Xvisio_SDK_Guide

v1.2/ 2022.03

Xvisio Ltd.

Room 408, building 1, No.288, Tong Xie Road, Changning District, Shanghai

http://www.Xvisiotech.com

Revision History

Version	Description	Author
1.0	First version	Xvisio
1.1	XVSDK 3.0.2	Xvisio
1.2	XVSDK 3.1.0	Xvisio

Contents

1. Overview 5

2. Environment Construction for Xvisio SDK 5

2.1 Android SDK Environment Construction 5

2.1.1 Android SDK Structure 5

2.1.2 Android Box Requirements 6

2.1.3 Android Environment Test Verify 6

2.2 Windows SDK Environment Construction 7

2.3 Ubuntu SDK 13

3. Xvisio SDK Development 16

3.1 Xvisio Slam 16

3.1.1. Slam Mode Settings 16

3.1.2. Slam Center Point 16

3.1.3. 6dof Structure 17

3.1.4. Get 6dof 18

3.2 VIO Mode 20

3.3 CSLAM Mode 20

3.4 3DOF（Rotation） 25

3.5 Plane Detection (TOF/stereo) 25

3.6 Map Sharing 27

3.7 HID interfaces 28

3.8 Calibration Parameter API 28

3.9 HOT Plug 29

3.10 Get Fisheye Data 29

3.11 Get RGB Data 30

3.12 Get TOF Data 30

3.13 Get RGBD Data 31

3.14 CNN 31

3.15 Gesture Function 33

3.16 Get SGBM Data 35

3.17 Python-wrapper 36

1. Overview

This document describes as a user guide of Xvisio SDK which mainly used to help developers to know Xvisio SDK well and also can develop related applications (slam, camera, audio...) quickly. This document is mainly divided into two parts: SDK environment construction and sample code introduction.

Currently Xvisio SDK supports 3 platforms: Android, Ubuntu and Windows. The mainly difference between these 3 platforms is that ***.so*** is compiled separately, but API is the same. Section 2 introduces how to install/use Xvisio SDK with these 3 platforms. Section 3 describes how to call the SDK API to achieve different functions for all platforms.

2. Environment Construction for Xvisio SDK

This chapter sequentially introduces how to install and use Xvisio SDK on Android, Ubuntu and Windows platforms. And also related precautions are included.

2.1 Android SDK Environment Construction

2.1.1 Android SDK Structure

Android SDK structure is shown as below:

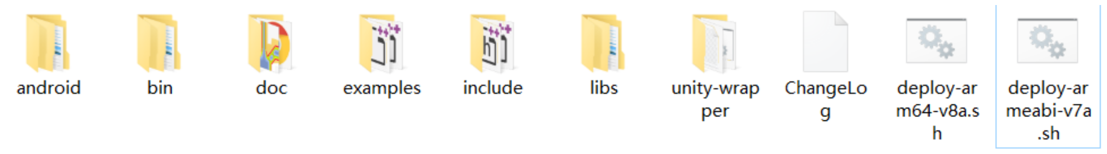


Figure2-1 SDK Structure

The folder **"Android"** contains the makefile compiled by NDK. If user modify the demo code in the samples folder, user can use "ndk-build -j 5" to compile and generate the corresponding executable file. The compiled file path is in */Android/libs/ arm64-v8a (armeabi-v7a)*.

File **“bin”** is the tool of 64bit and 32bit.

File **“doc”** contains definition file of class and interfaces.

Folder **“examples”** is our demo code, mainly describes how to use our SDK API.

The **"include"** folder is the header file of the SDK API.

The folder **“libs”** contains "*SDK so*" files. "*arm64-v8a*" is a 64bit library, and "*armeabi-v7a*" is a 32bit library.
"deploy.sh" is a push script file. Executing "*./deploy-arm64-v8a.sh*" will push the 64bit library to the Android platform; executing "./deploy-armeabi-v7a.sh" will push the 32bit library to the Android platform, Modify the **"libdir"** in *deploy.sh* to modify the push lib path.

2.1.2 Android Box Requirements

- The Android OS version should be above (include) 7;
- Developers need to confirm that the Android kernel supports HID and UVC modules first. If not, kernel needs to be recompiled and also add HID&UVC modules should be added;
- Android OS is userdebug version;
- If only slam function is needed, usb2.0 can satisfy the demand. For complete functions (slam, RGB, TOF, audio), usb3.0 is needed.

2.1.3 Android Environment Test Verify

We took Android native system box as an example to introduce how to test and verify Xvisio Android SDK.
Connect the box with PC after powered on. Execute the following commands to push Xvisio SDK to box:

- *adb root*
- *adb remount*
- *cd android_xxx_sdk*
- *adb push libs /data/test/*
- *adb push bin /data/test/*

Execute the following commands after connecting:

lsusb (make sure whether the device can be found. if the terminal shows f408 which means the device has been connected with box successfully.)

adb shell

cd /data/test/bin/arm64-v8a/

LD_LIBRARY_PATH=../../libs/ arm64-v8a/   ./demo-api

Open a new terminal: *adb shell*， *cd /data/test/bin/arm64-v8a/*，

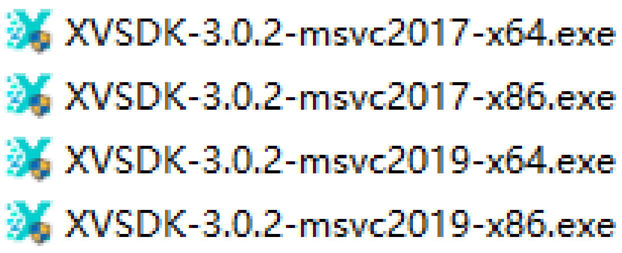
LD_LIBRARY_PATH=../../libs/ arm64-v8a/   ./pipe_srv

The test verification steps of Android SDK is to refer to the test command and input the command successively (input 1 to obtain 3DOF data, input 2 to obtain 6DOF data...)

If the 3dof, 6dof and other sensor data can be obtained normally in the end which means there is no problem with the Android environment, and subsequent developers can integrate the SDK into the Android system.

2.2 Windows SDK Environment Construction

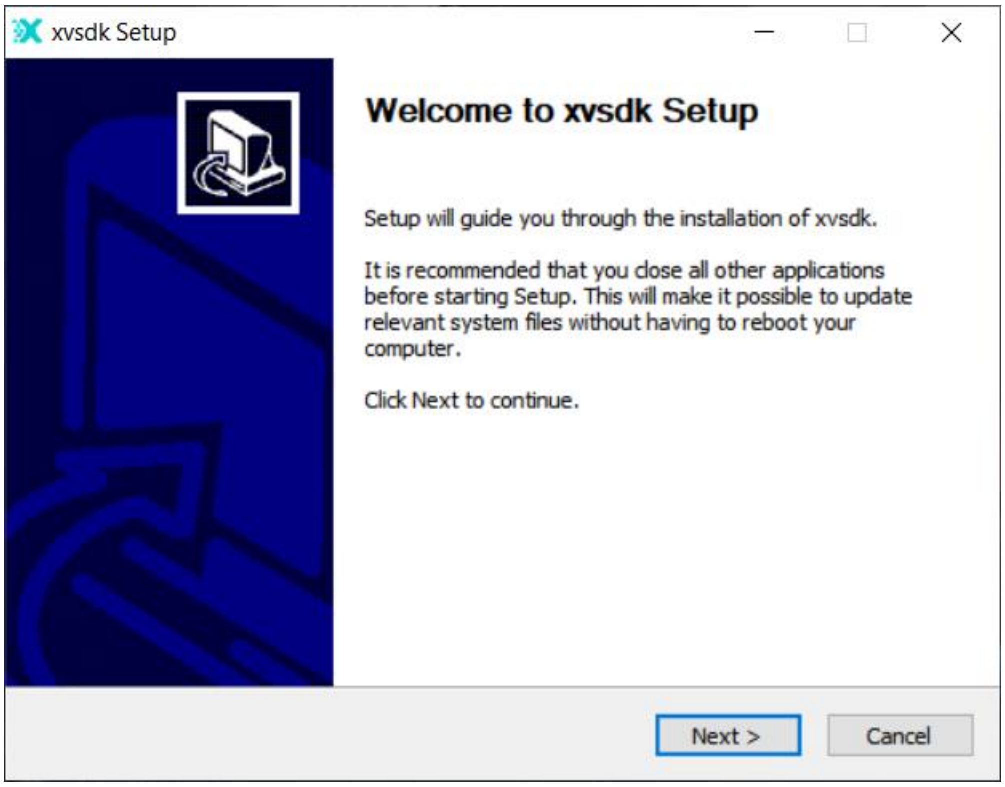
2.2.1 XVSDK Installation



XVSDK-3.0.2-msvc2017-x64.exe
XVSDK-3.0.2-msvc2017-x86.exe
XVSDK-3.0.2-msvc2019-x64.exe
XVSDK-3.0.2-msvc2019-x86.exe

Figure 2-2 XVSDK Installation

Select the installation file according to the configuration of PC. Double click the .exe file and click next step to complete the installation.



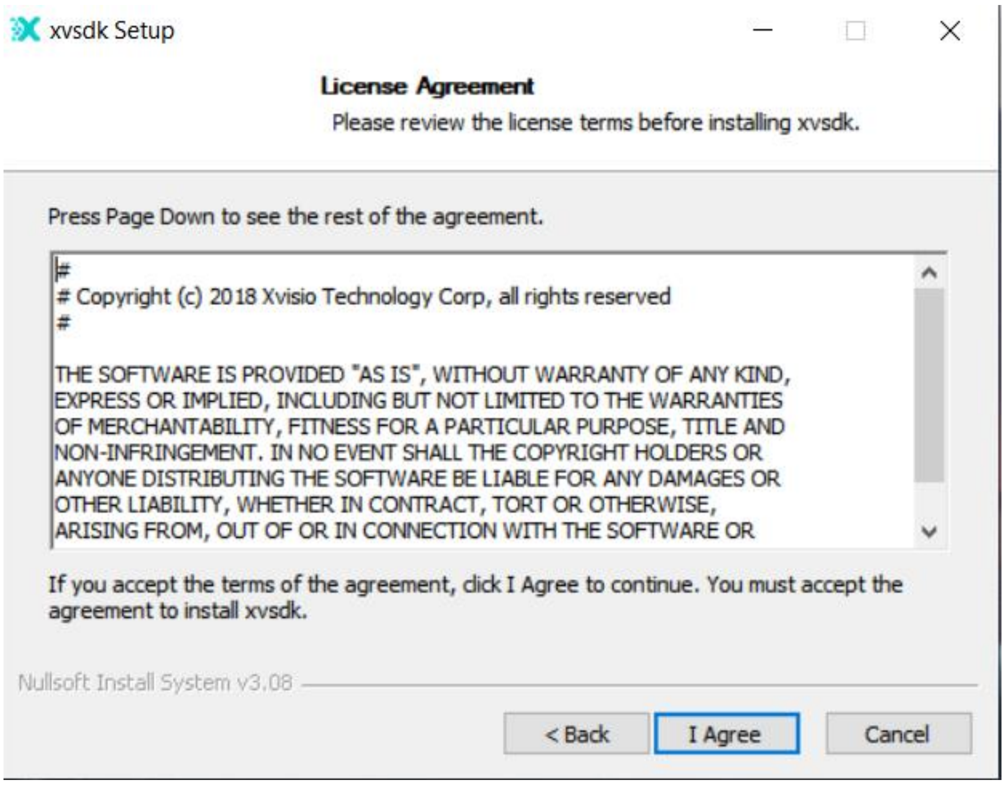
Welcome to xvsdk Setup

Setup will guide you through the installation of xvsdk.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 2-3 Installation1



License Agreement

Please review the license terms before installing xvsdk.

