HPS-3D160 Solid-state Depth camera

SDK manual

Hypersen

HYPERSEN TECHNOLOGIES CO., LTD. 海伯森技术

目录

1. SDK introduction

The SDK provides the application interface of the HPS3D160 Solid-State Depth camera, which is currently available on the Linux platform, the Windows platform, the ROS platform, and most of the microcontrollers that do not run the operating system; the SDK is a secondary development kit tool, and the interface provided includes Most of the operating instructions of the HPS3D160 Solid-State Depth camera developed by our company, please read this manual carefully;

2. Integrate SDK into IDE

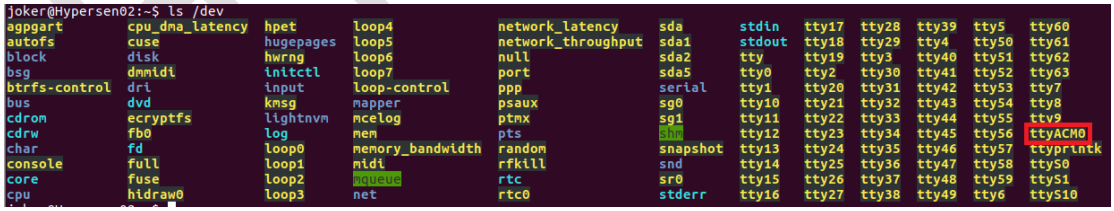
Currently, the IDE environment for embedded programs can be roughly divided into three types: Keil, IAR, and Eclips-based IDEs (such as TrueStudio, Simplicity Studio, etc.). Each project management strategy is inconsistent, but the difference is not very large. The following are different platforms. Integration of the SDK;

2.1 Environment configuration and integration into the IDE under Linux platform

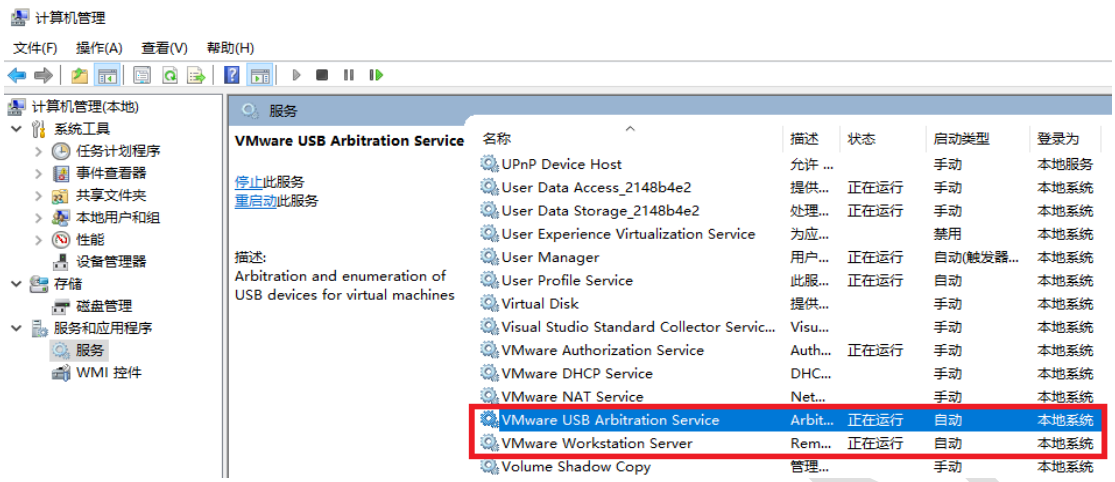
xxx.so is suitable to use on Linux operation system, take Ubuntu as example. This example is based on the SDK with API version number 2018.12.04 V1.0.3

2.1.1 HPS3D160 device connection

Connect the HPS3D160 device to your computer, open the terminal and type ls /dev to view the device ttyACM*, as shown below:

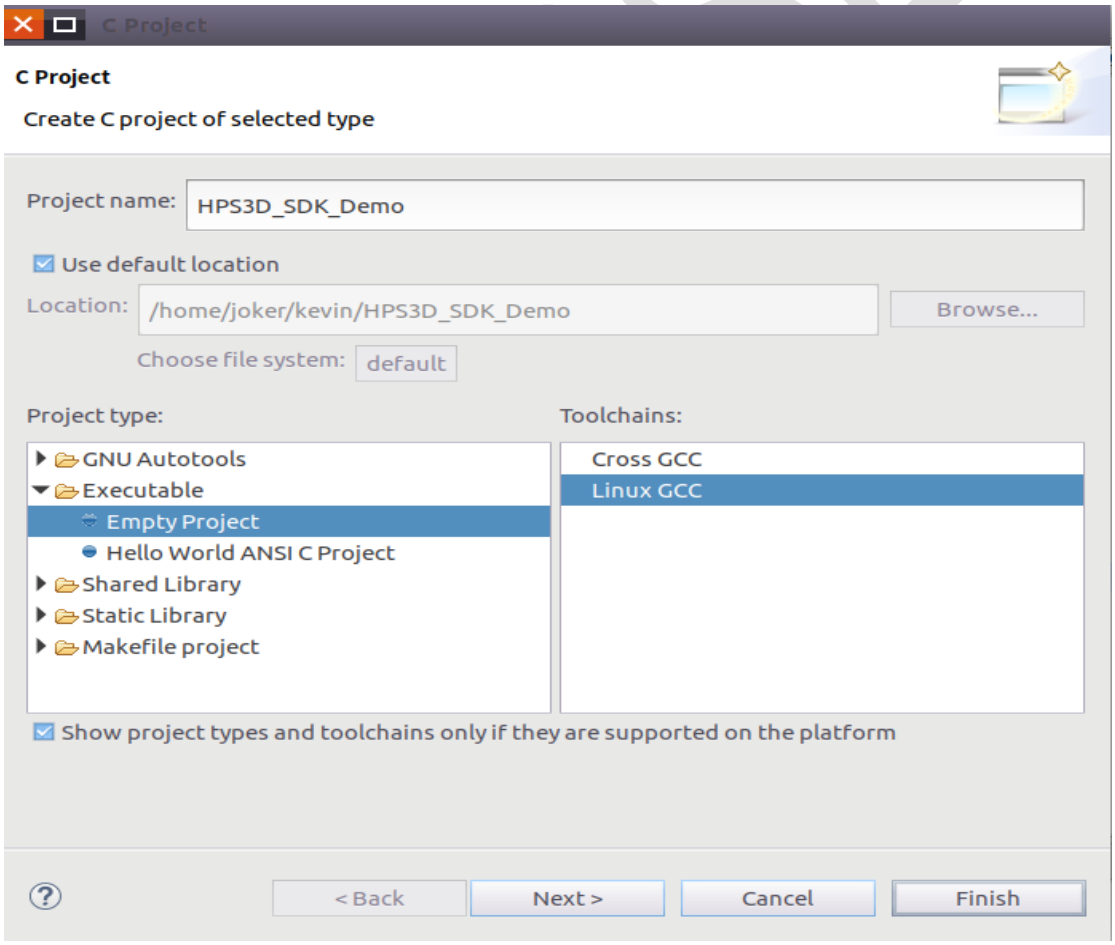


If you do not see the ttyACM* device name, you need to re-plug and view it again. If not, go to "Computer Management -> Services and Applications -> Services" to see if the VMware USB Arbitration Service is running. If disabled, then turn on the operation, then re-plug the device; if you don't want to start the USB device service every time you log in to the virtual machine, you can set the VMware Workstation Server and VMware USB Arbitration Service to run and auto, restart the computer.

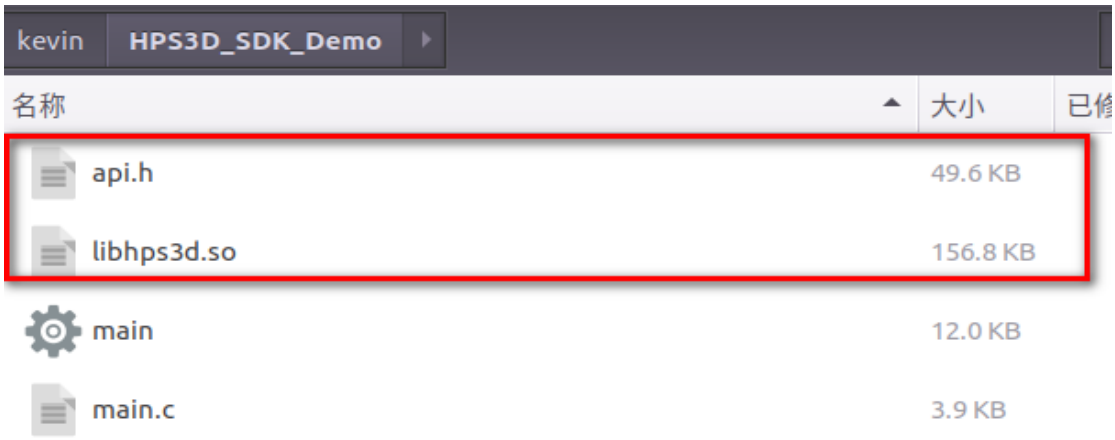


2.1.2 Engineering environment configuration and integration

In ubuntu eclipse build project (other IDE tools are available), here eclipse as an example, after the following configuration, click finish to complete the project.



