint16_t *min_distance);
参数	device_id 设备ID min_distance 返回最小距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_SIMPLEDISTANCERECVD 或者 EVENT_FULLDISTANCERECVD 或者 EVENT_FULLPOINTCLOUDRECVD 通知时调用

3.11 获取所有平均幅值（回调函数中使用）

HPS3D_GetAllAverAmplitude_CB

功能	获取所有平均幅值（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetAllAverAmplitude_CB(__IN uint8_t device_id, __OUT uint16_t *all_aver_amp);
参数	device_id 设备ID all_aver_amp 所有平均幅值（包括0幅值）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_SIMPLEDISTANCERECVD 或者 EVENT_FULLDISTANCERECVD 或者 EVENT_FULLPOINTCLOUDRECVD 通知时调用

3.12 获取有效的平均幅值（回调函数中使用）

HPS3D_GetValidAverAmplitude_CB

功能	获取有效的平均幅值（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetValidAverAmplitude_CB(__IN uint8_t device_id, __OUT uint16_t *valid_aver_amp);
参数	device_id 设备ID valid_aver_amp 返回有效的平均幅值（不包括0幅值）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_SIMPLEDISTANCERECVD 或者 EVENT_FULLDISTANCERECVD 或者 EVENT_FULLPOINTCLOUDRECVD 通知时调用

3.13 获取距离数据（回调函数中使用）

HPS3D_GetDistanceData_CB

功能	获取距离数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetDistanceData_CB(__IN uint8_t device_id, __OUT uint16_t *distance_data);
参数	device_id 设备ID distance_data 距离数据,单位毫米（需要开辟307200个空间）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_FULLDISTANCERECVD 通知时调用

3.14 获取灰度数据（回调函数中使用）

HPS3D_GetGrayscaleData_CB

功能	获取灰度数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetGrayscaleData_CB(__IN uint8_t device_id, __OUT uint8_t *grayscale_data);
参数	device_id 设备ID grayscale_data 灰度数据,单位灰度值（范围0~255）（需要开辟307200个空间）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_AMPLITUDE 通知时调用

3.15 获取幅值数据（回调函数中使用）

HPS3D_GetAmplitudeData_CB

功能	获取幅值数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetAmplitudeData_CB(__IN uint8_t device_id, __OUT uint16_t *amplitude_data);
参数	device_id 设备ID amplitude_data 幅值数据,（需要开辟307200个空间）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_AMPLITUDE 通知时调用

3.16 获取最大幅值数据（回调函数中使用）

HPS3D_GetMaxAmplitudeData_CB

功能	获取最大幅值数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetMaxAmplitudeData_CB(__IN uint8_t device_id, __OUT uint16_t *max_amplitude);
参数	device_id 设备ID max_amplitude 最大幅值数据
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_AMPLITUDE 通知时调用

3.17 获取最小幅值数据（回调函数中使用）

HPS3D_GetMinAmplitudeData_CB

功能	获取最小幅值数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetMinAmplitudeData_CB(__IN uint8_t device_id, __OUT uint16_t *min_amplitude);
参数	device_id 设备ID min_amplitude 最小幅值数据
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_AMPLITUDE 通知时调用

3.18 获取点云数据（回调函数中使用）

HPS3D_GetPointCloudData_CB

功能	获取点云数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetPointCloudData_CB(__IN uint8_t device_id, __OUT double(*point_data)[3], __OUT uint32_t *point_count);
参数	device_id 设备ID point_data 点云数据,定义方式,point_data[307200][3] point_count 有效点的数量
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_FULLPOINTCLOUDRECVD 或者 EVENT_ROIPOINTCLOUDRECVD 通知时调用 3.point_data中，[3]的值：0表示X轴值，1表示Y轴值，2表示Z轴值 4.其中，无效点已经被自动过滤，不会输出

3.19 获取输出的 ROI 参数信息（回调函数中使用）

HPS3D_GetOutRoiParam_CB

功能	获取输出的 ROI 参数信息（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiParam_CB(__IN uint8_t device_id, __OUT uint8_t *group_id, __OUT uint8_t *roi_number, __OUT uint8_t *roi_id);
参数	device_id 设备ID group_id 返回ROI组ID roi_number 返回ROI个数 roi_id 返回ROI的ID，需要开辟8个空间，roi_id[8]
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用