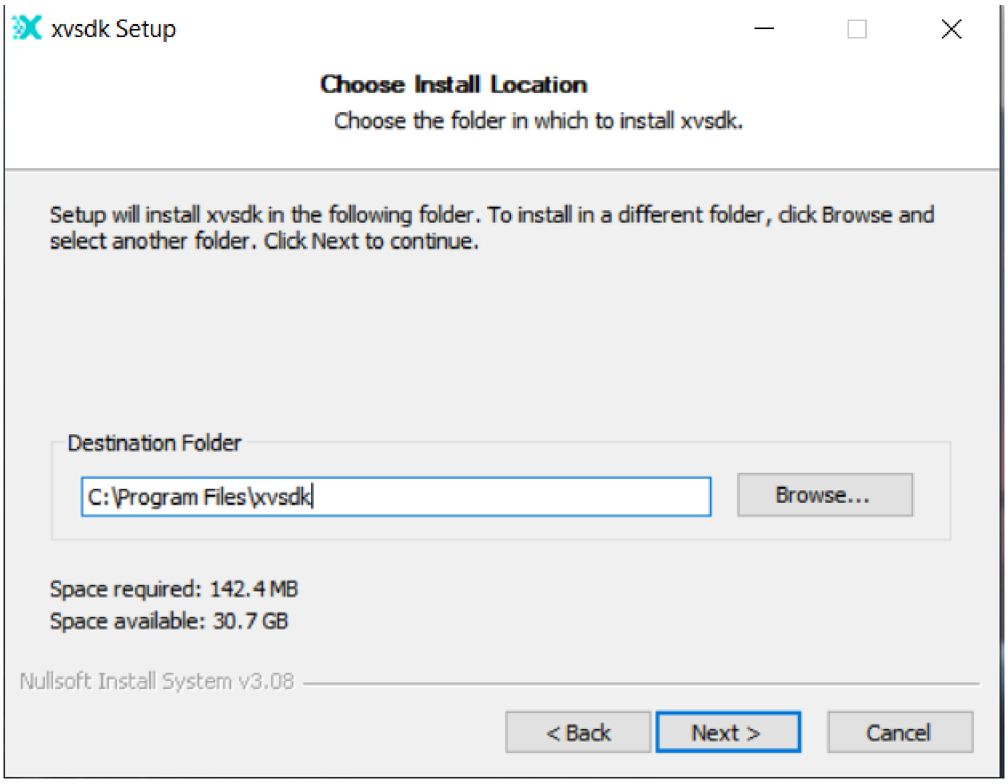
Press Page Down to see the rest of the agreement.

```
#
# Copyright (c) 2018 Xvisio Technology Corp, all rights reserved
#

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND
NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR
OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install xvsdk.

Figure 2-4 Installation2



xvsdk Setup

Choose Install Location

Choose the folder in which to install xvsdk.

Setup will install xvsdk in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

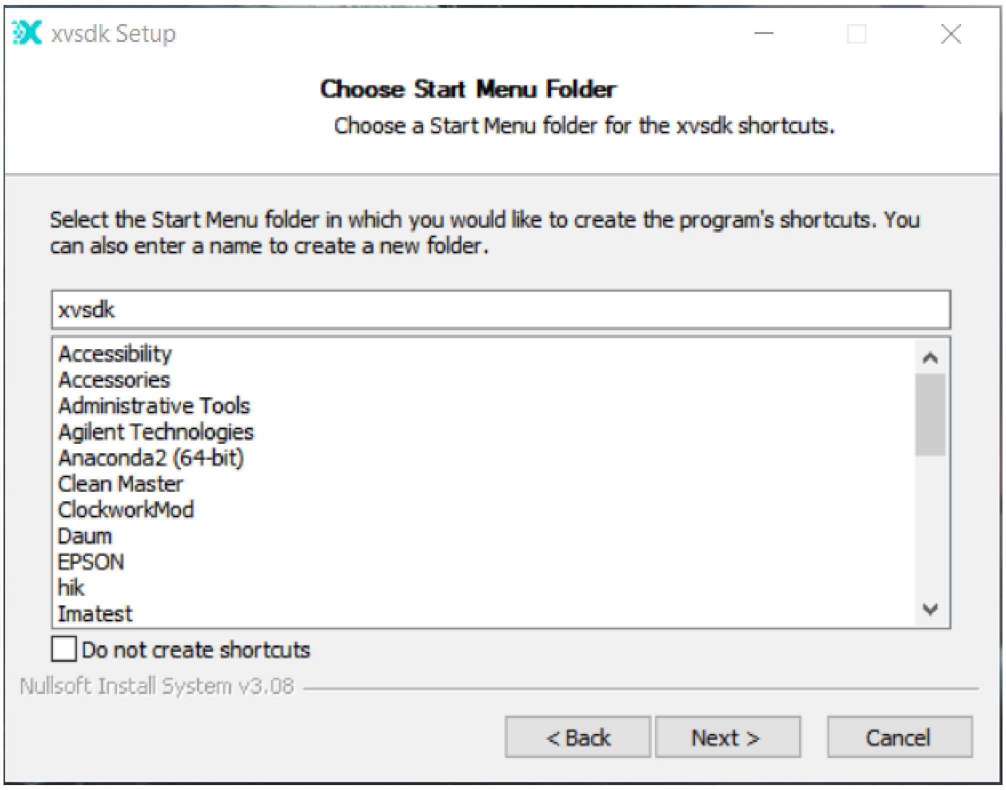
C:\Program Files\xvsdk

Space required: 142.4 MB

Space available: 30.7 GB

Nullsoft Install System v3.08

Figure 2-5 Installation 3



xvsdk Setup

Choose Start Menu Folder

Choose a Start Menu folder for the xvsdk shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

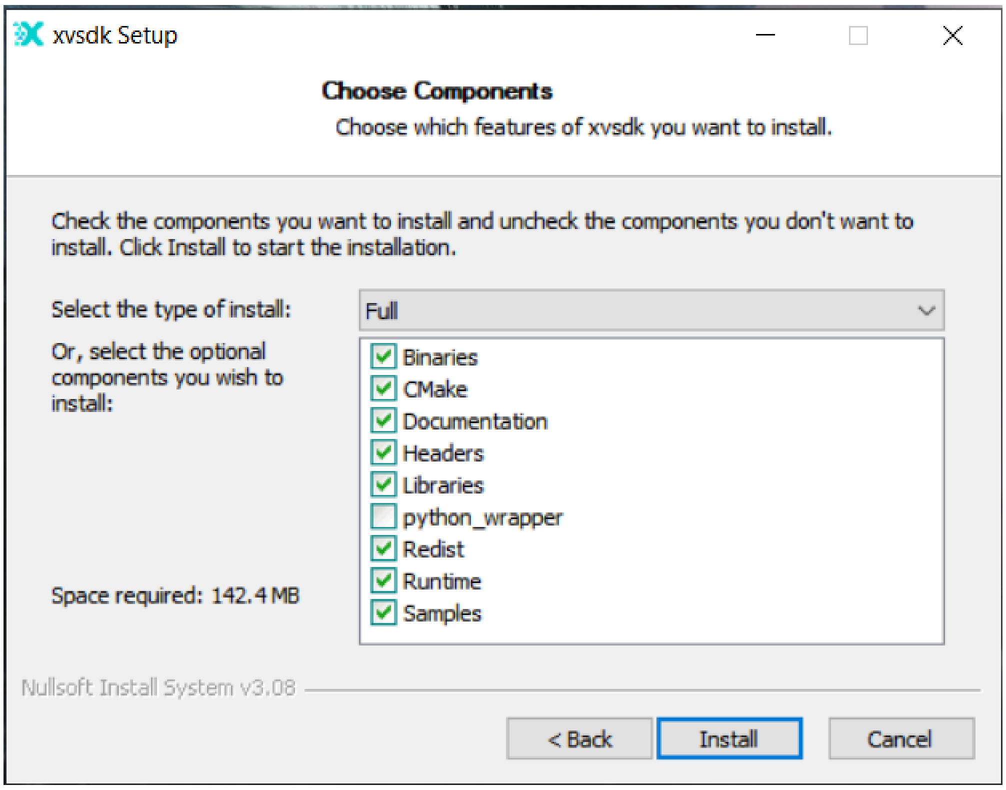
xvsdk

Accessibility
Accessories
Administrative Tools
Agilent Technologies
Anaconda2 (64-bit)
Clean Master
ClockworkMod
Daum
EPSON
hik
Imatest

☐ Do not create shortcuts

Nullsoft Install System v3.08

Figure 2-6 Installation 4



xvsdk Setup

Choose Components

Choose which features of xvsdk you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Install to start the installation.

Select the type of install: Full

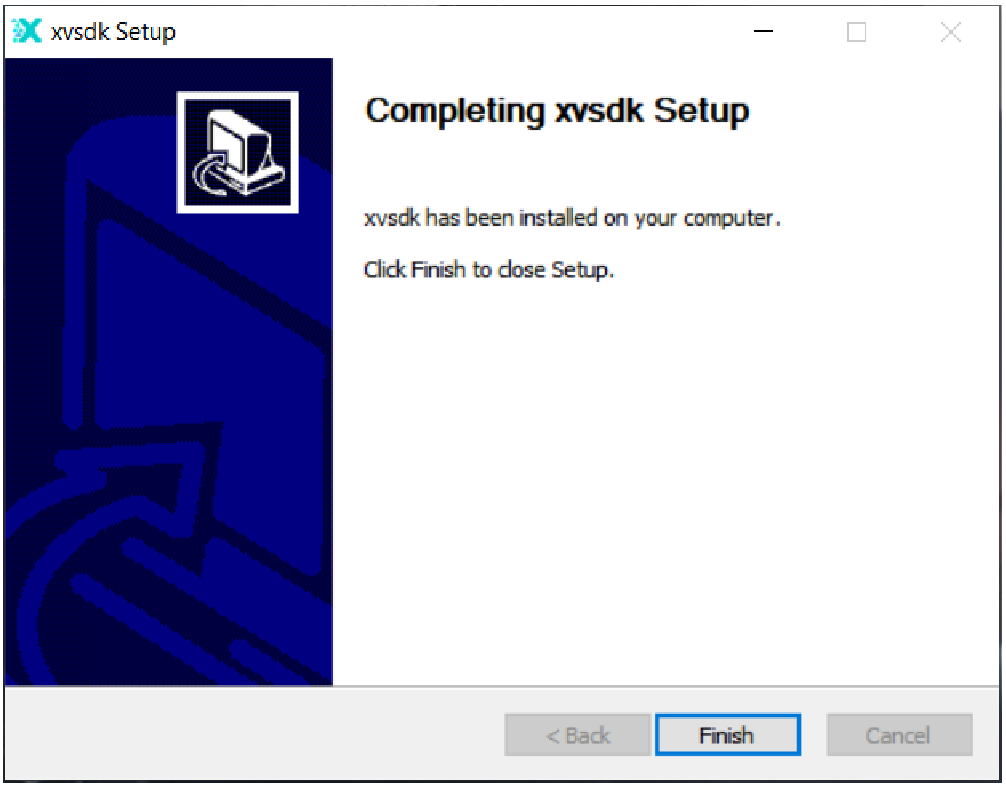
Or, select the optional components you wish to install:

- [✓] Binaries
- [✓] CMake
- [✓] Documentation
- [✓] Headers
- [✓] Libraries
- [ ] python_wrapper
- [✓] Redist
- [✓] Runtime
- [✓] Samples

Space required: 142.4 MB

Nullsoft Install System v3.08

Figure 2-7 Installation 5



xvsdk Setup

Completing xvsdk Setup

xvsdk has been installed on your computer.

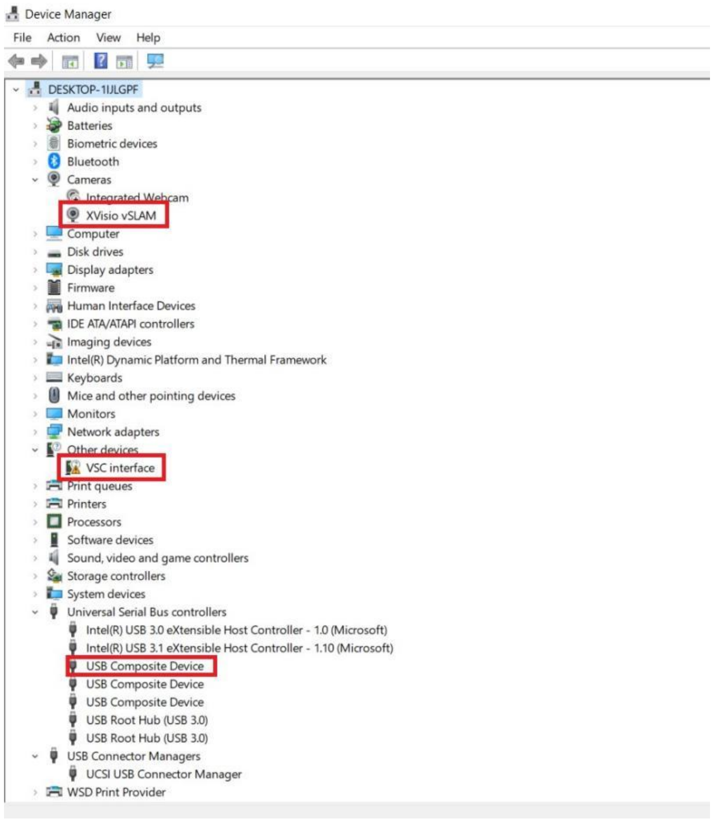
Click Finish to close Setup.

Figure 2-8 Installation 6

2.2.2 Driver Installation

1) After connecting the device with PC, open the Windows device manager.

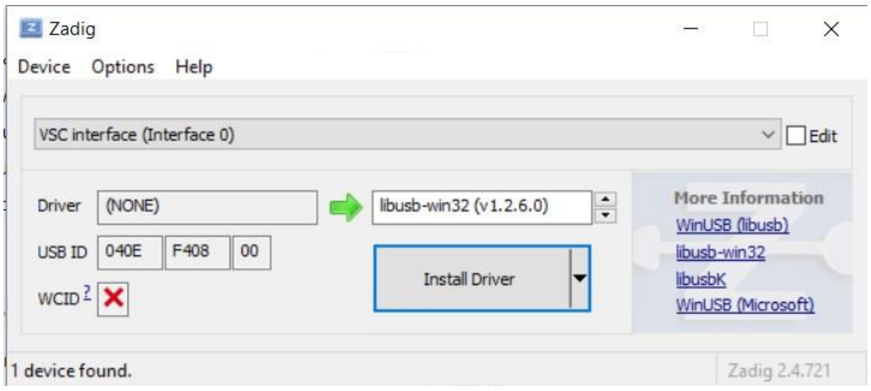
Check the USB port as below:



- DESKTOP-1IJLGPF
  - Audio inputs and outputs
  - Batteries
  - Biometric devices
  - Bluetooth
  - Cameras
    - Integrated Webcam
    - XVisio vSLAM
  - Computer
  - Disk drives
  - Display adapters
  - Firmware
  - Human Interface Devices
  - IDE ATA/ATAPI controllers
  - Imaging devices
  - Intel(R) Dynamic Platform and Thermal Framework
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - Other devices
    - VSC interface
  - Print queues
  - Printers
  - Processors
  - Software devices
  - Sound, video and game controllers
  - Storage controllers
  - System devices
  - Universal Serial Bus controllers
    - Intel(R) USB 3.0 eXtensible Host Controller - 1.0 (Microsoft)
    - Intel(R) USB 3.1 eXtensible Host Controller - 1.10 (Microsoft)
    - USB Composite Device
    - USB Composite Device
    - USB Composite Device
    - USB Root Hub (USB 3.0)
    - USB Root Hub (USB 3.0)
  - USB Connector Managers
    - UCSI USB Connector Manager
  - WSD Print Provider

Figure 2-9 Device Manager

2) Download tool "Zadig" from *https://zadig.akeo.ie/downloads/zadig-2.4.exe.* Double click "Zadig": select "VSC interface" and "libusb-win32", then click “Install Driver”.



VSC interface (Interface 0)

Driver (NONE) → libusb-win32 (v1.2.6.0)

USB ID 040E F408 00

WCID ? ❌

1 device found.

Zadig 2.4.721

Figure 2-10 Zadig

3) Installation successfully.

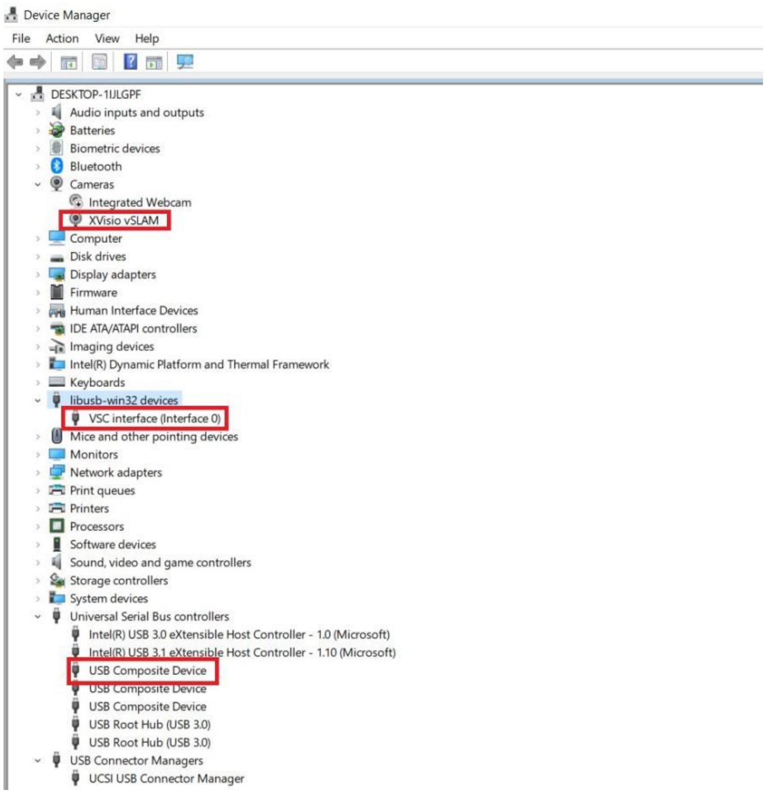
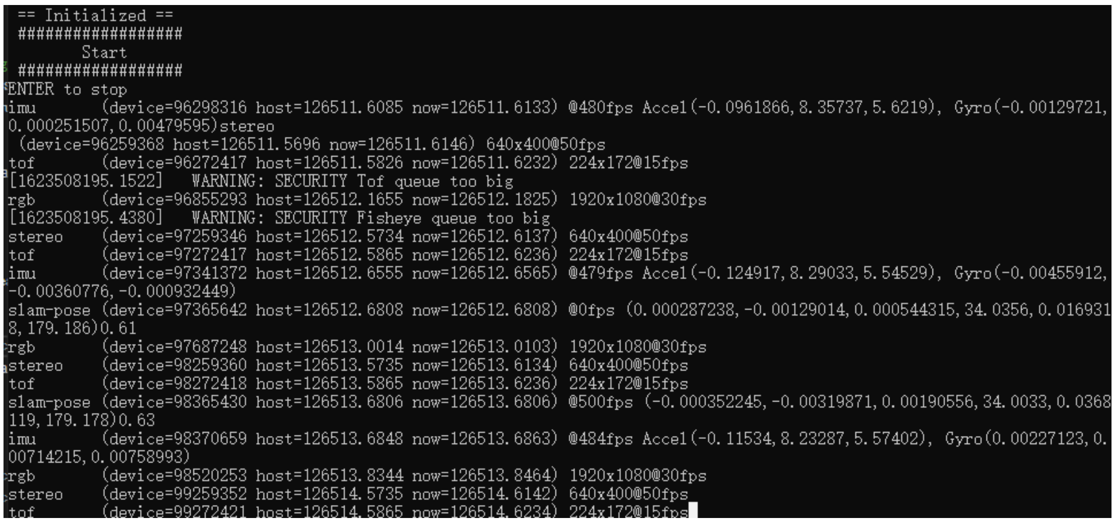


Figure 2-11 Device Manager

4) After installation, double click "all_stream.exe" (path: C:\Program Files\xvsdk\bin). If the data and image can be achieved successfully which means installation successfully.