Copy the api.h and libhps3d.so files into the project directory;



Open the terminal, type sudo cp libhps3d.so /usr/local/lib/, copy libhps3d.so to the /usr/local/lib/ directory; then execute sudo ldconfig to load it.

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

After writing the test code in main.c, type: gcc main.c -L./ -lhps3d -o app in the terminal, compile the connection, and then use sudo ./app to execute the program.

```
@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
```

Select the connectable device. After the initialization is successful, the measurement result can be output normally:

```
distance average:1619
distance average:1643
distance average:1637
distance average:1626
distance average:1623
distance average:1643
distance average:1722
distance average:1613
distance average:1633
distance average:1633
distance average:1603
distance average:1632
distance average:1631
```

2.1.3 Use SDK in user project

1. In project, it includes head file, example code:

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

2. Call the device connection function, before you need to enter the handle->DeviceName device name path, this function gets the handle->DeviceFd device file descriptor and handle->ConnectStatus connection status (true). The Handle->DeviceFd device file descriptor is required for all command function interfaces. Sample code:

```
HPS3D_HandleTypeDef handle;
RET_StatusTypeDef ret = RET_OK;
ret = HPS3D_Connect(&handle);
if(RET_OK != ret)
{
    printf("Connect Failed!  ret = %d\n",ret);
    break;
}
```

3. call the initialization configuration function, before this step, you need to perform step 2, using the handle->DeviceFd device file descriptor obtained in step 2, after calling this function will initialize the device configuration, the default configuration is standby mode and asynchronous communication mode, And create an asynchronous communication thread; get the handle->DeviceAddr device address, handle->RoiNumber device support ROI number, handle->ThresholdNumber device ROI area support threshold number, handle->opticalEnable optical enable parameter (for 3D Point cloud space coordinate transformation), and dynamically allocate space for MeasureData. Sample code:

```
/*Device initialization*/
ret = HPS3D_ConfigInit(&handle);
if(RET_OK != ret)
{
    printf("Initialization failed! error code is:%d\n", ret);
    break;
}
printf("Initialization succeed!\n");
```

Note:

(1) Step 2: The handle->DeviceFd device file descriptor returned by the device function and the handle->DeviceAddr device address returned by the initialization function in step 3 are used in all command function interfaces.

(2) Be sure to check the return value of the initialization function and determine whether the initialization is successful based on the return value. Most of the SDK's APIs provide operational state return. It is recommended that users check the return value of each API to ensure reliability.

4. According to the user's need to call other command function interface, configure the camera, and set the running mode after the configuration is completed. After being configured as a continuous measurement command, the command return value of the set operation mode may not be detected. In this case, the return value of the function can be ignored; sample code:

```
/*Set to continuous measurement mode*/
handle.RunMode = RUN_CONTINUOUS;
HPS3D_SetRunMode(&handle);
```

Note:

(1) When all command function interfaces are configured, a stop measurement command will be sent. Therefore, in the continuous measurement process, after calling the command function interface, you need to reset the operation mode.

(2) In the operation mode configuration, three modes can be selected:

①RUN_IDLE: Idle mode, in which the camera enters standby mode.

②RUN_SINGLE_SHOT: Single measurement mode, in which the camera takes a measurement and then automatically switches to RUN_IDLE. There are two ways to measure, one is synchronous measurement and the other is asynchronous measurement (default is asynchronous).

③ RUN_CONTINUOUS: Continuous measurement mode, in which the camera will automatically perform continuous measurement until the user manually sets it to RUN_IDLE or RUN_SINGLE_SHOT mode; the measurement data is returned by the structure form, refer to the MeasureDataTypeDef structure, and the data type is enumerated by returning the packet type. (RetPacketTypedef) makes a distinction.

5. After all the steps are successfully configured, there are two ways to obtain measurement data, one is synchronous measurement (only one measurement is supported), and the other is asynchronous (supports single measurement and continuous measurement).

(1) Synchronous measurement (only one measurement is supported), and the HPS3D_SingleMeasurement function in api.h is called to perform a single measurement. The sample code is as follows:

```
HPS3D_SingleMeasurement(&handle);
printf("distance average:%d\n",handle.MeasureData.full_depth_data->distance_average);
```

Note：

In the HPS3D_SingleMeasurement function, the communication mode handle.SyncMode is set to the synchronous mode SYNC. If it is set to the asynchronous mode, the handle.SyncMode is set to the asynchronous mode ASYNC after the function.

（2）Asynchronous measure(Support single measure and continuously measure)

①Set operation mode, code is as follow:

```
/*Set to continuous measurement mode*/
handle.RunMode = RUN_CONTINUOUS;
/*Set to single measurement mode*/
handle.RunMode = RUN_SINGLE_SHOT;
HPS3D_SetRunMode(&handle);
```

②Write the observer callback function, the code is as follows:

```
/*
 * User processing function,Continuous measurement or asynchronous mode
 * in which the observer notifies the callback function
 * */
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("distance average:%d
```

```
                \n",event->MeasureData.full_depth_data->distance_average);
                break;
                case SIMPLE_DEPTH_PACKET:
                    break;
                case NULL_PACKET:
                    break;
                default:
                    printf("system error\n");
                    break;
            }
        }
        return 0;
}
```

③Observer initialization ,code is as follow:

```
/*Observer initialization*/
AsyncIObserver_t My_Observer; /*Define observer */
/*An observer subscribes to an event as a data receive event*/
My_Observer.AsyncEvent = ISubject_Event_DataRecvd;
My_Observer.NotifyEnable = true; /*observer enable*/
/*Set the observer id,which currently supports only a single observer*/
My_Observer.ObserverID = 0;
```

④Add asynchronous observer, code is as follow:

```
/*Adding asynchronous observers,Only valid in asynchronous or continuous
measurement mode*/
HPS3D_AddObserver(&User_Func,&handle,&My_Observer);
```

6. The configuration of steps 1-5 can measure the data normally. The default data is the complete depth map data packet (including depth data). There are four types of returned packets: simple ROI packets (without depth data), complete ROI packets (with depth data), simple depth packets (without depth data), and complete deep packets (with depth data). It can also be converted to point cloud packets (only full ROI packets and full depth packets are available for point cloud format conversion).

Note: Depth data is stored in a one-dimensional array, stored in order, if you need, please traverse.

(1) The default is the complete depth map data packet (including depth data) output. To configure simple packet output, you need to call the function HPS3D_SetPacketType to set it. code show as below:

```
/*set measure packet*/
handle.PacketType = PACKET_SIMPLE;/*set simple packet*/
HPS3D_SetPacketType(&handle);
```

(2) If it is to be configured as a ROI packet (simple and complete package with (1) configuration), just set the ROI area and enable the ROI. If you need to set the threshold to configure itself, the code is as follows:

```
ROIConfTypeDef roi_conf;
/*set ROI config*/
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) To configure the point cloud data output, you need to call the optical parameter enable function HPS3D_SetOpticalEnable and the function HPS3D_SetPointCloudEn to enable point cloud data output before setting the running mode. The data returned under single measurement and continuous measurement is point cloud data. ,code show as below:

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

The point cloud data obtained by this SDK is ordered point cloud data. The point cloud data format uses (x, y, z) space coordinates as point cloud data; provides the structure (in api.h), the code is as follows:

```
/*point cloud data struct */
typedef struct
{
    float32_t x;                /*x,y,z coordinates in space*/
    float32_t y;
    float32_t z;
}PerPointCloudDataTypeDef;
/*Ordered point cloud data*/
typedef struct
{
    PerPointCloudDataTypeDef point_data[MAX_PIX_NUM]; /*point cloud data */
    uint16_t width;                   /*width,the number of points a row */
    uint16_t height;                  /*height,line number */
    uint32_t points;                  /*total points */
}PointCloudDataTypeDef;
```

Note：

(1) The optical enable needs to be turned on before enabling the output point cloud data. Purpose: To obtain the vertical distance.

(2) In the data of the depth map, there are invalid points. Here, the invalid coordinates are also given. The spatial coordinates (x, y, z) are: z = distance[] (the original invalid data value is retained.); x, y is the position of z in the distance (that is, the position of the invalid point)

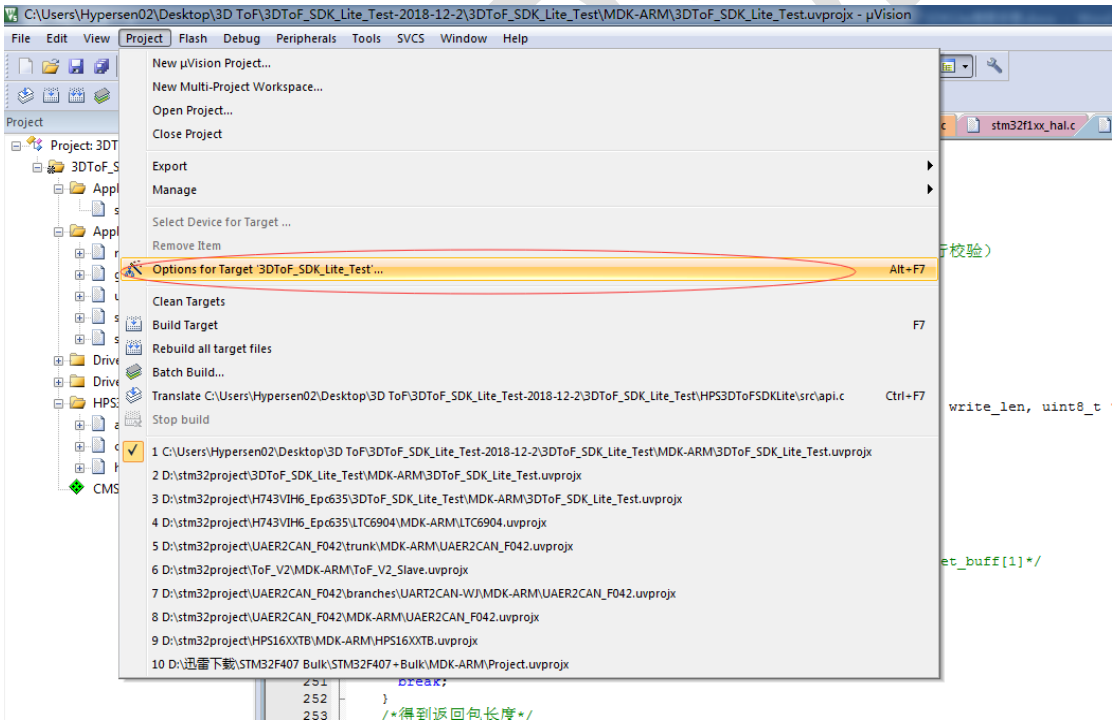
(3) In the return packet structure MeasureDataTypeDef type, the defined point cloud data packet is the structure array PointCloudDataTypeDef, and the point cloud data converted for the depth map data is stored in the array [0], and the point cloud for ROI data conversion. The data is stored in the array in order.

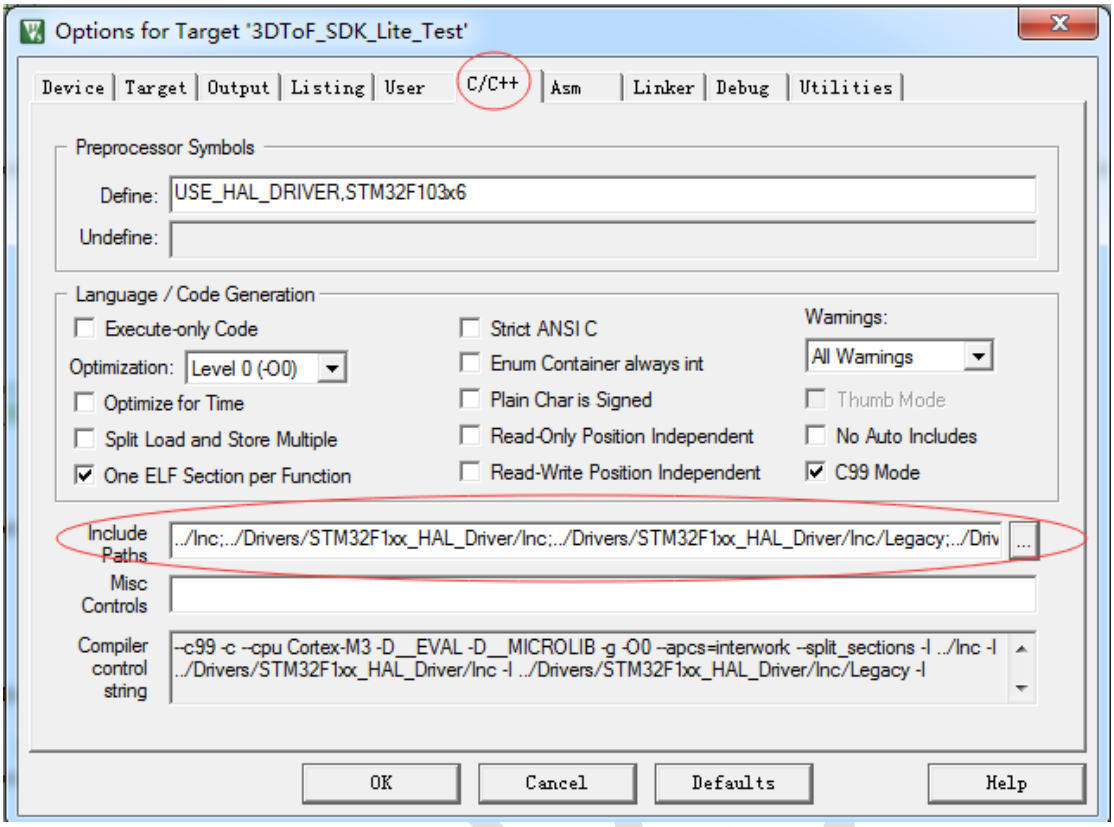
2.2 Integrate SDK Lite into IDE

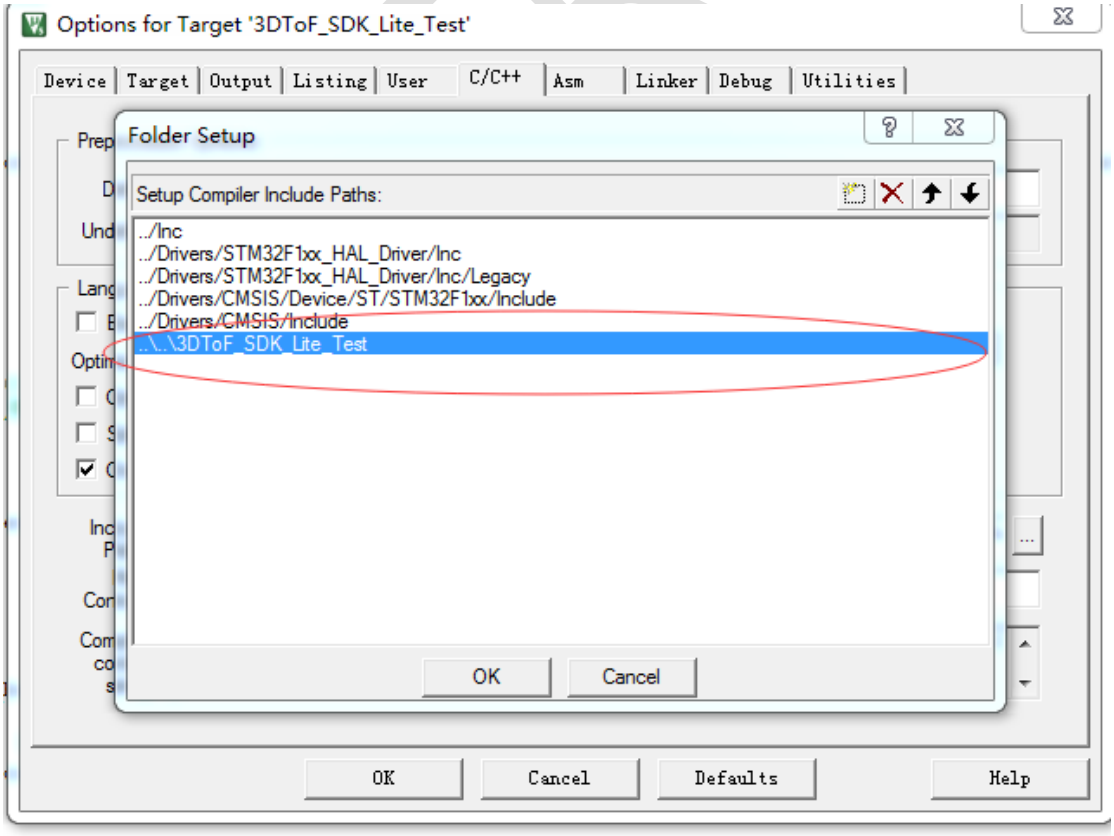