3.20 获取输出 ROI 的点云数据（回调函数中使用）

HPS3D_GetOutRoiPointCloudData_CB

功能	获取输出 ROI 的点云数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiPointCloudData_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT double(*roi_point_data)[3], __OUT uint32_t *roi_point_count);
参数	device_id 设备ID roi_id Roi的ID roi_point_data 点云数据,定义方式,point_data[307200][3] roi_point_count 返回有效点的数量
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 通知时调用 3.roi_point_data中，[3]的值：0表示X轴值，1表示Y轴值，2表示Z轴值 4.其中，无效点已经被自动过滤，不会输出 5.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.21 获取输出的 ROI 深度数据（回调函数中使用）

HPS3D_GetOutRoiDistanceData_CB

功能	获取输出的 ROI 深度数据（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiDistanceData_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *roi_distance_data, __OUT uint16_t *roi_area);
参数	device_id 设备ID roi_id Roi的ID roi_distance_data ROI的深度数据，单位毫米，（需要开辟307200个空间） roi_area ROI的范围，需要开辟4个空间，roi_area[4]
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIFULLDISTANCERECVD 通知时调用 3.roi_area[0]：表示左上角点的X坐标 roi_area[1]：表示左上角点的Y坐标 roi_area[2]：表示右下角点的X坐标 roi_area[3]：标志右下角点的Y坐标 4.该区域的像素个数计算公式为： (right_bottom_x - left_top_x + 1) * (right_bottom_y - left_top_y + 1)); pixel_number = (roi_area[2] - roi_area[0] + 1) * (roi_area[3] - roi_area[1] + 1) 5.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.22 获取输出的 ROI 平均距离（回调函数中使用）

HPS3D_GetOutRoiAverageDistance_CB

功能	获取输出的 ROI 平均距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiAverageDistance_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *aver_distance);
参数	device_id 设备ID roi_id Roi的ID aver_distance 返回平均距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用 3.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.23 获取输出的 ROI 最大距离（回调函数中使用）

HPS3D_GetOutRoiMaxDistance_CB

功能	获取输出的 ROI 最大距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiMaxDistance_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *max_distance);
参数	device_id 设备ID roi_id Roi的ID max_distance 返回最大距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用 3.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.24 获取输出的 ROI 最小距离（回调函数中使用）

HPS3D_GetOutRoiMinDistance_CB

功能	获取输出的 ROI 最小距离（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiMinDistance_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *min_distance);
参数	device_id 设备ID roi_id Roi的ID min_distance 返回最小距离，单位毫米
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用 3.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.25 获取输出的 ROI 平均幅值（回调函数中使用）

HPS3D_GetOutRoiAllAverAmplitude_CB

功能	获取输出的 ROI 平均幅值（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiAllAverAmplitude_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *all_aver_amp);
参数	device_id 设备ID roi_id Roi的ID all_aver_amp 返回平均幅值（包括0幅值）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

	2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用 3.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.26 获取输出的 ROI 有效平均幅值（回调函数中使用）

HPS3D_GetOutRoiValidAverAmplitude_CB

功能	获取输出的 ROI 有效平均幅值（回调函数中使用）
函数定义	uint8_t __stdcall HPS3D_GetOutRoiValidAverAmplitude_CB(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT uint16_t *valid_aver_amp);
参数	device_id 设备ID roi_id Roi的ID valid_aver_amp 返回有效平均幅值（不包括0幅值）
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.该接口只能在回调函数中收到 EVENT_ROIPOINTCLOUDRECVD 或者 EVENT_ROIFULLDISTANCERECVD 或者 EVENT_ROISIMPLEDISTANCERECVD通知时调用 3.调用前先调用 HPS3D_GetOutRoiParam_CB 接口获取对应的输入参数

3.27 设置运行模式

HPS3D_SetRunMode

功能	设置运行模式
函数定义	uint8_t __stdcall HPS3D_SetRunMode(__IN uint8_t device_id, __IN uint8_t run_mode);
参数	device_id 设备ID run_mode 运行模式，RUN_IDLE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.run_mode 参照 HPS3D_DEFINE.h 中的定义 3.每次设置为 RUN_SINGLE_SHOT 将通过回调函数吐一帧数据