```
 == Initialized ==
 ##################
        Start
 ##################
ENTER to stop
imu       (device=96298316 host=126511.6085 now=126511.6133) @480fps Accel(-0.0961866,8.35737,5.6219), Gyro(-0.00129721,
0.000251507,0.00479595)stereo
 (device=96259368 host=126511.5696 now=126511.6146) 640x400@50fps
tof       (device=96272417 host=126511.5826 now=126511.6232) 224x172@15fps
[1623508195.1522]   WARNING: SECURITY Tof queue too big
rgb       (device=96855293 host=126512.1655 now=126512.1825) 1920x1080@30fps
[1623508195.4380]   WARNING: SECURITY Fisheye queue too big
stereo    (device=97259346 host=126512.5734 now=126512.6137) 640x400@50fps
tof       (device=97272417 host=126512.5865 now=126512.6236) 224x172@15fps
imu       (device=97341372 host=126512.6555 now=126512.6565) @479fps Accel(-0.124917,8.29033,5.54529), Gyro(-0.00455912,
-0.00360776,-0.000932449)
slam-pose (device=97365642 host=126512.6808 now=126512.6808) @0fps (0.000287238,-0.00129014,0.000544315,34.0356,0.016931
8,179.186)0.61
rgb       (device=97687248 host=126513.0014 now=126513.0103) 1920x1080@30fps
stereo    (device=98259360 host=126513.5735 now=126513.6134) 640x400@50fps
tof       (device=98272418 host=126513.5865 now=126513.6236) 224x172@15fps
slam-pose (device=98365430 host=126513.6806 now=126513.6806) @500fps (-0.000352245,-0.00319871,0.00190556,34.0033,0.0368
119,179.178)0.63
imu       (device=98370659 host=126513.6848 now=126513.6863) @484fps Accel(-0.11534,8.23287,5.57402), Gyro(0.00227123,0.
00714215,0.00758993)
rgb       (device=98520253 host=126513.8344 now=126513.8464) 1920x1080@30fps
stereo    (device=99259352 host=126514.5735 now=126514.6142) 640x400@50fps
tof       (device=99272421 host=126514.5865 now=126514.6234) 224x172@15fps
```

Figure 2-12 Get Stream Normally

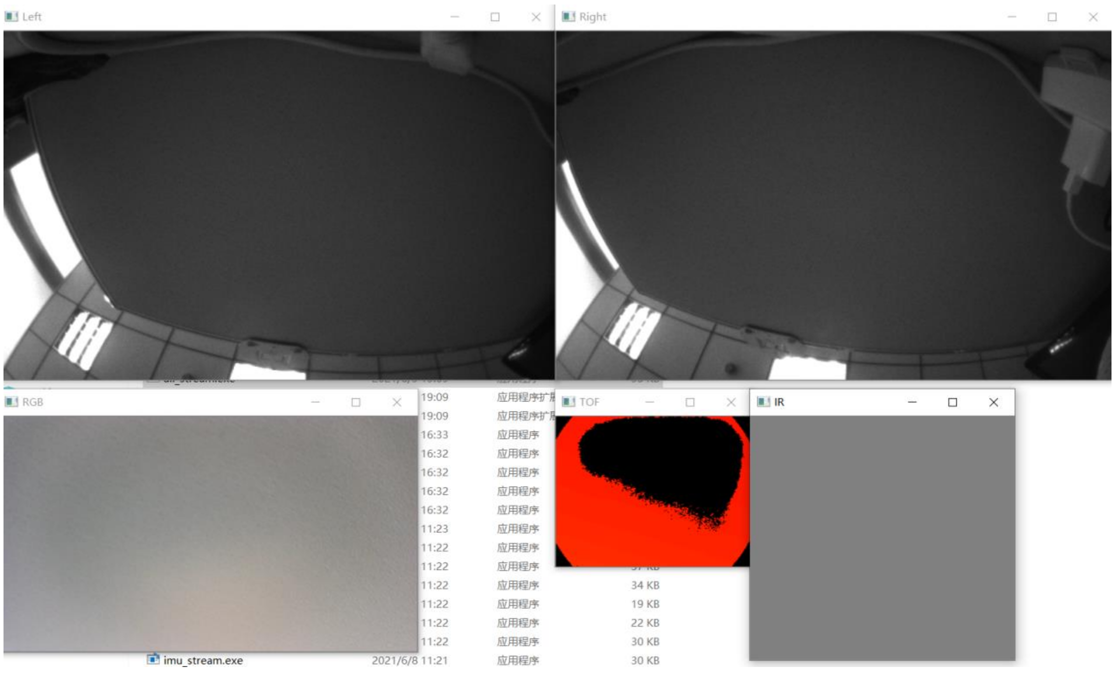


Figure 2-13 Get Image Normally

2.2.3 Windows SDK

Windows SDK contains files as below:

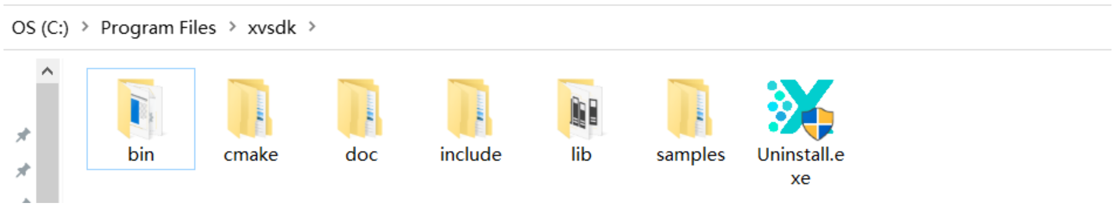


Figure 2-14 Windows SDK

“bin” contains windows tools and dynamic link library

“cmake” contains cmake configuration file.

“doc” contains HTML documents related to xvsdk class, interface and structures. Tool Doxygen needs to be pre-installed.

“include” contains header file of xv-sdk.h and xv-types.h.

“lib” contains xvsdk static library.

“samples” contains xvsdk example code.

2.3 Ubuntu SDK

2.3.1 Ubuntu SDK Installation:

Supported PC version:

- **Ubuntu**

  16.04 'Xenial'

  18.04 'Bionic'

  20.04 'Focal'

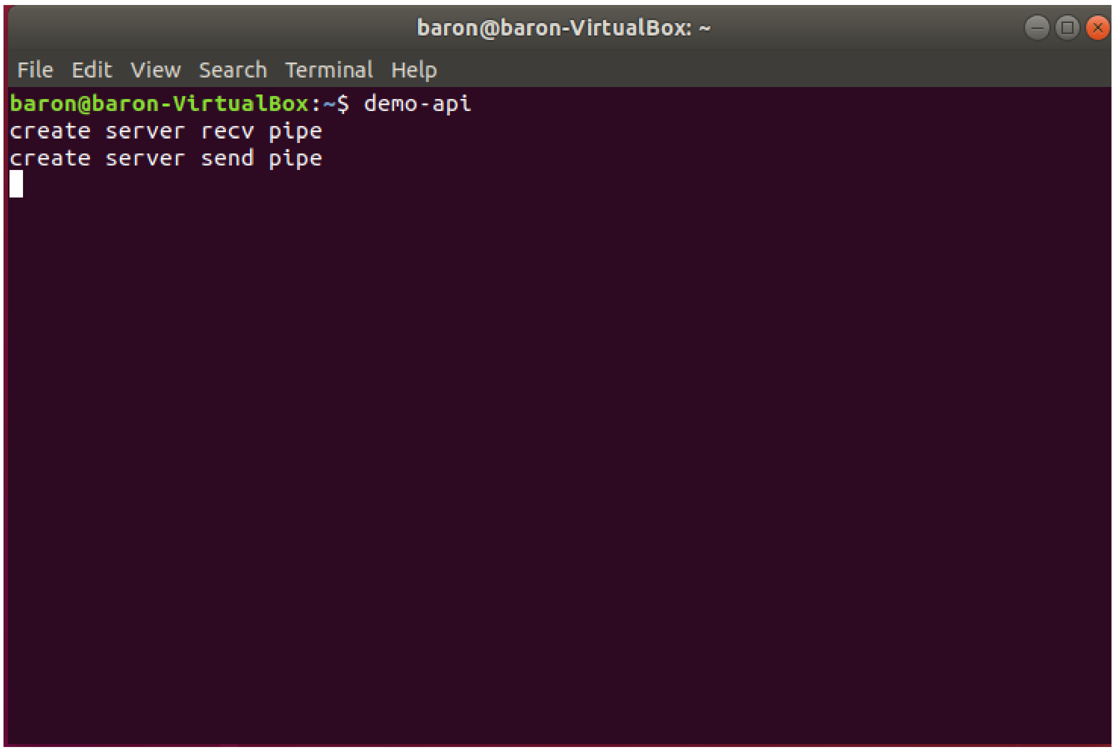
Install xvsdk by ”deb”:

sudo apt-get update

sudo apt-get install -y g++ cmake libjpeg-dev zlib1g-dev udev libopencv-core3.2 libopencv-highgui-dev liboctomap1.8 libboost-chrono-dev libboost-thread-dev libboost-filesystem-dev libboost-system-dev libboost-program-options-dev libboost-date-time-dev

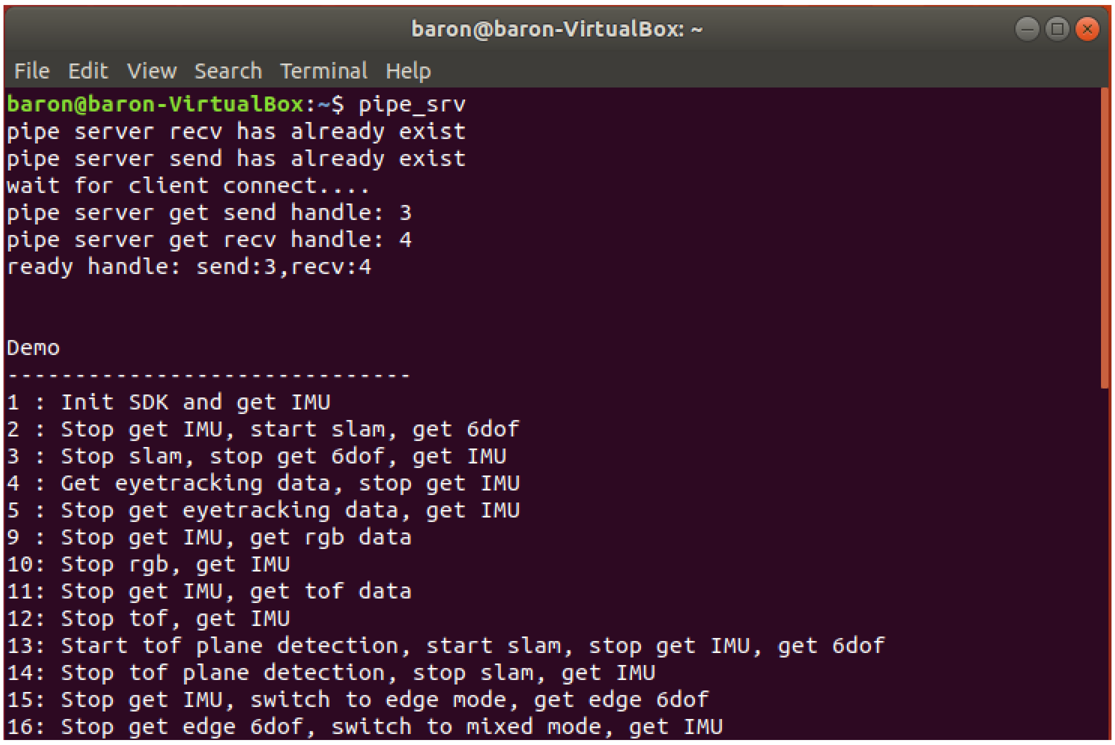
sudo dpkg -i xvsdk_3.2.0-20220304_bionic_amd64.deb

The file is saved in /usr/ after installation finished. Input the tool name in terminal to run the tool. (Note: keep demo-api and pipe_srv in the same path is needed.)


baron@baron-VirtualBox: ~
File Edit View Search Terminal Help