SDK Lite provides a lightweight application interface for the HPS3D160 Depth camera. It is suitable for most microcontroller platforms without running the operating system. In view of the differences in various MCU platforms, a source-level SDK is provided, so it needs to be integrated into the user's project. The source code is compiled. SDK Lite is a lightweight secondary development package that provides only the basic operation interface. Since the depth map and the complete ROI (Region of interest) data occupy a large memory (may require more than 1 Mbyte of RAM), SKD Lite does not currently support depth map data and complete ROI, it only supports the streamlined packet format parsing; this example is based on the SDK with the SDK version number 2018.12.03 V1.0.0.

2.2.1 Add head file search path

The header file in SDKLite contains the relative path relative to the project root directory. To ensure that the SDK can be compiled normally, you need to include the project root directory path in the header search path in the IDE. Take Keil as an example:



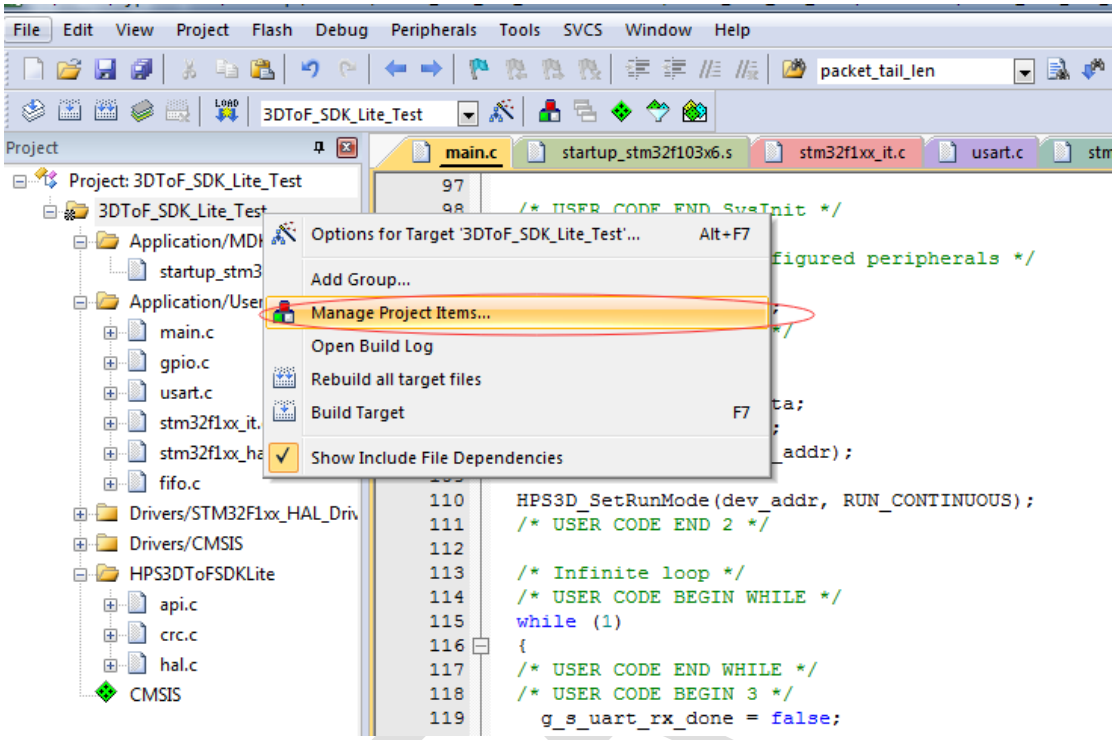


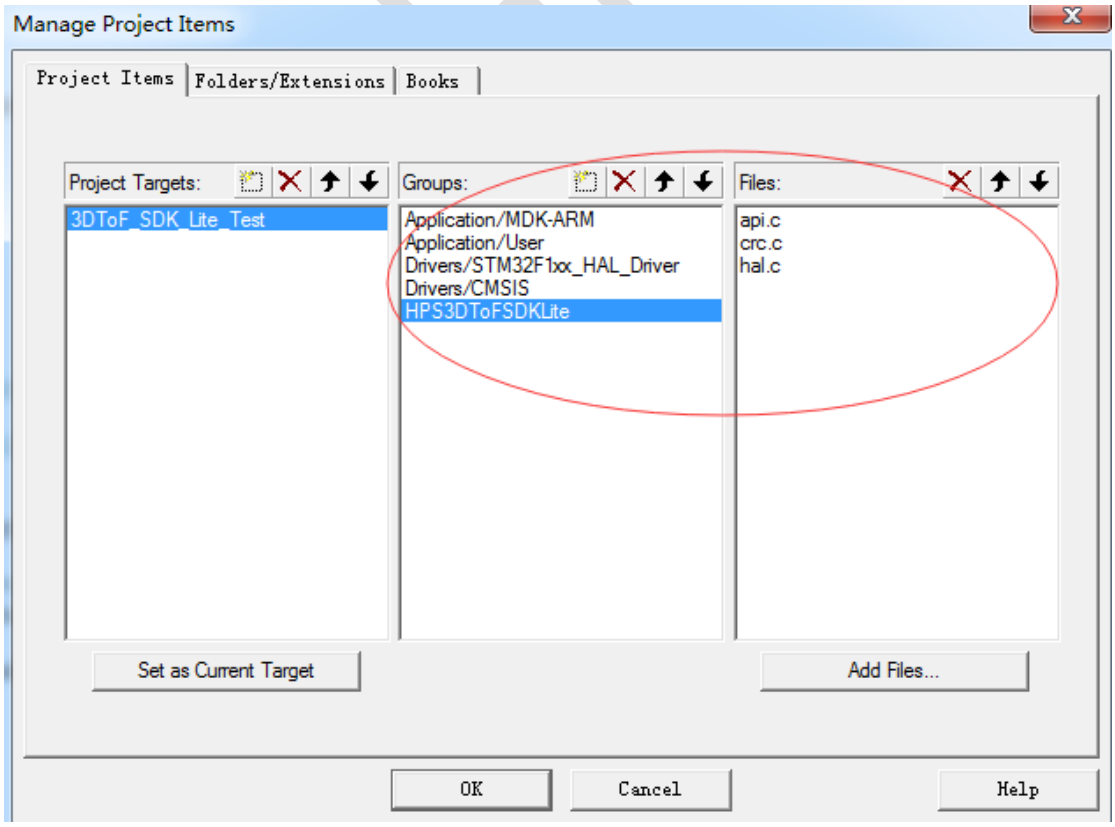


Above pic is 3DToF_SDK_Lite_Test, the project's root directory

2.2.2Add SDK directory source code to project

Take Keil as example, add SDK directory source code to project





2.2.3 Port to the user's platform

SDK Lite only supports HPS3D160 Depth camera with RS232 and RS485, due to the differences between platforms, bottom layer such as communication interfaces need to be ported

1、 Edit HPS3DToFSDKLite/src/hal.c file

2、 Adapt Uart_Read and Uart_Write interfaces is fine， Take an example based on HAL library STM32 platform