例：

```
uint8_t ret = RET_OK;
ret =  HPS3D_SetRunMode(g_device_id, RUN_CONTINUOUS);
if (ret != RET_OK)
{
    printf("设置运行模式失败!\n");
}
```

3.28 获取运行模式

HPS3D_GetRunMode

功能	获取运行模式
函数定义	uint8_t __stdcall HPS3D_GetRunMode(__IN uint8_t device_id, __OUT uint8_t *run_mode);
参数	device_id 设备ID run_mode 返回运行模式，RUN_IDLE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.run_mode 参照 HPS3D_DEFINE.h 中的定义

例：

```
uint8_t ret = RET_OK;
uint8_t run_mode = RUN_IDLE;
ret =  HPS3D_GetRunMode(g_device_id, &run_mode);
if (ret != RET_OK)
{
    printf("设置运行模式失败!\n");
}
```

3.29 设置点云模式配置

HPS3D_SetPointCloudMode

功能	设置点云模式配置
函数定义	uint8_t __stdcall HPS3D_SetPointCloudMode(__IN uint8_t device_id, __IN bool point_cloud_enable, __IN uint8_t mirror_mode);
参数	device_id 设备ID point_cloud_enable 点云图使能 mirror_mode 点云图镜像模式,MIRROR_DISABLE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.mirror_mode 参照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t ret = RET_OK;
ret = HPS3D_SetPointCloudMode(g_device_id, true, MIRROR_DISABLE);
if (ret != RET_OK)
{
    printf("设置点云模式配置失败!\n");
}
```

3.30 获取点云模式配置

HPS3D_GetPointCloudMode

功能	获取点云模式配置
函数定义	uint8_t __stdcall HPS3D_GetPointCloudMode(__IN uint8_t device_id, __OUT bool *point_cloud_enable, __OUT uint8_t *mirror_mode);
参数	device_id 设备ID point_cloud_enable 点云图使能 mirror_mode 点云图镜像模式,MIRROR_DISABLE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.mirror_mode 参照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t ret = RET_OK;
uint8_t mirror_mode = MIRROR_DISABLE;
bool enable = false;
ret = HPS3D_GetPointCloudMode(g_device_id, &enable, &mirror_mode);
if (ret != RET_OK)
{
    printf("获取点云模式配置失败!\n");
}
```

3.31 设置测量模式

HPS3D_SetMeasureMode

功能	设置测量模式
函数定义	uint8_t __stdcall HPS3D_SetMeasureMode(__IN uint8_t device_id, __IN uint8_t measure_mode);
参数	device_id 设备ID measure_mode 测量模式，DSITMODE_GENERAL
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.asure_mode 参照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t ret = RET_OK;
ret = HPS3D_SetMeasureMode(g_device_id, DSITMODE_GENERAL);
if (ret != RET_OK)
{
    printf("设置测量模式失败!\n");
}
```

3.32 获取测量模式

HPS3D_GetMeasureMode

功能	获取测量模式
函数定义	uint8_t __stdcall HPS3D_GetMeasureMode(__IN uint8_t device_id, __OUT uint8_t *measure_mode);
参数	device_id 设备ID measure_mode 返回测量模式，DSITMODE_GENERAL
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.measure_mode 照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t ret = RET_OK;
uint8_t measure_mode = DSITMODE_GENERAL;
ret = HPS3D_GetMeasureMode(g_device_id, &measure_mode);
if (ret != RET_OK)
{
    printf("获取测量模式失败!\n");
}
```

3.33 获取 SDK 版本信息

HPS3D_GetSDKVersion

功能	获取测量模式
函数定义	uint8_t __stdcall HPS3D_GetSDKVersion(__OUT uint8_t *version);
参数	version[0] year version[1] month; version[2] day; version[3] major; version[4] minor; version[5] rev;
返回	成功返回 RET_OK,失败返回错误描述码
注意	version数组最少开辟6个空间，version[6]

例：

```
uint8_t sdk_version[6] = { 0 };
ret = HPS3D_GetSDKVersion(sdk_version);
if (ret != RET_OK)
{
    printf("SDK版本信息获取失败!\n");
}
else
```

```
    {
        printf("SDK版本:20%d-%d-%d V%d.%d.%d\n", sdk_version[0], sdk_version[1]
            , sdk_version[2], sdk_version[3], sdk_version[4], sdk_version[5]);
    }
```