```
baron@baron-VirtualBox:~$ demo-api
create server recv pipe
create server send pipe
```

Figure 2-15 Run demo-api

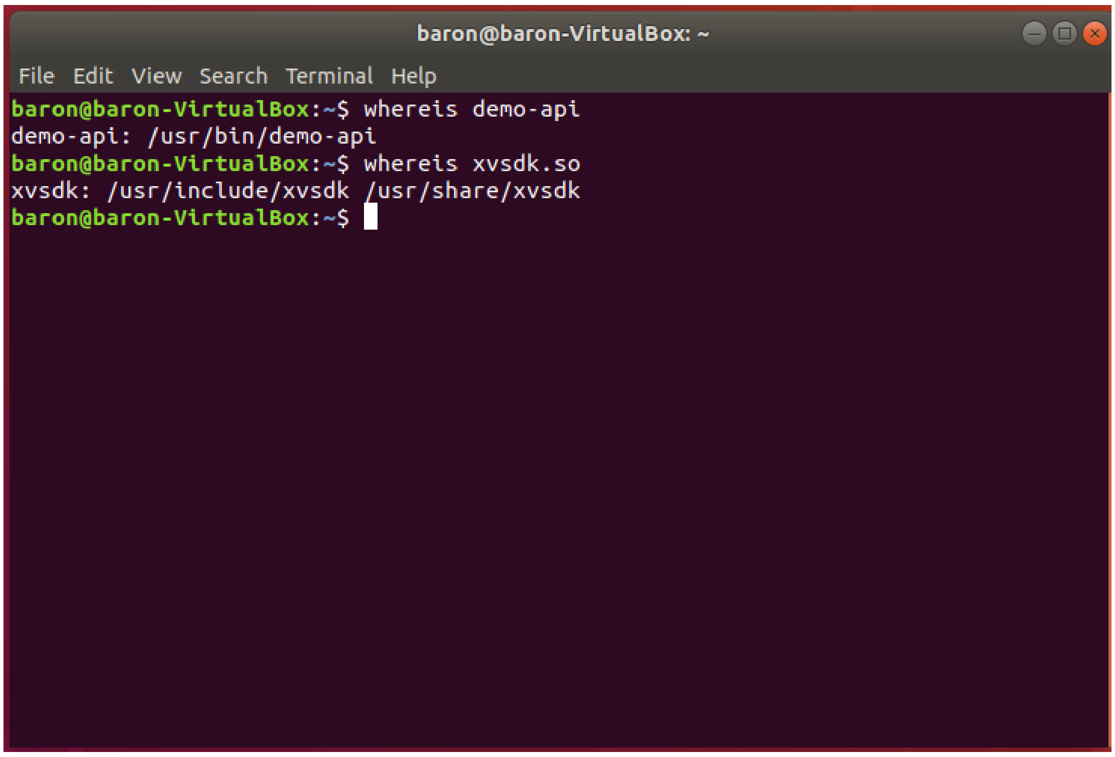

baron@baron-VirtualBox: ~
File Edit View Search Terminal Help