```
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 Use SDK in user's project

1. The source file includes head file, sample code：
   #include "HPS3DToFSDKLite/inc/api.h"

2. Call the initialization API to initialize the Depth camera accordingly. This function will set the camera's packet format to a compact format and stop the current continuous measurement, returning the current camera's device address. This address will be used in other APIs. , be sure to check the return value of the initialization function, and determine whether the initialization is successful based on the return value. Most of the SDK's APIs provide operational state return. It is recommended that users check the return value of each API to ensure reliability. Sample code:
HPS3D_Initialize(&dev_addr);

3. After step 2, you can use the API in the SDK to parse the packet and configure the Depth camera parameters.

2.2.5 Analyze measurement packets using the SDK

The SDK provides a data parsing API, but does not implement the receipt of measurement data because it is too relevant to the platform.

1. Set the camera's operating mode, three modes are available:

   1) RUN_IDLE: Idle mode, in which the camera enters standby mode without any measurement.

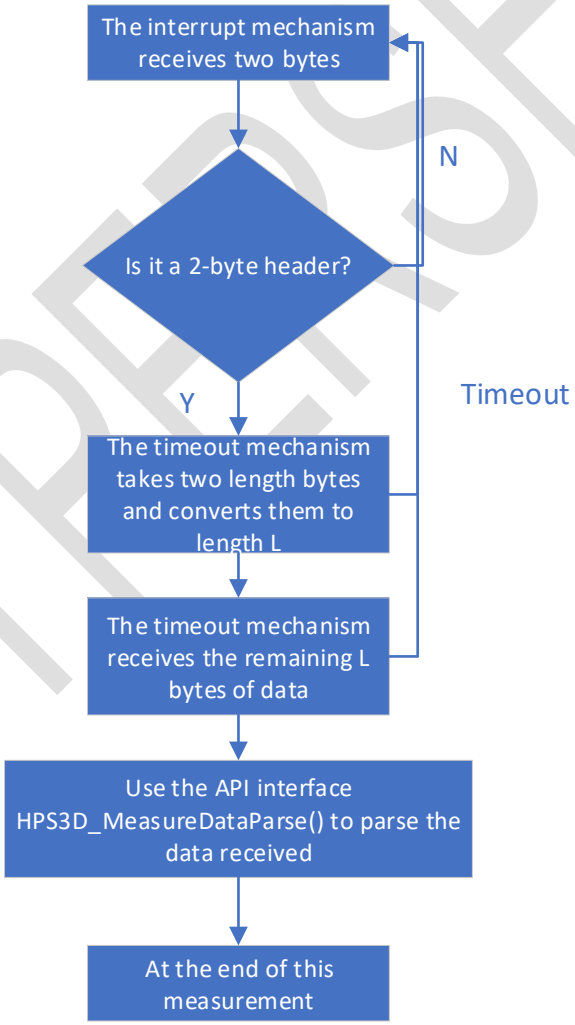
2) RUN_SINGLE_SHOT: Single measurement mode. After setting this mode, the camera takes a measurement and then automatically switches to RUN_IDLE.

3) RUN_CONTINUOUS: Continuous measurement mode. After setting this mode, the camera will automatically make continuous measurement until the user manually sets it to RUN_IDLE or RUN_SINGLE_SHOT mode. In this mode, the data will continuously output measurement data through RS485 or RS232.

2. The serial port receives the measurement data packet. For the format of the data packet, please refer to the HPS-3D160 specification. Please follow the format definition to receive the data packet. Refer to the following data receiving process to improve the system according to its own platform:

1) Interrupt + timeout mechanism to receive measurement data

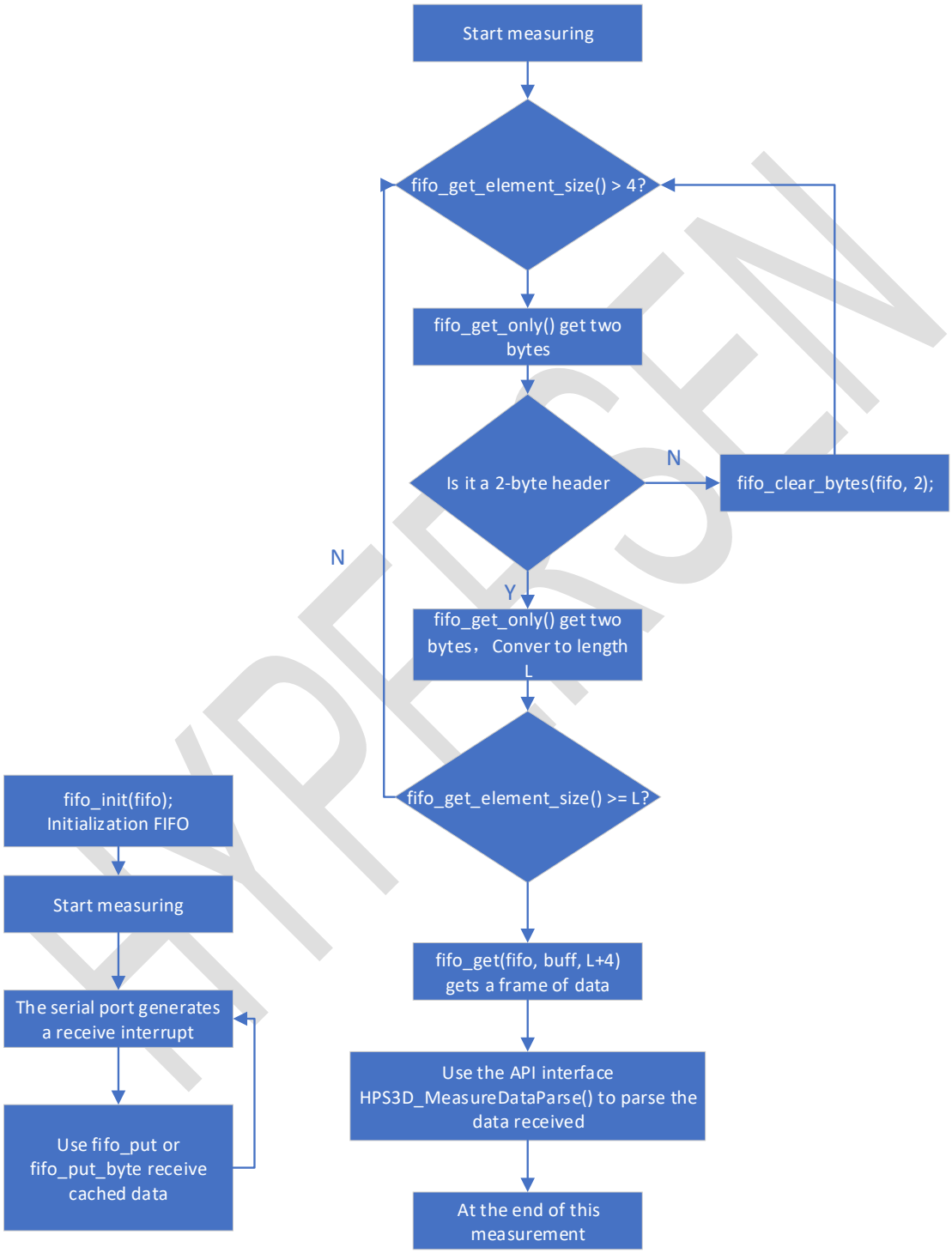
The advantage of this receiving mechanism is that no additional data buffer is needed, but there may be frame dropping in environments with more peripheral interrupts, so this mechanism is more suitable for low-load environments or RAM-tight platforms, with high memory space utilization, but at the expense of time utilization.



2) Data asynchronous caching mechanism

This data receiving mechanism needs to open a data buffer separately for buffering data. The circular FIFO can be used (the fifo module is provided by the SDK test program), which can greatly alleviate the frame loss caused by receiving data for a relatively high load environment. The

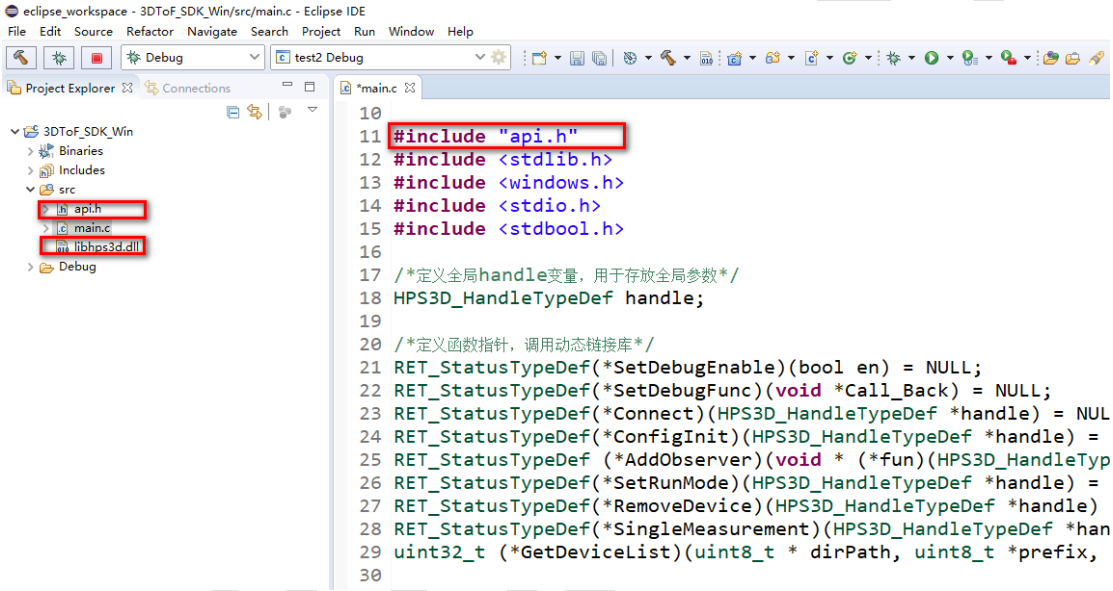
implementation of the mechanism needs to open the serial port interrupt. The serial port buffers the data into the FIFO every time the serial port generates the receiving interrupt. The main program only needs to check and judge the data length of the FIFO, and can fully utilize the waiting time of the mode 1), which has higher Time utilization, but at the expense of space utilization.