3.34 获取设备版本信息

HPS3D_GetDeviceVersion

功能	获取设备版本信息
函数定义	uint8_t __stdcall HPS3D_GetDeviceVersion(__IN uint8_t device_id, __OUT uint8_t *version);
参数	device_id 设备ID version[0] year version[1] month; version[2] day; version[3] major; version[4] minor; version[5] rev;
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.version数组最少开辟6个空间，version[6] 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
    uint8_t device_version[6] = { 0 };
    ret = HPS3D_GetDeviceVersion(g_device_id, device_version);
    if (ret != RET_OK)
    {
        printf("设备版本信息获取:失败!\n");
    }
    else
    {
        printf("设备版本:20%d-%d-%d V%d.%d.%d\n", device_version[0],
device_version[1], device_version[2], device_version[3], device_version[4],
device_version[5]);
    }
```

3.35 获取设备序列号

HPS3D_GetDeviceSN

功能	获取设备序列号
函数定义	uint8_t __stdcall HPS3D_GetDeviceSN(__IN uint8_t device_id, __OUT char * SN);
参数	device_id 设备ID SN 返回设备序列号
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.SN数组最少开辟64个空间，SN[64] 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
char SN[64] = { 0 };
ret = HPS3D_GetDeviceSN(g_device_id, SN);
if (ret != RET_OK)
{
    printf("设备序列号获取失败!\n");
}
else
{
    printf("设备序列号:%s\n", SN);
}
```

3.36 设置输出数据类型

HPS3D_SetOutputDataType

功能	设置输出数据类型
函数定义	uint8_t __stdcall HPS3D_SetOutputDataType(__IN uint8_t device_id, __IN uint8_t data_type);
参数	device_id 设备ID data_type 返回输出数据类型,OUTPUT_AMPLITUDE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.data_type 照 HPS3D_DEFINE.h 中的定义 3.OUTPUT_AMPLITUDE，将通过回调函数输出振幅数据 OUTPUT_DISTANCE_FULL，将通过回调函数输出完整深度数据，包含各像素点数据 OUTPUT_DISTANCE_SIMPLE，将通过回调函数输出简单深度数据，不包含各像素点数据，只包含数学统计数据 4.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SetOutputDataType(g_device_id, OUTPUT_AMPLITUDE);
if (ret != RET_OK)
{
    printf("设置输出数据类型失败!\n");
}
```

3.37 获取输出数据类型

HPS3D_GetOutputDataType

功能	获取输出数据类型
函数定义	uint8_t __stdcall HPS3D_GetOutputDataType(__IN uint8_t device_id, __OUT uint8_t *data_type);
参数	device_id　　设备ID data_type　　返回输出数据类型,OUTPUT_AMPLITUDE
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.data_type 照 HPS3D_DEFINE.h 中的定义 3.OUTPUT_AMPLITUDE，将通过回调函数输出振幅数据 OUTPUT_DISTANCE_FULL，将通过回调函数输出完整深度数据，包含各像素点数据 OUTPUT_DISTANCE_SIMPLE，将通过回调函数输出简单深度数据，不包含各像素点数据，只包含数学统计数据 4.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t data_type = OUTPUT_AMPLITUDE;
ret = HPS3D_GetOutputDataType(g_device_id, &data_type);
if (ret != RET_OK)
{
    printf("获取输出数据类型失败!\n");
}
```

3.38 设置积分时间

HPS3D_SetIntegTime

功能	设置积分时间
函数定义	uint8_t __stdcall HPS3D_SetIntegTime(__IN uint8_t device_id, __IN uint32_t time_us);
参数	device_id　　设备ID time_us　　积分时间(单位us，范围50~1000)
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SetIntegTime(g_device_id, 200);
if (ret != RET_OK)
{
    printf("设置积分时间失败!\n");
}
```

3.39 获取积分时间

HPS3D_GetIntegTime

功能	获取积分时间
函数定义	uint8_t __stdcall HPS3D_GetIntegTime(__IN uint8_t device_id, __OUT uint32_t *time_us);
参数	device_id 设备ID time_us 积分时间(单位us，范围50~1000)
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint32_t time_us = 0;
ret = HPS3D_GetIntegTime(g_device_id, &time_us);
if (ret != RET_OK)
{
    printf("获取积分时间失败!\n");
}
```