```
baron@baron-VirtualBox:~$ pipe_srv
pipe server recv has already exist
pipe server send has already exist
wait for client connect....
pipe server get send handle: 3
pipe server get recv handle: 4
ready handle: send:3,recv:4


Demo
------------------------------
1 : Init SDK and get IMU
2 : Stop get IMU, start slam, get 6dof
3 : Stop slam, stop get 6dof, get IMU
4 : Get eyetracking data, stop get IMU
5 : Stop get eyetracking data, get IMU
9 : Stop get IMU, get rgb data
10: Stop rgb, get IMU
11: Stop get IMU, get tof data
12: Stop tof, get IMU
13: Start tof plane detection, start slam, stop get IMU, get 6dof
14: Stop tof plane detection, stop slam, get IMU
15: Stop get IMU, switch to edge mode, get edge 6dof
16: Stop get edge 6dof, switch to mixed mode, get IMU
```

Figure 2-15 Run demo-api Normally

Command “whereis” is used to find file position.


baron@baron-VirtualBox: ~
File Edit View Search Terminal Help

```
baron@baron-VirtualBox:~$ whereis demo-api
demo-api: /usr/bin/demo-api
baron@baron-VirtualBox:~$ whereis xvsdk.so
xvsdk: /usr/include/xvsdk /usr/share/xvsdk
baron@baron-VirtualBox:~$ 
```

Figure 2-17 Find File Path

2.3.2 Ubuntu SDK Directory Structure

“bin” contains windows tools and dynamic link library

“include” contains header file of xv-sdk.h and xv-types.h.

“lib” contains xvsdk static library.

“share/doc” contains HTML documents related to xvsdk class, interface and structures. Tool Doxygen needs to be pre-installed.

“share/xvsdk” contains xvsdk sample code.

“share/ros-wrapper” contains ros sdk environment.

3. Xvisio SDK Development

This chapter sequentially introduces how to develop slam, cslam, plane detection, 3dof and other functions based on Xvisio SDK. Also introduces how to obtain camera data such as RGB, TOF, eyetracking and how to integrate audio functions. Specific example code can refer to "*examples\xslam-edge-plus-SDK\demo\demo.cpp*".

3.1 Xvisio Slam

3.1.1. Slam Mode Settings

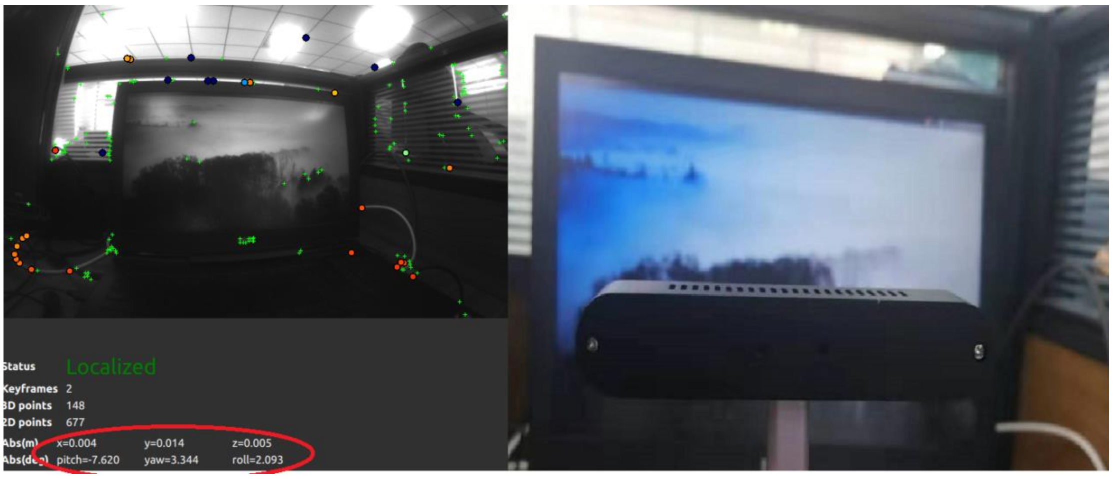
Xvisio slam supports two modes: mix mode (part of the slam algorithm runs on the device side, and some runs on the host side), and edge mode (the slam algorithm runs entirely on the device side).

```
//enable edge mode
device->slam()->start(xv::Slam::Mode::Edge);
//enable mix mode
device->slam()->start(xv::Slam::Mode::Mixed);
```

3.1.2. Slam Center Point

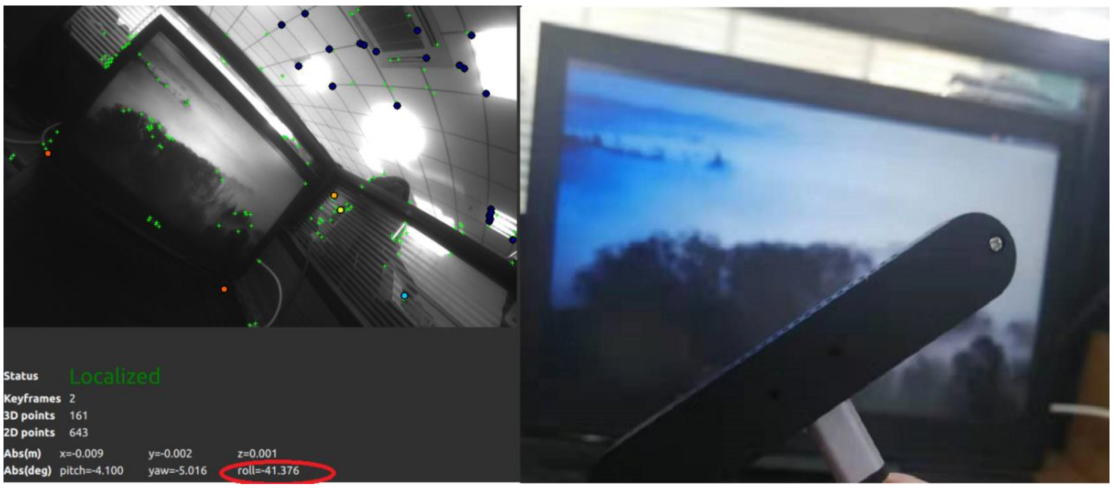
Device's 6dof center point is on IMU (6dof's xyz means IMU's translation; pitch/yaw/roll means IMU's rotation). After slam staring, a coordinate system will be established based on the device's gravity direction. The origin of the coordinate system is on IMU when slam starts (the xyz value is 0 when slam starts;

pitch/yaw/roll represent the rotation of IMU/device at that time). Refer to the figure as below:



```
Status      Localized
Keyframes  2
3D points   148
2D points   677
Abs(m)     x=0.004          y=0.014          z=0.005
Abs(deg)   pitch=-7.620     yaw=3.344        roll=2.093
```

Figure 3-1 Start the device horizontally



```
Status      Localized
Keyframes  2
3D points   161
2D points   643
Abs(m)     x=-0.009         y=-0.002         z=0.001
Abs(deg)   pitch=-4.100     yaw=-5.016       roll=-41.376
```

Figure 3-2 Device tilted to start

3.1.3. 6dof Structure

6dof data supports 3 formats (Eulerian angle, rotation matrix, quaternion) to express rotation, and the 6dof structure is as follows:

```
struct Pose : public details::PosePred_<double> {
  Pose();
  /**
  * @brief Construct a pose with a translation, rotation, timestamps and confidence.
  */
  Pose(Vector3d const& translation, Matrix3d const& rotation,
       double hostTimestamp = std::numeric_limits<double>::infinity(), std::int64_t edgeTimestamp = (std::numeric_limits<std::int64_t>::min)(), double c=0.);
  /**
   * @brief Prediction of the pose based on angular and linear velocity and acceleration.
   * @param dt amount of prediction (in s)
   * @return The predicted (2nd order extrapolation) of the orientation.
   */
  Pose prediction(double dt) const;
};
```

The time base of **hostTimestamp** is the host side, start timing with the host side boot (start timing from 0);

The time base of **deviceTimestamp** is the device side (glass or module), which starts timing with device boot (start timing from 0);

Confidence is the trustworthy level of 6dof, and value 0 means slam lost.

3.1.4. Get 6dof

Mix mode supports two ways to obtain 6dof data, one is callback, and the other one is active acquisition. The edge mode currently only supports the callback method to obtain 6dof.

1) Callback

device->slam()->registerCallback(poseCallback);

The frame rate of callback is generally determined by the frame rate of IMU (6DOF uses IMU for fusion); if user wants to get 6dof data with prediction from callback, just set filter_prediction value in the “ini” file.

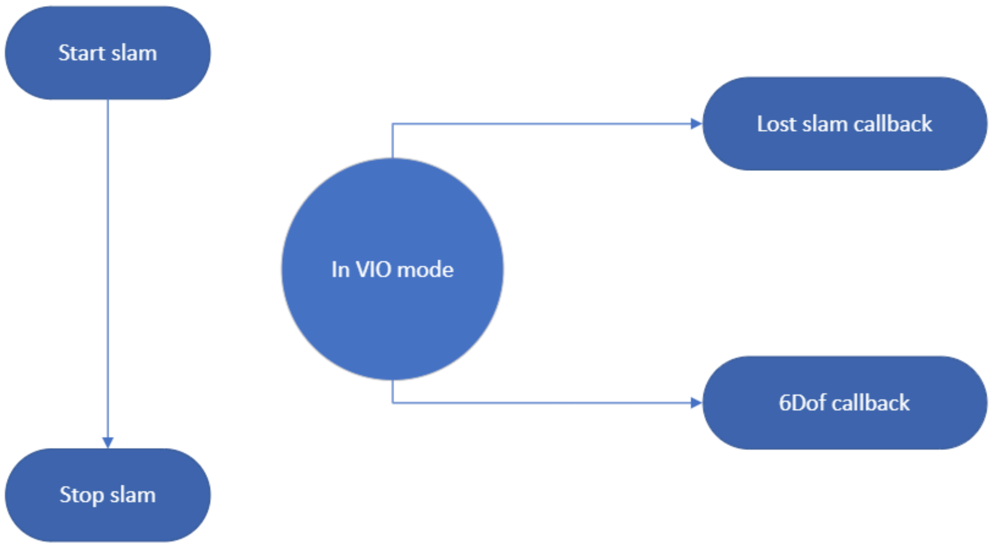
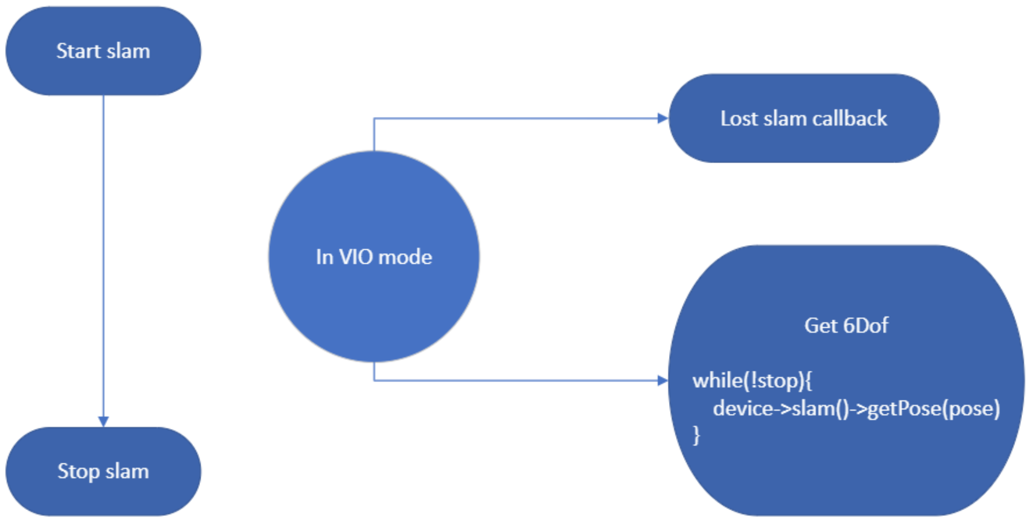


Figure3-3 callback method

2) Active acquisition

bool getPose(Pose &pose, double prediction) ;

Developers can call this interface to actively obtain 6dof data and set the prediction time at the same time, but it is recommended to be less than 0.016 (16ms, if the prediction is too large, 6dof prediction may not be accurate).