2.3 Integrate SDK to IDE under Windows platform

Xxx.dll is suitable for use on the Windows operating system platform. Here is an example of eclipse under Windows. This example is written based on the SDK with API version number 2018.12.08 V1.0.0.

2.3.1 Add xxx.dll dynamic link library and api.h head files to the project



The xxx.dll dynamic link library file can also be placed under other paths, and the absolute path can be filled when called;

2.3.2 SDK Use SDK in user's project

The following example loads the dynamic link library using the LoadLibraryA function interface under Windows:

1. api.h header file is included in the project, and the api.h header file is included in the project directory.

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

2、Define the used function pointer

```
/*Define function pointers to call dynamic link libraries*/
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、Load the dynamic library and get the corresponding function address

```
    /*Load Library*/
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. Call the corresponding interface function for device connection, device initialization configuration, etc. For detailed examples, please refer to the Demo program under Windows platform;

① device connection

```
handle.DeviceName = "\\\\.\\COMxx";     /*port number*/
ret = Connect(&handle);  /*device connect*/
```

```
if (ret != RET_OK)
{
    printf("device connect failed! ret = %d\n", ret);
}
```

② device initialization, the default configuration is standby mode and asynchronous communication mode, mainly to create an asynchronous thread, obtain the device address (the rest of the commands use the device address), allocate memory space, etc.

```
/*Device initialization */
ret = ConfigInit(&handle);
if (RET_OK != ret)
    {
        printf("Initialization failed! error code is: %d\n", ret);
    }
    printf("Initialization succeed! \n");
```

③ Set measure mode: single measure or continuously measure

```
/*Set to continuously measurement mode */
handle.RunMode = RUN_CONTINUOUS;
HPS3D_SetRunMode(&handle);

/*Set to single measurement mode */
Handle.RunMode = RUN_SINGLE_SHOT;
HPS3D_SetRunMode(&handle);
```

After the configuration is completed according to the above steps, you can get the complete deep data packet of the measurement output under the default configuration.

Among them, the continuous mode requires an asynchronous thread to continuously output. In this case, the observer mode can be used to monitor the measured return data. The sample code is as follows:

```
/*
 * User processing function,Continuous measurement or asynchronous mode
 * in which the observer notifies the callback function
 * */
void* User_Fun(HPS3D_HandleTypeDef *handle,AsyncIObserver_t *event)
{
    if(event->AsyncEvent == ISubject_Event_DataRecvd)
    {
        switch(event->RetPacketType)
        {
            case SIMPLE_ROI_PACKET:
                printf("distance
average:%d\n",event->MeasureData.simple_roi_data[0].distance_average);
                break;
            case FULL_ROI_PACKET:
                printf("distance
average: %d\n",event->MeasureData.full_roi_data[0].distance_average);
```

```
                break;
            case FULL_DEPTH_PACKET:
                printf("distance
average: %d\n",event->MeasureData.full_depth_data->distance_average);
                break;
            case SIMPLE_DEPTH_PACKET:
                printf("distance
average: %d\n",event->MeasureData.simple_depth_data->distance_average);
                break;
            case NULL_PACKET:
                break;
            default:
                printf("system error\n");
                break;
        }
    }
}

/*Observer initialization*/
AsyncIObserver_t My_Observer;
/*An observer subscribes to an event as a data receive event*/
My_Observer.AsyncEvent = ISubject_Event_DataRecvd;
My_Observer.NotifyEnable = true; /*enable observer*/
My_Observer.ObserverID = 0;  /*observer id*/

/*Adding asynchronous observers */
ret = HPS3D_AddObserver(&User_Fun,&handle,&My_Observer);
if(RET_OK != ret)
{
    printf("observer add failed, error code:%d\n", ret);
}
```

Single measurement mode can support synchronization or asynchronous mode. Asynchronous mode is the same as above. It is recommended to use synchronous mode. The synchronization mode is as follows: Note: Synchronous single measurement function will set handle.SyncMode to SYNC synchronization mode, if you want to switch to asynchronous The mode needs to set this parameter to ASYNC;

```
ret = HPS3D_SingleMeasurement(&handle);
if(ret == RET_OK)
{
    switch(handle.RetPacketType)
    {
        case SIMPLE_ROI_PACKET:
            printf("Simple Roi measure distance average:%d
\n",handle.MeasureData.simple_roi_data[0].distance_average);
```

```
            break;
        case FULL_ROI_PACKET:
            printf("Full Roi measure distance average:%d
\n",handle.MeasureData.full_roi_data[0].distance_average);
            break;
        case FULL_DEPTH_PACKET:
            printf("Full depth measure distance average:%d
\n",handle.MeasureData.full_depth_data->distance_average);
            break;
        case SIMPLE_DEPTH_PACKET:
            printf("simple depth measure distance average:%d
\n",handle.MeasureData.simple_depth_data->distance_average);
            break;
        case NULL_PACKET:
            printf("return packet is null\n");
            break;
        default:
            printf("system error\n");
            break;
    }
}
```

2.4 Environment configuration under ROS platform and integration of SDK into IDE

SDK ROS provides the application interface of HPS3D160 Depth camera. The generated 32-bit/64-bit .so dynamic link library is suitable for ROS platform on Linux operating system. The .so and api.h interfaces can be integrated into the user's project source code. Compile. The SDK is a secondary development package that provides only the basic operational interface. You can get the depth map and the complete ROI (sensitive area) data. If you need to convert to point cloud data, you can enable the interface to convert the point cloud data before acquiring the data. This document is based on the SDK with API version number 2018.12.10 V1.0.0.

SDK ROS is suitable for use on the Linux operating system ROS platform. Here Ubuntu 14.04 is taken as an example. Because the Linux operating system of Ubuntu 14.04 is installed, the corresponding ROS version is installed as the distribution version indigo, and the 1.11.21 version is installed here. Enter rosrun at the terminal and run the ROS bus to view the version of ROS running, as shown below:

```
roscore http://ubuntu:11311/
dote@ubuntu:~$ roscore
... logging to /home/dote/.ros/log/e49dea3e-faa1-11e8-8239-000c299617c1/roslaunc
h-ubuntu-3017.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:39110/
ros_comm version 1.11.21


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [3039]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to e49dea3e-faa1-11e8-8239-000c299617c1
process[rosout-1]: started with pid [3052]
started core service [/rosout]
```

For device connection, please refer to "2.1.1 HPS3D160 Device Connection" in this document.

2.4.1 Create a work space

Before creating a workspace, first look at the environment variables, enter echo $ROS_PACKAGE_PATH in the terminal, and view the environment variables on Linux, as shown below:

```
dote@ubuntu:~$ echo $ROS_PACKAGE_PATH
/opt/ros/indigo/share:/opt/ros/indigo/stacks
dote@ubuntu:~$
```

Then check if the catkin tool is installed. If it is not installed, please install the catkin tool first. By default, there is a catkin tools, catkin is an official compilation of ROS build system, it is the successor to the original compilation of the ROS build system.

(1) Enter the environment variable of the ROS system and enter source /opt/ros/indigo/setup.bash at the terminal.

(2) Create a workspace under the home directory and enter mkdir -p ~/HPS3D_SDK_ROS_Demo/src.

(3) In the src directory, type cd ~/HPS3D_SDK_ROS_Demo/src, execute the initialization space, and enter catkin_init_workspace, as shown below:

```
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) Through the above configuration, you can create a workspace, and go to the created workspace, enter cd ~ / HPS3D_SDK_ROS_Demo /; although this space is empty, we can still build it, input catkin_make, as shown below:

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

(5)Enter ls to view the current working directory, you will find two extra folders "build" and "devel". In the devel folder, you can see a lot of setup.*sh files. Enter source devel/setup.bash to configure your workspace as shown below:

```
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$
```

Note: Any source files, python libraries, scripts, and other static files will be left in the source space src. However, all generated files, such as library files, executable files, and generated code, are placed in devel.

2.4.2 Create a ROS packet (Catkin packet)

(1) Create a package named hps_camera, which directly depends on the following three packages: std_msgs, rospy and roscpp. Enter cd ~/HPS3D_SDK_ROS_Demo/src/ in the terminal and enter the src directory, then enter, catkin_create_pkg hps_camera std_msgs rospy roscpp, as follows The figure shows:

```
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) Enter rospack depends hps_camera in the terminal, you can see that the package can have many dependencies, you can see the three dependencies added by itself: std_msgs, rospy and roscpp, as shown below:

```
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$
```

Note: If the rospack depends hps_camera is executed, the following error occurs (when the ros* command is used, an error occurs, it may be that the workspace has failed, it can be re-validated, or it can be written into the ros environment variable) Go back to the workspace directory and execute source devel/setup.bash again to make the workspace work, as shown in the following figure:

```
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
```

2.4.3 Create ROS message msg and service srv

The msg file is a simple text file that describes the fields of messages in ROS and is used to generate source code for different languages for messages.