3.40 保存到用户设置表

HPS3D_ProfileSaveToUser

功能	保存到用户设置表
函数定义	uint8_t __stdcall HPS3D_ProfileSaveToUser(__IN uint8_t device_id);
参数	device_id 设备ID
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_ProfileSaveToUser(g_device_id);
if (ret != RET_OK)
{
    printf("保存到用户设置表失败!\n");
}
```

3.41 清除用户设置表

HPS3D_ProfileClearUser

功能	保存到用户设置表
函数定义	uint8_t __stdcall HPS3D_ProfileClearUser(__IN uint8_t device_id);
参数	device_id 设备ID
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_ProfileClearUser(g_device_id);
if (ret != RET_OK)
{
    printf("清除用户设置表失败!\n");
}
```

3.42 还原出厂设置

HPS3D_ProfileRestoreFactory

功能	还原出厂设置
函数定义	uint8_t __stdcall HPS3D_ProfileRestoreFactory(__IN uint8_t device_id);
参数	device_id 设备ID
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_ProfileRestoreFactory(g_device_id);
if (ret != RET_OK)
{
    printf("还原出厂设置失败!\n");
}
```

3.43 选择 ROI 组

HPS3D_SetROIGroup

功能	选择 ROI 组
函数定义	uint8_t __stdcall HPS3D_SetROIGroup(__IN uint8_t device_id, __IN uint8_t group_id);
参数	device_id 设备ID group_id ROI组的ID,0~1

返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SetROIGroup(g_device_id, 1);
if (ret != RET_OK)
{
    printf("ROI组设置失败!\n");
}
```

3.44 获取当前的 ROI 组 ID

HPS3D_GetROIGroup

功能	获取当前的 ROI 组 ID
函数定义	uint8_t __stdcall HPS3D_GetROIGroup(__IN uint8_t device_id, __OUT uint8_t *group_id);
参数	device_id 设备ID group_id ROI组的ID,0~1
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t group_id = 0;
ret = HPS3D_GetROIGroup(g_device_id, &group_id);
if (ret != RET_OK)
{
    printf("ROI组获取失败!\n");
}
```

3.45 设置 ROI 使能

HPS3D_SetROIEnable

功能	设置 ROI 使能
函数定义	uint8_t __stdcall HPS3D_SetROIEnable(__IN uint8_t device_id, __IN uint8_t roi_id, __IN bool enable);
参数	device_id 设备ID roi_id ROI的ID enable ROI使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SetROIEnable(g_device_id, 1, true);
if (ret != RET_OK)
{
    printf("设置ROI使能失败!\n");
}
```

3.46 获取 ROI 使能

HPS3D_GetROIEnable

功能	获取 ROI 使能
函数定义	uint8_t __stdcall HPS3D_GetROIEnable(__IN uint8_t device_id, __IN uint8_t roi_id, __OUT bool *enable);
参数	device_id 设备ID roi_id ROI的ID enable ROI使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
bool enable = false;
ret = HPS3D_GetROIEnable(g_device_id, 1, &enable);
if (ret != RET_OK)
{
    printf("获取ROI使能失败!\n");
}
```

3.47 获取当前设备支持的 ROI 数量和阈值数量

HPS3D_GetNumberOfROI

功能	获取当前设备支持的 ROI 数量和阈值数量
函数定义	uint8_t __stdcall HPS3D_GetNumberOfROI(__IN uint8_t device_id, __OUT uint8_t *roi_number, __OUT uint8_t *threshold_number, __OUT uint8_t *roi_number_group);
参数	device_id 设备ID roi_number 支持的ROI数量 threshold_number 支持的阈值数量 roi_number_group 支持的ROI组数量
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
    uint8_t roi_number = 0;
    uint8_t threshold_number = 0;
    uint8_t roi_number_group = 0;
    ret = HPS3D_GetNumberOfROI(g_device_id, &roi_number, &threshold_number,
&roi_number_group);
    if (ret != RET_OK)
    {
        printf("获取失败!\n");
    }
```