```
while(!stop){
   device->slam()->getPose(pose)
}
```

Figure3-4 Get 6Dof

3.2 VIO Mode

The following interfaces are mainly used to realize the VIO function:

bool start();

bool stop();

int registerCallback(std::function<void (xv::Pose const&)>);

bool unregisterCallback(int callbackId);

bool getPose(Pose &pose, double prediction) ;

VIO mode does not include map loopclosure. As the odometer increases, the cumulative error will increase. In normal use, it is generally recommended that users use cslam mode, first build a map, and then run slam. The following takes the initiative to obtain 6dof as an example to introduce how to pass the above. The interface runs slam, and the interface calling process is as follows:

1. Register lost callback,
2. Call start() to turn on slam,
3. Call getPose() to get 6dof,
4. Call stop() to stop slam.

For specific code, refer to demo.cpp (case 2, case 3) for further information.

3.3 CSLAM Mode

The following interfaces are mainly used to achieve CSLAM function:

bool start();

bool stop();

bool loadMapAndSwitchToCslam(std::streambuf &mapStream, std::function<void (int)> done_callback, std::function<void (float)> localized_on_reference_map);

bool saveMapAndSwitchToCslam(std::streambuf &mapStream, std::function<void (int, int)> done_callback, std::function<void (float)> localized_on_reference_map);

CSLAM steps:

Steps to build a map:

To build a good map, firstly you should consider how to use the map. The map must contain all the viewpoints required by the application. If the final application is in another room, there is no reason to record the map in the current room. Similarly, if the final application displays some virtual objects on the ground, no need to record the other view of the room. The path to be moved by the application should be the final path of the recorded map. In order to ensure a good loop closure, walk on the same path twice: for example, start from a starting point, walk away, return to the starting point, then walk away on the same path again, and then return to the starting point. During the recording process, walking twice on the same path ensures a good overlap between the different recording viewpoints of the loop closure detection. During recording, it is important to avoid moving fast or facing areas with no features.

Sample:

Refer to *demon-api.cpp* (case 6, case7) for code. The following is the API call process and schematic diagram, which are introduced according to two usage scenarios:

1. Switch to cslam after mapping

API call process:

a) Call start(),register 6dof callback

   device->slam()->start(xv::Slam::Mode::Mixed);

   device->slam()->registerCallback(poseCallback);

b) Mapping

c) After mapping, call *saveMapAndSwitchToCslam* to save the map and switch to cslam. (done_callback, localized_on_reference_map is need to register if use callback to get 6DOF)

   device->slam()->saveMapAndSwitchToCslam(mapStream, cslamSavedCallback, cslamLocalizedCallback);

d) Call stop() to stop cslam.

   device->slam()->stop();

The flow chart is shown as below:

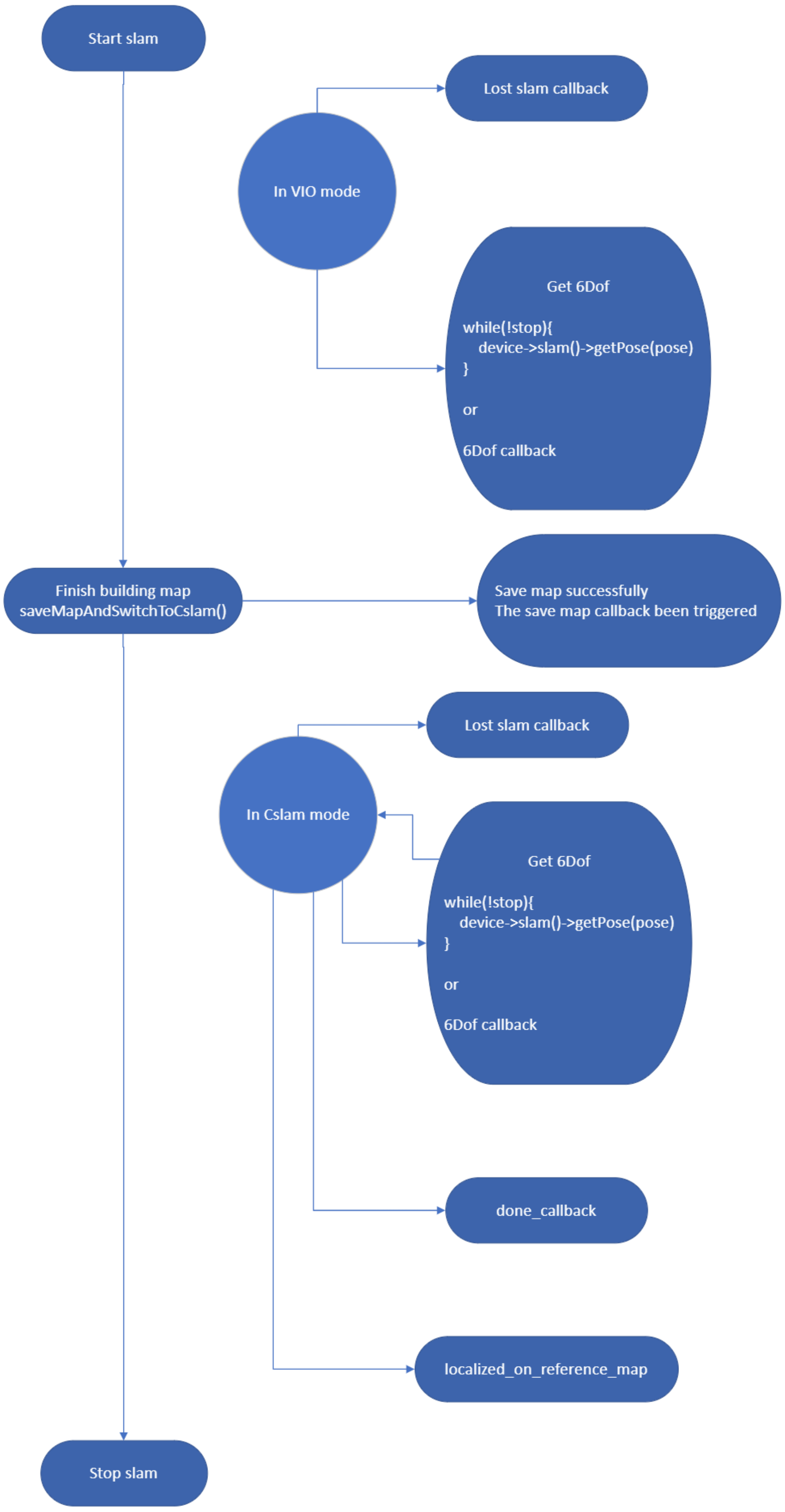


Figure3-5 Call Process

2. Switch to CSLAM after loading the map

1) Call start(), register 6dof callback (register is needed if use callback to get 6DOF),

   device->slam()->start(xv::Slam::Mode::Mixed);

   device->slam()->registerCallback(poseCallback);

2) Call loadMapAndSwitchToCslam, load map and switch to cslam (register done_callback, localized_on_reference_map if use callback to get 6DOF.)

   device->slam()->loadMapAndSwitchToCslam(mapStream,

   cslamSwitchedCallback, cslamLocalizedCallback);

3) Call stop() to stop CSLAM.

   device->slam()->stop();

The flow chart is as follows:

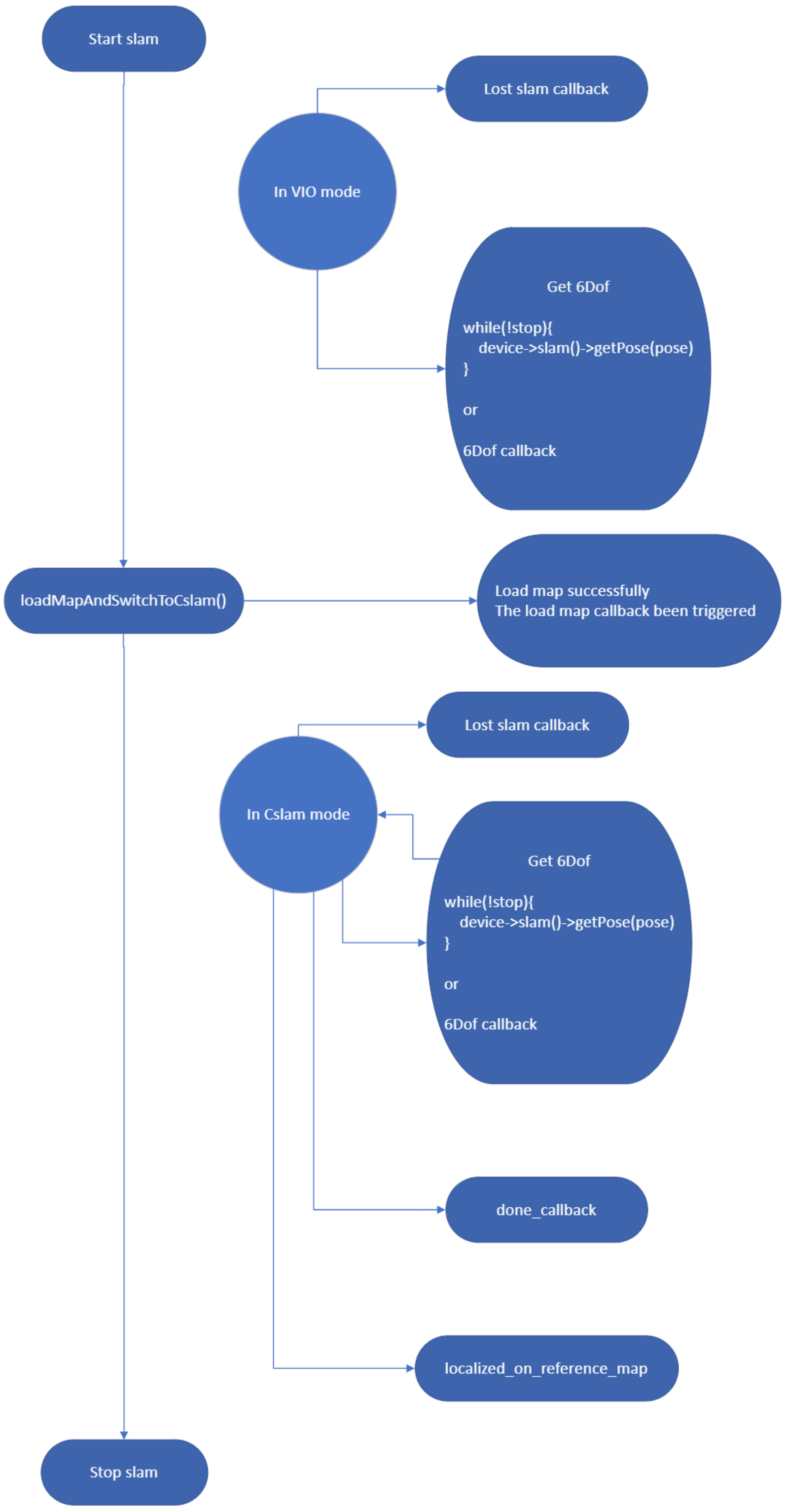

Start slam
Lost slam callback
In VIO mode
Get 6Dof
while(!stop){
device->slam()->getPose(pose)
}
or
6Dof callback
loadMapAndSwitchToCslam()
Load map successfully
The load map callback been triggered
Lost slam callback
In Cslam mode
Get 6Dof
while(!stop){
device->slam()->getPose(pose)
}
or
6Dof callback
done_callback
localized_on_reference_map
Stop slam

Figure 3-6 Call Process

3.4 3DOF（Rotation）

If APP only needs 3DOFdata, SDK can get IMU data and then convert it to 3DOF. 3DOF structure and interface are shown as below:

```
class Orientation {
        Matrix3d m_rotation;
        Vector4d m_quaternions; //!< [qx,qy,qz,qw]
        Vector3D m_angularVelocity = Vector3D{0.,0.,0.};
        Vector3D m_angularAcceleration = Vector3D{0.,0.,0.};
public:
        double hostTimestamp = std::numeric_limits<double>::infinity();
        std::int64_t edgeTimestampUs = (std::numeric_limits<std::int64_t>::min)();
        Vector4d const& quaternion() const;
        Orientation prediction(double dt) const;
     ……
     };
  int registerCallback(std::function<void (Orientation const &)>);
  bool unregisterCallback(int callbackID);
```

Sample:

Refer to demon-api.cpp (case 1, case 2) for specific code.

3.5 Plane Detection (TOF/stereo)

The interface of Plane detection is shown as below:

```
//stereo plane detection API
Int registerStereoPlanesCallback(std::function<void (std::shared_ptr<const std::vector<xv::Plane>>)> planeCallback);
bool unregisterStereoPlanesCallback(int callbackID);
//tof plane detection API
```

```
int registerTofPlanesCallback(std::function<void (std::shared_ptr<const
std::vector<Plane> >)> planeCallback);
bool unregisterTofPlanesCallback(int callbackId);
struct Plane {
    /// @brief Points lying on the plane.
    /// Array of 3D points lying on the plane that describes the
    /// polygon that borders the actually detected area.
    std::vector<Vector3d> points;
    /// @brief Unit vector normal to the plane
    Vector3d normal;
    /// @brief Signed distance to origin.
    /// Signed distance between the plane and the origin of the world. The distance
is
    /// signed according to the direction of the normale.
    double d;
    /// @brief Plane unique identifier
    std::string id;
  };
```

Plane detection needs to work with SLAM, and the position of the plane is based on the SLAM coordinate system (the world coordinate system is established with the location of the device-IMU as the origin when slam is started);

“std::vector<Vector3d> points” represents the 3D feature points of the plane. People can get the real detected plane area by connecting these 3D features. “int id” represents the plane ID. The specific opinion is shown in the figure below. Totally 3 planes are detected. By connecting 3D feature points, a specific plane area is obtained. All the planes are based on the SLAM world coordinate system.

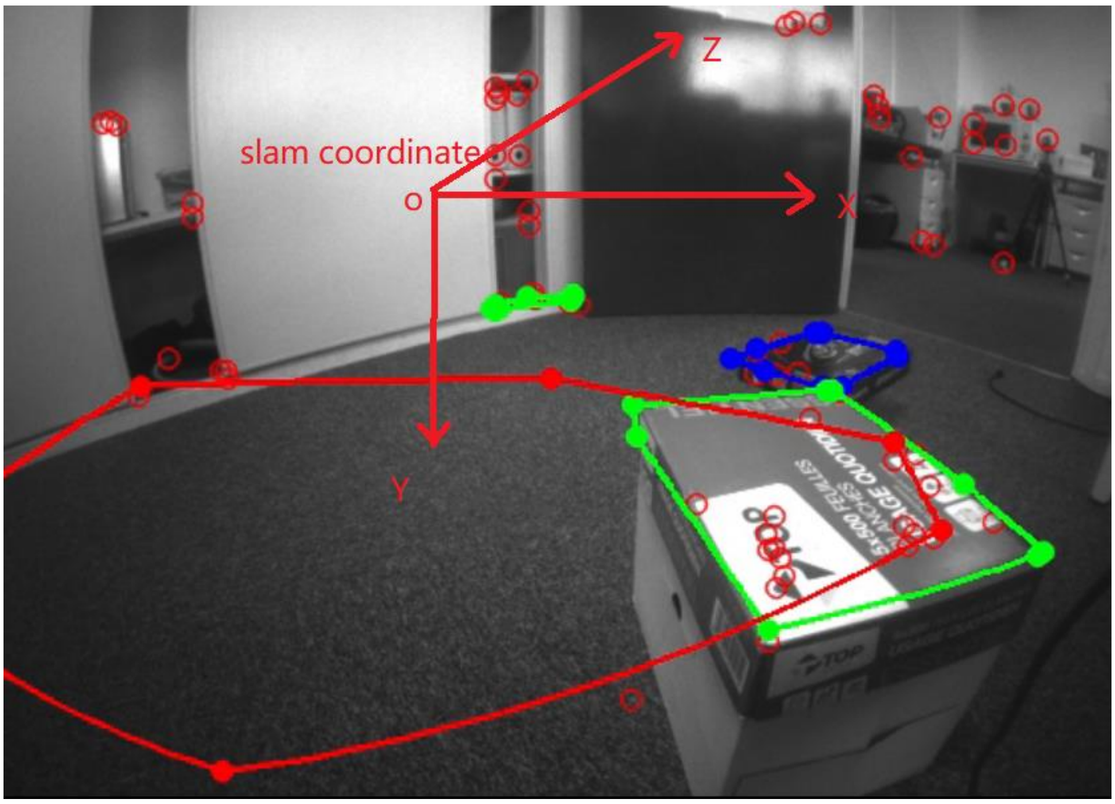


Figure3-7 World Coordinate System.

Sample:

Refer to demon-api.cpp (case13 and case14) for TOF plane detection. Refer to demon-api.cpp (case23 and case24) for stereo plane detection..

3.6 Map Sharing

The interface of Map sharing is as follows:

bool start();

bool stop();

bool loadMapAndSwitchToCslam(std::streambuf &, std::function<void (int)>, std::function<void (float)>);

Map sharing is an advanced function of CSLAM. Build map and then switch to the SLAM function. However, compared with CSLAM, the difference is that this map can be shared with other device. The application scenario is that multiple people can real-time interaction in the same scene (such as a battle game, co-working). As shown in the figure below, the core is that users are based on the same world coordinate system (using the same map).

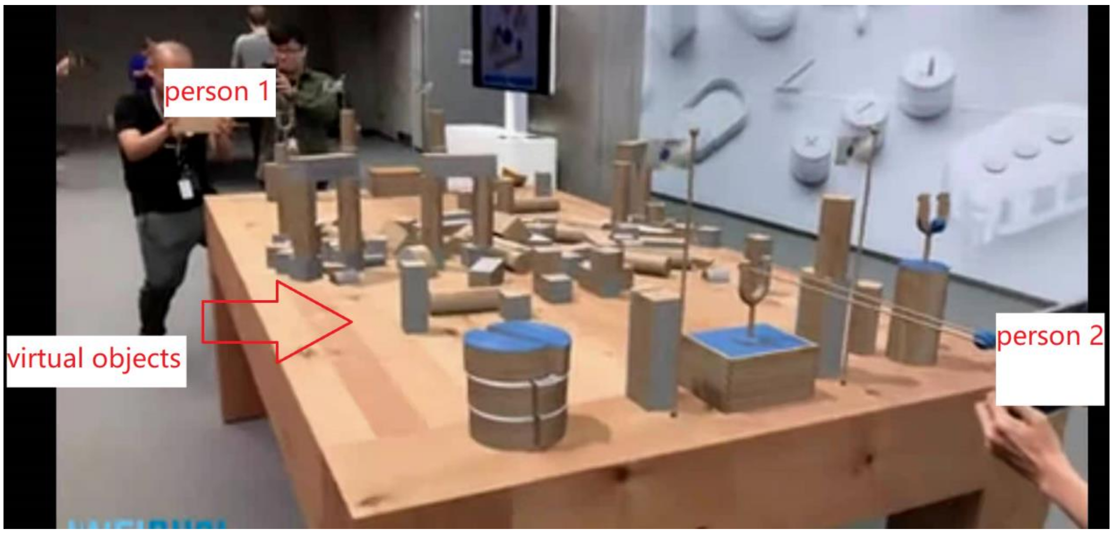


Figure 3-8 Map Sharing

Sample:

Refer to demon-api.cpp (case 19 and case20) for mapping code. Refer to mapdemon-api.cpp (case21 and case22) for using code.

3.7 HID interfaces

Many control commands of device are based on HID channels. The read and write interfaces of HID are shown as below:

bool hidWriteAndRead(const std::vector<unsigned char> &command, std::vector<unsigned char> &result)

Sample:

Introduce how to call the HID cmd interface:

m_deviceDriver->device()->hidWriteAndRead({0x02, 0xfe, 0x20, 0x0D}, result);

3.8 Calibration Parameter API

The SDK provides calibration parameter API of fisheye, RGB, TOF, and display. The external coordinate (right-hand) system of all cameras is based on IMU as the origin. Fisheye internal parameters support two types, Polynomial Distortion Model (rgb, tof) and Unified camera model (fisheye).

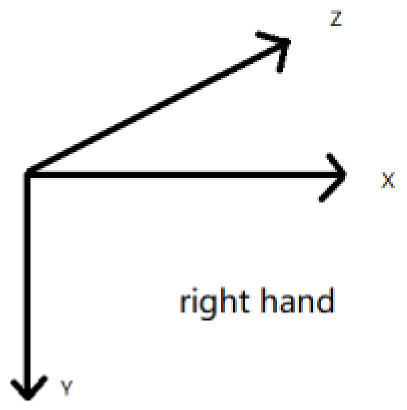


Figure 3-9 Right Hand Coordinate System

The interface is as follows:

virtual const std::vector<Calibration>& calibration();

Sample:

Refer demon-api.cpp (case37) for fisheye calibration parameter. Refer demon-api.cpp (case31) for RGB calibration parameter. Refer demon-api.cpp (case38) for TOF calibration parameter.

3.9 HOT Plug

Interfaces of Hotplug:

int registerPlugEventCallback(const std::function<void (std::shared_ptr<Device> device, PlugEventType type)> &Callback);

3.10 Get Fisheye Data

Fisheye frequency:

```
enum class ResolutionMode {
    LOW = 1, ///< Low resolution (typically QVGA)
    MEDIUM = 2, ///< Medium resolution (typically VGA)
    HIGH = 3 ///< High resolution (typically HD 720)
 };
```

Fisheye interfaces:

virtual int registerCallback(std::function<void (T)>) = 0;

virtual bool unregisterCallback(int callbackId) = 0;

3.11 Get RGB Data

The interface of RGB is as follows:

virtual int registerCallback(std::function<void (T)>) = 0;

virtual bool unregisterCallback(int callbackId) = 0;

Sample:

Refer to demon-api.cpp (case 9) for getting RGB data.

3.12 Get TOF Data

Introduction of TOF mode:

Frequency: single frequency (SF) | double frequency (DF)

Mode: IQ | M2 | edge

Frequency: 5-30 FPS

Different mode corresponds to different FPS (note that FPS setting items will not output the real setting mode in some modes.

IQ DF: 30FPS

IQ SF: 30FPS

M2 DF: 4.5FPS

M2 SF: 13FPS

M2 DF: 3.5FPS

M2 SF: 7FPS

The interface of TOF is as follows:

virtual int registerCallback(std::function<void (T)>) = 0;

virtual bool unregisterCallback(int callbackId) = 0;

virtual bool setStreamMode(StreamMode mode);

virtual bool setDistanceMode(DistanceMode mode);

enum class StreamMode { DepthOnly = 0, CloudOnly, DepthAndCloud, None };

enum class DistanceMode { Short = 0, Middle, Long };

Sample:

Refer to demon-api (case 11, case 39,case 40) for getting RGB data.

3.13 Get RGBD Data

Interfaces of RGBD:

int registerColorDepthImageCallback(std::function<void(const DepthColorImage&)>);

virtual bool unregisterColorDepthImage(int callbackId);

struct DepthColorImage {

  std::size_t width = 0; //!< width of the image (in pixel)

  std::size_t height = 0; //!< height of the image (in pixel)

  std::shared_ptr<const std::uint8_t> data = nullptr; //! RGBD = RGB + Depth Map. image data of RGB-D pixels : RGB (3 bytes) D (float 4bytes)

  double hostTimestamp = std::numeric_limits<double>::infinity();

};

Sample:

Note: Both enable colorCamera and TOFCamers to get RGBD data. Refer to demon-api.cpp (case9) to get RGB data.

3.14 CNN

3.14.1 CNN Interfaces:

virtual bool setDescriptor(const std::string &filepath) = 0;

virtual bool setModel(const std::string &filepath) = 0;

virtual bool setSource(const Source &source) = 0;

virtual xv::ObjectDetector::Source getSource() const = 0;

virtual xv::ObjectDescriptor getDescriptor() const = 0;

enum class Source { LEFT = 0, RIGHT, RGB, TOF };

Sample:

Refer demon-api.cpp (case42) for CNN function.

3.10.1 Xvisio AI

- **Dependency:**

  OpenVINO Toolkit

  Download OpenVINO from *https://docs.openvinotoolkit.org/latest/index.html*.

Install OpenVINO and the engineering environment. The installed path is /opt/intel/.

- **Process:**

1) Get model and convert model. Get .blob model file.

Download "OpenVINO Model Zoo" from: *https://github.com/openvinotoolkit/open_model_zoo/blob/master/models/intel/index.md*

Use the model download tool of OpenVINO Toolkit to download:

cd /opt/intel/openvino_2021/deployment_tools/tools/model_downloader/python downloader.py --name human-pose-estimation-0001

cd intel/human-pose-estimation-0001/

(the model format is OpenVINO IR(.bin & .xml). Take FP16 format as an example, copy folder FP16 to workspace.)

cp -r FP16 /home/workspace/example/

cd /home/workspace/example/

Convert OpenVINO IR(.bin & .xml) to Myraid VPU(.blob), for example:

/opt/intel/openvino_2021/deployment_tools/inference_engine/lib/intel64/myriad_compile -m FP16/human-pose-estimation-0001.xml -o humanpose.blob -ip U8 -VPU_NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_SLICES 4

(Note: more command parameter configurations can be found on OpenVINO's official website)

2) config.json

"config.json" is used for enable xvsdk to obtain the name of the model correctly and the corresponding parsing method. So that to get the correct results.

For example for pose detection:

```
{
    "model_type":"tensorflow",
    "classes":["background",
    "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair",
    "cow", "diningtable", "dog", "horse",
    "motorbike", "person", "pottedplant",
    "sheep", "sofa", "train", "tvmonitor" ],
    "threshold":0.45,

    "video":"video2",
    "model":"/home/baron/Desktop/xvsdk-release-
3.1.0/xvsdk/install/bin/CNN_2x8x_r14_5.blob", (absolute path of .blob)
    "source":"RGB",
    "CNN_input_flip_stereo":"true",
    "CNN_input_flip_RGB":"false",
    "CNN_input_flip_TOF":"false"
}
```

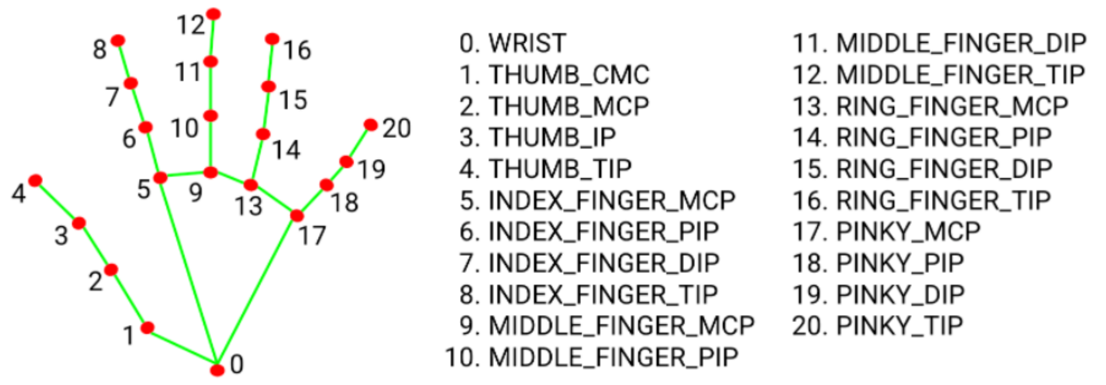
3) After configuring .blob and .json, put .blob and .json into the default path (/data/down/) which set by demo-api. Run the demo-api of xvsdk. Select No.42 to set CNN function of different cameras.

3.15 Gesture Function

3.15.1 MNN

Push MNN model file (include hand_landmark.mnn, palm_detection_pb.mnn) to default path /etc/xvisio/gesture/. New MNN model file path can be set by setConfigPath after pushing the model file.

21 feature points of gesture:



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig 2. 21 hand landmarks.

3.15.2 Interfaces of Gesture:

struct keypoint {

float x = -1;

float y = -1;

float z = -1;

};

struct GestureData {

int index[2] = {-1,-1};//gesture index value, left hand first and then right hand

keypoint position[2];//position of gesture, left hand first and then right hand. Default of zero feature points position.

double hostTimestamp = std::numeric_limits<double>::infinity(); //start timing (0) from host booting.

std::int64_t edgeTimestampUs = std::numeric_limits<std::int64_t>::min();start timing (0) from device booting

float distance;//reserved to keep the moving distance of dynamic gesture.

float confidence;//reserved to keep confidence。

};

virtual void setConfigPath(std::string config) = 0; //set path of MNN configuration file

virtual int registerDynamicGestureCallback(std::function<void (GestureData const &)>) = 0; //register dynamic gesture callback, get index value of dynamic gesture.

virtual bool UnregisterDynamicGestureCallback(int callbackID) = 0;//unregister dynamic gesture callback

virtual int registerKeypointsCallback(std::function<void (std::shared_ptr<const std::vector<keypoint>>)> callback) = 0;//register gesture 21Dof callback which based on rgb coordinate system and 2D coordinate.

virtual bool unregisterKeypointsCallback(int callbackId) = 0;//unregister gesture 21Dof callback

virtual int registerSlamKeypointsCallback(std::function<void (std::shared_ptr<const std::vector<keypoint>>)> callback) = 0;//reserved to get gesture 21Dof&3D coordinate value which based on slam coordinate system.

virtual bool unregisterSlamKeypointsCallback(int callbackId) = 0;//reserved to cancel the callback which based on slam coordinate system.

3.15.3 Example Code

Refer to demo-api (case 43-47) test items.

3.16 Get SGBM Data

SGBM interfaces:

virtual int registerCallback(std::function<void (T)>) = 0;

virtual bool unregisterCallback(int callbackId) = 0;

virtual Mode mode() const = 0;//set SGBM mode interface，0-Hardware，1-Software

virtual bool start(const std::string &sgbmConfig) = 0;

virtual bool start(const sgbm_config &sgbmConfig) = 0;

virtual bool setConfig(const std::string &sgbmConfig) = 0;

SGBM default start configuration information:

static struct xv::sgbm_config global_config = {

0, //enable_dewarp

3.5, //dewarp_zoom_factor

1, //enable_disparity

1, //enable_depth

0, //enable_point_cloud

```
    0.11285, //baseline
    69, //fov
    255, //disparity_confidence_threshold
    {1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0}, //homography
    0, //enable_gamma
    2.2, //gamma_value
    0, //enable_gaussian
    0, //mode
    5000, //max_distance
    100, //min_distance
};
```

Sample:

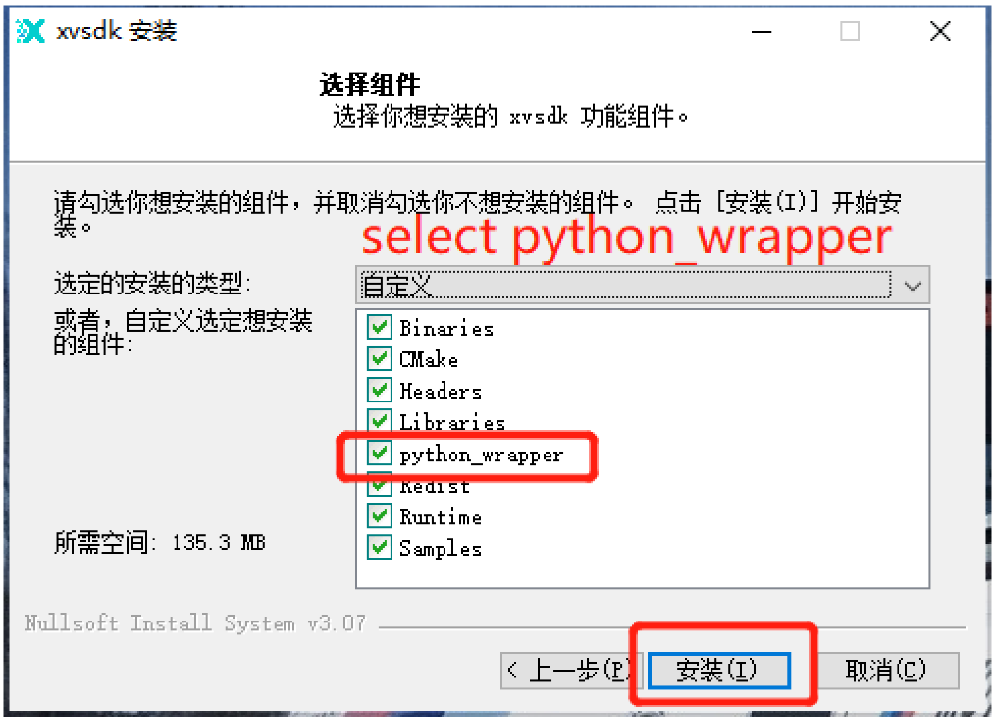
Get SGBM data from demon-api.cpp (case58，59)

3.17 Python-wrapper

Provide python interface to get images or parameter data of SLAM, IMU, fisheye, RGB, TOF, SGBM and etc.

Python-wrapper only support Windows OS:

1) Install windows xvsdk, select “python-wrapper”:


xvsdk 安装

选择组件

选择你想安装的 xvsdk 功能组件。

请勾选你想安装的组件，并取消勾选你不想安装的组件。 点击 [安装(I)] 开始安装。

select python_wrapper

选定的安装的类型: 自定义

或者，自定义选定想安装的组件:

- [✓] Binaries
- [✓] CMake
- [✓] Headers
- [✓] Libraries
- [✓] python_wrapper
- [✓] Redist
- [✓] Runtime
- [✓] Samples

所需空间: 135.3 MB

Nullsoft Install System v3.07

Figure 3-10 XVSDK

2) The installation path \bin\python-wrapper contains PythonDemo.py.

3) Connecting the device with PC after installing python3.9, use cmd to run python.