The srv file describes a service consisting of two parts, a request and a service.

Tip: A msg file or a srv file is equivalent to a structure, so you can compare the provided api.h. There are five data types in the depth camera data return package. Each data type is a structure, that is, When writing msg files, you can nest them.

1、Create ROS message msg

(1) In the created package, create a message msg directory to store the msg file. Enter cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/ in the terminal, go to the package directory; enter mkdir msg, create the msg directory; type echo "uint16 distance_average" > msg/distance.msg, create the distance.msg file, and write the average distance variable. As shown below:

```
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）Enter sudo gedit package.xml in the terminal, configure package.xml, add the following two lines of code to the file, as shown below:

```
<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>
```

Then, package.xml configuration is completed, save and exit.

Enter sudo gedit CMakeLists.txt in the terminal, configure CMakeLists.txt, and find the corresponding location to modify.

1 Add message_generation to the following code slice, the result is as shown below:

```
## 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 Add CATKIN_DEPENDS message_runtime to following code part，the result is shown as below after adding.

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

③Find the code part as below:

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

Add it to the new created distance.msg message file, as below:

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

④Find the code part as below：

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

Modify it as below:

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

Then CMakeLists.txt configuration is completed, save and quit。

（3）Check new created msg message，enter rosmsg show distance in terminal，as below：

```
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、 Create ROS service srv

(1) In the created package, create a service srv directory to store the srv file. In the terminal, type cd ~/HPS3D_SDK_ROS_Demo/src/hps_camera/, go to the package directory; enter mkdir srv, create the srv directory; type sudo gedit srv/camera.srv, create the camera.srv file, and enter the following code, as shown below:

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

Note： ”string client_node_name” is request，The name of the storage client node is sent to the server, "---" is to separate the request and response, "string control_cmd" is the response, the control command sent by the storage server to the client.

```
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）Enter sudo gedit CMakeLists.txt in terminal，configure CMakeLists.txt，find corresponding

place and modify. Find code part as below：

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

Add it to the new created camera.srv service file， shown as below：

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

Now CMakeLists.txt configuration is completed， save and quit。

（3）Check new created srv service, enter rossrv show camera in terminal， shown as below：

```
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、Re-build ROS packet

In work directory, enter catkin_make， and build， then it is shown as below：

```
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$
```

2.4.4 Create ROS Depth camera client node and server

Routines are provided in this section. Users can modify or write their own programs as needed. The procedures for the Depth camera client node and server routines given in this document are:

(1) The client node is configured with the data that the user needs to collect (or configured by the command sent by the server);

(2) After the client logs in, connect the optional device file. After the connection is successful, send the client name (custom name) to the server;

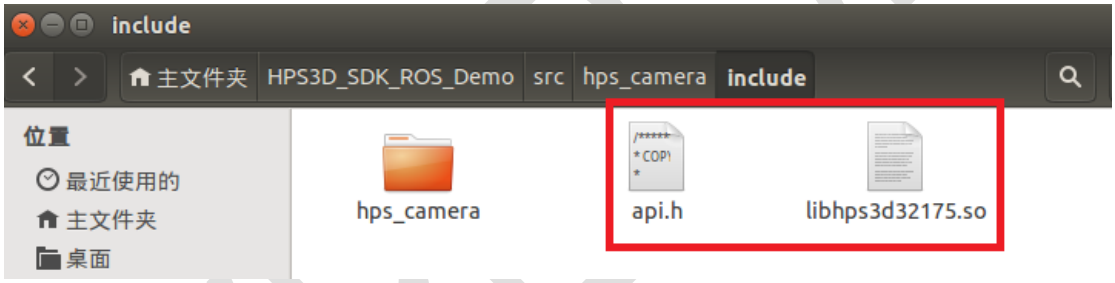
(3) After the server receives the message (client name) sent by the Depth camera client, it performs name judgment, is it a depth camera client, and if so, sends a start command (custom command) to the client, otherwise it continues to wait for the client connection, sending a message;

(4) When the client receives the command sent by the server, it determines what command is, and if it is the start command, it starts to collect data and issues a message to the server;

(5) The server receives the message sent by the client node, and can perform further matching, setting, and etc.

1、Integrate api.h and .so file to engineering

(1) Copy api.h and lib*.so into the include directory of the package in the workspace, as shown below:



(2) Copy lib*.so to /usr/local/lib and enter sudo mv include/libhps3d32175.so /usr/local/lib/ in the terminal of the package directory. After copying, enter sudo ldconfig to load Configuration, as shown below:

```
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.so /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）Enter sudo gedit CMakeLists.txt in the terminal of the package directory, configure the CMakeLists.txt file, and find the following code piece:

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

Modify it as shown below:

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

Now CMakeLists.txt configuration is completed，save and quit。

2、Create ROS Depth camera client node and server

（1）In the src directory of the package directory, create ros_camera_client.cpp and ros_camera_server.cpp, and enter sudo touch src/ros_camera_client.cpp src/ros_camera_server.cpp in the terminal of the package directory, as shown below:

```
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$
```

(2) Enter sudo gedit CMakeLists.txt in the terminal of the package directory, configure the CMakeLists.txt file, and add the following code to the CMakeLists.txt file, as shown below:

```
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 Write ROS Depth camera client node and server

（1）Write server program

①Add head file, code as below:

```
#include "ros/ros.h"//ros
```

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

②ROS Initialize in the main function, create node, create topic and etc., code as below:

```
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();
```

③Write service function, code as below:

```
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 Subscribe to the topic's callback function, the code is as follows:

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

（2） Write Depth camera client node program

In the client node, the use of the depth camera api interface is the same as the "2.1.3 Using the SDK in User Projects" configuration of this document. For details, please refer to "2.1.3 Using the SDK in User Projects" in this document or give the sample code.

1 Add a header file, the code is as follows:

```
#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 In the main function, ros initialization, node creation, topic creation and depth camera api interface configuration, the code is as follows:

```
ros::Publisher camera_pub;
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 In "2.1.3 Using the SDK in User Projects" in this document, point 5 (2) Asynchronous Measurement 2 Write the observer callback function, and assign the value to the msg message structure when the value is measured. And released. code show as below:

```
/*observer callback function*/
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:
                printf("return packet is null!\n");
                break;
            default:
                printf("system error!\n");
                break;
        }
        event->RetPacketType  = NULL_PACKET;
    }
    return 0;
}
```

4、Test ROS Depth camera client node and server

（1）Enter catkin_make in the terminal of the workspace directory, execute the compile link, and you can see the executable file in the /devel/lib/hps_camera folder, as shown below:

```
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）Enter cd /dev in terminal，check device file, enter ll ttyACM* check device file detailed information, find ttyACM* device， enter sudo chmod 777 ttyACM0， Modify the device file permissions as shown below:：

```
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）New open a terminal input roscore，run ros bus，shown as below：

```
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） New open two terminals, all enter cd ~/HPS3D_SDK_ROS_Demo/ into workspace, all enter source devel/setup.bash, effective workspace

①Run the server on one of the terminals，enter hps_camera ros_camera_server, start the server, as shown below:

```
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
```

②On another terminal running client node, enter hps_camera ros_camera_client，start client，shown as below：

```
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
Current connectable device (please select) :
0: /dev/ttyACM0
Please enter the corresponding serial number :
```

③ Choose connectable device on client terminal, enter 0, then get the following results:

Client terminal：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_client
send name = camera_client
Current connectable device (please select) :
0: /dev/ttyACM0
Please enter the corresponding serial number :
0
Initialization succeed
rev cmd = start
login succeed!
distance = 1303
distance = 1310
distance = 1304
distance = 1304
distance = 1307
distance = 1305
distance = 1301
```

Server terminal：

```
dote@ubuntu:~/HPS3D_SDK_ROS_Demo$ rosrun hps_camera ros_camera_server
waiting client login
client_name: camera_client
send_cmd: start
distance_average = 1303
distance_average = 1310
distance_average = 1304
distance_average = 1304
distance_average = 1307
distance_average = 1305
distance_average = 1301
```

3、Command function interface

The command function interface is the basic command in the HPS3D depth camera; each command function interface contains command packet packing and return packet parsing for the command. If the function interface integrated in api.h is not enough for the user to use, the user can configure the new interface according to the command function interface. (Note: After sending the command, you need to reset the running mode!!!)