3.48 设置平滑滤波器

HPS3D_SetSmoothFilter

功能	设置平滑滤波器
函数定义	uint8_t __stdcall HPS3D_SetSmoothFilter(__IN uint8_t device_id, __IN uint8_t filter_type, __IN uint32_t filter_times);
参数	device_id 设备ID filter_type 平滑滤波器类型，SMOOTH_FILTER_DISABLE filter_times 平滑滤波器滤波次数，范围0~10
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.filter_type参照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
    ret = HPS3D_SetSmoothFilter(g_device_id, SMOOTH_FILTER_DISABLE, 1);
    if (ret != RET_OK)
    {
        printf("设置平滑滤波器失败!\n");
    }
```

3.49 获取平滑滤波器设置

HPS3D_GetSmoothFilter

功能	获取平滑滤波器设置
函数定义	uint8_t __stdcall HPS3D_GetSmoothFilter(__IN uint8_t device_id, __OUT uint8_t *filter_type, __OUT uint32_t *filter_times);
参数	device_id 设备ID filter_type 平滑滤波器类型，SMOOTH_FILTER_DISABLE filter_times 平滑滤波器滤波次数，范围0~10
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

	2.filter_type参照 HPS3D_DEFINE.h 中的定义 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
uint8_t filter_type = SMOOTH_FILTER_DISABLE;
uint8_t filter_times = 0;
ret = HPS3D_GetSmoothFilter(g_device_id, &filter_type, &filter_times);
if (ret != RET_OK)
{
    printf("获取平滑滤波器设置失败!\n");
}
```

3.50 设置卡尔曼滤波器

HPS3D_SetSimpleKalmanFilter

功能	设置卡尔曼滤波器
函数定义	uint8_t __stdcall HPS3D_SetSimpleKalmanFilter(__IN uint8_t device_id, __IN bool kalman_enable, __IN float64_t kalman_k, __IN uint32_t kalman_number, __IN uint32_t kalman_threshold);
参数	device_id 设备ID kalman_enable 卡尔曼滤波器使能 kalman_k 卡尔曼滤波器比例系数 kalman_number 卡尔曼滤波器阈值检查帧数 kalman_threshold 卡尔曼滤波器噪声阈值
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.enable使能设置为false后其他值将不会被写入，可设置为NULL 3.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SetSimpleKalmanFilter(g_device_id, false, 0, 0, 0);
if (ret != RET_OK)
{
    printf("设置卡尔曼滤波器失败!\n");
}
```

3.51 获取卡尔曼滤波器配置

HPS3D_GetSimpleKalmanFilter

功能	获取卡尔曼滤波器配置
函数定义	uint8_t __stdcall HPS3D_GetSimpleKalmanFilter(__IN uint8_t device_id, __OUT bool *kalman_enable, __OUT float64_t *kalman_k, __OUT uint32_t *kalman_number, __OUT uint32_t *kalman_threshold);

参数	device_id 设备ID kalman_enable 返回卡尔曼滤波器使能 kalman_k 返回卡尔曼滤波器比例系数 kalman_number 返回卡尔曼滤波器阈值检查帧数 kalman_threshold 返回卡尔曼滤波器噪声阈值
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
    bool kalman_enable = false;
    float64_t kalman_k = 0.0;
    uint32_t kalman_number  = 0;
    uint32_t kalman_threshold = 0;
    ret = HPS3D_GetSimpleKalmanFilter(g_device_id, &kalman_enable, &kalman_k,
&kalman_number, &kalman_threshold);
    if (ret != RET_OK)
    {
        printf("获取卡尔曼滤波器配置失败!\n");
    }
```

3.52 设置距离偏移

HPS3D_SetOffset

功能	设置距离偏移
函数定义	uint8_t __stdcall HPS3D_SetOffset(__IN uint8_t device_id, __IN int16_t offset_mm);
参数	device_id 设备ID offset_mm 偏移距离，单位MM
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
    ret = HPS3D_SetOffset(g_device_id, 0);
    if (ret != RET_OK)
    {
        printf("设置距离偏移失败!\n");
    }
```

3.53 获取距离偏移

HPS3D_GetOffset

功能	获取距离偏移
函数定义	uint8_t __stdcall HPS3D_GetOffset(__IN uint8_t device_id, __OUT int16_t *offset_mm);
参数	device_id 设备ID offset_mm 偏移距离，单位MM
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
int16_t offset_mm = 0;
ret = HPS3D_GetOffset(g_device_id, &offset_mm);
if (ret != RET_OK)
{
    printf("获取距离偏移失败!\n");
}
```

3.54 软件复位

HPS3D_SoftwareReset

功能	软件复位
函数定义	uint8_t __stdcall HPS3D_SoftwareReset(__IN uint8_t device_id);
参数	device_id 设备ID
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
ret = HPS3D_SoftwareReset(g_device_id);
if (ret != RET_OK)
{
    printf("软件复位失败!\n");
}
```

3.55 设置看门狗使能

HPS3D_SetWatchDogEnable

功能	设置看门狗使能
函数定义	uint8_t __stdcall HPS3D_SetWatchDogEnable(__IN uint8_t device_id, __IN bool enable);
参数	device_id 设备ID enable 看门狗使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE 3.单步调试程序时需要将看门狗使能关闭，否则可能会导致设备断连 4.正式运行程序时需要将看门狗使能打开

例：

```
ret = HPS3D_SetWatchDogEnable(g_device_id, false);
if (ret != RET_OK)
{
    printf("设置看门狗使能失败!\n");
}
```

3.56 获取看门狗使能

HPS3D_GetWatchDogEnable

功能	获取看门狗使能
函数定义	uint8_t __stdcall HPS3D_GetWatchDogEnable(__IN uint8_t device_id, __OUT bool *enable);
参数	device_id 设备ID enable 看门狗使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应 2.执行前必须将运行模式设置为待机模式，RUN_IDLE

例：

```
bool enable = false;
ret = HPS3D_GetWatchDogEnable(g_device_id, &enable);
if (ret != RET_OK)
{
    printf("获取看门狗使能失败!\n");
}
```

3.57 设置边缘滤波器

HPS3D_SetEdgeFilter

功能	设置边缘滤波器
函数定义	uint8_t __stdcall HPS3D_SetEdgeFilter(__IN uint8_t device_id, __IN bool enable, __IN uint32_t args);
参数	device_id 设备ID enable 边缘滤波器使能状态 args 边缘滤波器过滤参数，单位MM，默认1000
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

例：

```
ret = HPS3D_SetEdgeFilter (g_device_id, false, 200);
if (ret != RET_OK)
{
    printf("设置边缘滤波器失败!\n");
}
```

3.58 获取边缘滤波器设置

HPS3D_GetEdgeFilter

功能	获取边缘滤波器设置
函数定义	uint8_t __stdcall HPS3D_GetEdgeFilter(__IN uint8_t device_id, __OUT bool *enable, __OUT uint32_t *args);
参数	device_id 设备ID enable 边缘滤波器使能状态 args 边缘滤波器过滤参数，单位MM，默认1000
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

例：

```
bool enable = false;
ret = HPS3D_GetEdgeFilter(g_device_id, &enable, 200);
if (ret != RET_OK)
{
    printf("获取边缘滤波器设置失败!\n");
}
```

3.59 设置 HDR 模式使能

HPS3D_SetHDREnable

功能	设置 HDR 模式使能
函数定义	uint8_t __stdcall HPS3D_SetHDREnable(__IN uint8_t device_id, __IN bool hdr_enable);
参数	device_id 设备ID hdr_enable HDR模式使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

例:

```
bool hdr_enable = false;
ret = HPS3D_SetHDREnable(g_device_id, hdr_enable);
if (ret != RET_OK)
{
    printf("设置HDR模式使能失败!\n");
}
```

3.60 获取 HDR 模式使能

HPS3D_GetHDREnable

功能	设置 HDR 模式使能
函数定义	uint8_t __stdcall HPS3D_GetHDREnable(__IN uint8_t device_id, __OUT bool *hdr_enable);
参数	device_id 设备ID hdr_enable HDR模式使能
返回	成功返回 RET_OK,失败返回错误描述码
注意	1.device_id必须与连接时返回的ID对应

例:

```
bool hdr_enable = false;
ret = HPS3D_GetHDREnable(g_device_id, &hdr_enable);
if (ret != RET_OK)
{
    printf("获取HDR模式使能失败!\n");
}
```

四、修订历史纪录

Date	Revision	Description
2021/02/27	1.0.0	首次修订

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.