3.1 Set running mode

3.1.1 Sample code

```
HPS3D_HandleTypeDef handle;
int main(int argc, char *argv[])
{
    ......
    handle.DeviceName = "/dev/ttyACM0"; // handle.DeviceName = "\\\\.\\COM13"
    /*HPS3D connect*/
    ret = HPS3D_Connect(&handle);
    /*Device initialization, get handle.DeviceAddr*/
    ret = HPS3D_ConfigInit(&handle);
    /*Set to continuous measurement mode*/
    handle.RunMode = RUN_CONTINUOUS;
    ret = HPS3D_SetRunMode(&handle);
    ......
}
```

3.2 Get/set device address

3.2.1 Sample code

```
/*get device address*/
ret = HPS3D_GetDevAddr(&handle);
printf("1handle.DeviceAddr = %#x\n",handle.DeviceAddr);

/*set device address*/
ret = HPS3D_SetDevAddr(&handle, 0x01);
ret = HPS3D_GetDevAddr(&handle);
printf("2handle.DeviceAddr = %#x\n",handle.DeviceAddr);
```

3.2.2 Running result

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

3.3 Get device version information

3.3.1 Sample code

```
/*get device version*/
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 Running result

```
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 Get/set data packet type

3.4.1 Sample code

```
/*get packet type*/
HPS3D_GetPacketType(&handle);
printf("1handle.PacketType = %d\n", handle.PacketType);//PACKET_FULL = 0

/*set packet type*/
handle.PacketType = PACKET_SIMPLE;/*simple packet*/
HPS3D_SetPacketType(&handle);
HPS3D_GetPacketType(&handle);
printf("2handle.PacketType = %d\n", handle.PacketType);
```

3.4.2 Running result

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

3.5 Save/Clear/Reset factory setting

3.5.1 Sample code

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

3.6 Get the transfer type

3.6.1 Sample code

```
/*get transport type*/
TransportTypeDef transport_type;
HPS3D_GetTransportType(&handle, &transport_type);
printf("transport_type = %d\n", transport_type);//TRANSPORT_USB = 0
```

3.6.2 Running result

```
transport_type = 0
```

3.7 Get ROI group/get current ROI group ID

3.7.1 Sample code

```
uint8_t group_id = 0;
/*select group ID*/
HPS3D_SelectROIGroup(&handle, 3);

/*get group ID*/
HPS3D_GetROIGroupID(&handle, &group_id);
printf("group_id = %d\n",group_id);
```

3.7.2 Running result

```
group_id = 3
```

3.8 ROI relative settings

3.8.1 Sample code

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

/*group id = 0, roi_id = 0, threshold_id = 2, GPIO alarm enable = true, ROI
reference is distance min.*/
/*group id = 0, roi_id = 2, threshold_id = 1, GPIO alarm enable = false, ROI
reference is vaild amplitude */
/*set ROI alarm type*/
HPS3D_SetROIAlarmType(&handle, 0, 2, ROI_ALARM_GPIO);
HPS3D_SetROIAlarmType(&handle, 2, 1, ROI_ALARM_DISABLE);

/*set ROI reference type*/
HPS3D_SetROIReferenceType(&handle, 0, 2, ROI_REF_DIST_MIN);
HPS3D_SetROIReferenceType(&handle, 2, 1, ROI_REF_VAILD_AMPLITUDE);

/*set ROI config*/
```

```
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);

/*set ROI enable*/
HPS3D_SetROIEnable(&handle, 0, true);
HPS3D_SetROIEnable(&handle, 2, true);

/*set ROI threshold enable*/
HPS3D_SetROIThresholdEnable(&handle, 0, 2, true);
HPS3D_SetROIThresholdEnable(&handle, 2, 1, true);

/*set ROI threshold config*/
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);

/*get ROI config param*/
HPS3D_GetROIConfById(&handle, 0, &roi_conf1);
HPS3D_GetROIConfById(&handle, 2, &roi_conf2);

/*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");

/*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 Running result

```
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 Get current device support ROI number and threshold number

3.9.1 Sample code

```
/*Get the number of ROI and thresholds currently support this device*/
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 Running result

```
roi_number = 20
threshold_number = 3
```

3.10 Set/ Get output/ input settings

3.10.1 Sample code

```
/*get GPOUT parameter */
GPIOOutConfTypeDef gpio_out_conf;
gpio_out_conf.gpio = GPOUT_1;
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);

/*get GPIN parameter */
GPIOInConfTypeDef gpio_in_conf;
gpio_in_conf.gpio = GPIN_1;
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);

/* set GPOUT parameter */
gpio_out_conf.gpio = GPOUT_1;
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);

/* set GPIN parameter */
gpio_in_conf.gpio = GPIN_1;
gpio_in_conf.function = 0;
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 Running result

```
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 Set HDR mode

3.11.1 Sample code

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

/*set AUTO_HDR mode*/
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 Running result

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

3.12 Set/get HDR configuration

3.12.1 Sample code

```
/*get distance filter parameter*/
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、mode  = 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、mode  = AUTO-HDR*/
set_conf.hdr_mode = AUTO_HDR;
set_conf.qualtity_overexposed = 600;
set_conf.qualtity_overexposed_serious = 900;
set_conf.qualtity_weak = 80;
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、mode = SIMPLE-HDR*/
set_conf.hdr_mode = SIMPLE_HDR;
set_conf.simple_hdr_max_integration = 500;
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、mode = 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 Running result

```
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 Set/get distance filter configuration

3.13.1 Sample code

```
/*get distance filter parameter*/
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

/*set distance filter parameter*/
HPS3D_SetDistanceFilterType(&handle, DISTANCE_FILTER_SIMPLE_KALMAN);
HPS3D_GetDistanceFilterConf(&handle, &distance_filter_conf);
printf("2distance_filter_conf.filter_type
= %d\n",distance_filter_conf.filter_type);

/*set distance filter parameter*/
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 Running result

```
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 Set/get smoothing filter configuration

3.14.1 Sample code

```
/*get mooth filter parameter*/
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*/

/*set smooth filter parameter*/
set_conf.type = SMOOTH_FILTER_AVERAGE;
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 Running result

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

3.15 Set optical parameter enable/get optical parameter

3.15.1 Sample code

```
/*get optical parameter*/
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);

/*set optical enable = disable */
HPS3D_SetOpticalEnable(&handle, false);
HPS3D_GetOpticalParamConf(&handle, &optical_param_conf);
printf("2optical_param_conf.enable = %d\n",optical_param_conf.enable);
```

3.15.2 Running result

```
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 Set/get distance compensation

3.16.1 Sample code

```
/*get distance offset*/
int16_t offset;
HPS3D_GetDistanceOffset(&handle, &offset);
printf("1offset = %d\n",offset);

/*set distance offset*/
HPS3D_SetDistanceOffset(&handle, 20);

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

3.16.2 Running result

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

3.17 Set/get multi-machine interference parameters

3.17.1 Sample code

```
/*get interference detect parameter */
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

/* set interference detect enable */
HPS3D_SetInterferenceDetectEn(&handle, true);
```

```
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.enable = %d\n",interference_detect_conf.enable);

/* get interference detect integ. time*/
HPS3D_SetInterferenceDetectIntegTime(&handle, 200);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.integ_time
= %d\n",interference_detect_conf.integ_time);

/* get interference detect amplitude threshold*/
HPS3D_SetInterferenceDetectAmplitudeThreshold(&handle, 5);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.amplitude_threshold
= %d\n",interference_detect_conf.amplitude_threshold);

/* get interference detect capture number*/
HPS3D_SetInterferenceDetectCaptureNumber(&handle, 6);
HPS3D_GetInterferenceDetectConf(&handle, &interference_detect_conf);
printf("2interference_detect_conf.capture_num
= %d\n",interference_detect_conf.capture_num);

/* get interference detect check number*/
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 Running result

```
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 Set/get assemble angle parameter

3.18.1 Sample code

```
/*get assemble angle parameter*/
```

```
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);

/*set assemble angle enable*/
HPS3D_SetMountingAngleEnable(&handle, true);

/*set assemble angle parameter*/
set_conf.angle_vertical = 50;/*50 ° */
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 Operation result

```
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
```

4. Q&A

4.1 The encoding format does not match?

We provide SDK encoding format is UTF-9, if the encoding format error occurs when using the API provided by the SDK, please create a new api.h file in the project, and copy the api.h content provided by the SDK to the new api.h file to solve the compilation error.

5. Revision

Date	Revision	Description
2018/12/11	1.0.0	Initial version

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.