



## Xvisio\_SDK\_Guide 文档

---

v1.2/ 2022.03

***Xvisio Confidential***

## **Copyright and Proprietary Information Notice**

Copyright © 2018 Xvisio Ltd. All right reserved. This document contains confidential and proprietary information that is the property of Xvisio Ltd. All other product or company names may be trademarks of their respective owners.

Xvisio Ltd.

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

<http://www.Xvisiotech.com>

## 目录

1.	概况.....	4
2.	Xvisio SDK 环境搭建.....	4
2.1	Android SDK 环境搭建.....	4
2.2	Windows SDK 环境搭建.....	6
2.3	Ubuntu SDK 环境搭建.....	12
3.	Xvisio SDK 开发.....	14
3.1	Xvisio SLAM.....	15
3.2	VIO 模式.....	18
3.3	CSLAM 模式.....	19
3.4	3DOF (Rotation) .....	24
3.5	平面检测(TOF/stereo).....	24
3.6	地图共享.....	26
3.7	HID 接口调用.....	27
3.8	标定参数 API.....	27
3.9	热插拔处理.....	28
3.10	获取 Fishseye 数据.....	28
3.11	获取 RGB 数据.....	29
3.12	获取 TOF 数据.....	29
3.13	获取 RGBD 数据.....	30
3.14	CNN 功能介绍.....	30
3.15	手势功能介绍.....	32
3.16	获取 SGBM 数据.....	34
3.17	Python-wrapper 功能.....	35

# 1. 概况

本文档是 Xvisio SDK 的说明文档，主要是帮助开发者熟悉 Xvisio SDK，并能够快速开发相关应用（SLAM、camera、Audio...）。本文档主要分为两部分：SDK 环境搭建和 sample code 介绍。

Xvisio SDK 目前主要支持 3 个平台：Android、Ubuntu 和 Windows。主要区别是.so 针对这 3 个平台分别编译，但 API 都是相同的。第二章节主要介绍如何在这 3 个平台上安装使用 Xvisio SDK。第三章节介绍如何调用 SDK API 实现不同功能，不区分具体平台。

## 2. Xvisio SDK 环境搭建

本章依次介绍如何在 Android、Ubuntu、Windows 这三个平台上安装使用 Xvisio SDK，以及相关注意事项。

### 2.1 Android SDK 环境搭建

#### 2.1.1 Android SDK 目录结构

如下图所示，SDK 主要包含以下文件：

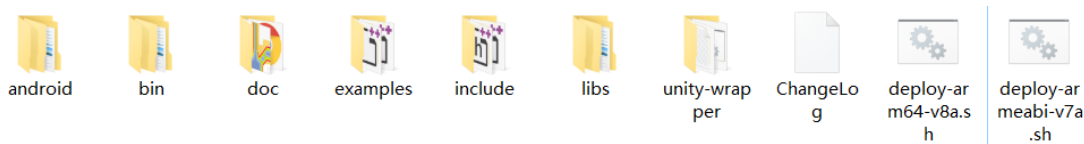


图 2-1 SDK 目录结构

“Android”文件夹包含 NDK 编译的 makefile，开发者如果修改 samples 文件夹中的 demo code，可以使用“ndk-build -j 5”编译生成对应的可执行文件，编译出的文件路径在/Android/libs/arm64-v8a(armeabi-v7a)。

“bin”文件夹是 64bit 和 32bit 的工具。

“doc”文件夹包含各个类和接口的定义文档。

“examples”文件夹是我们的 demo code，主要是如何使用我们 SDK API 的示例。

“include”文件夹是 SDK API 的头文件。

“libs”文件夹包含 SDK so 文件，“arm64-v8a”是 64bit 库，“armeabi-v7a”是 32bit 库。

“deploy.sh”是 push 脚本文件，执行“./deploy-arm64-v8a.sh”会将 64bit 库 push 到 Android platform；执行“./deploy-armeabi-v7a.sh”会将 32bit 库 push 到 Android platform。在 deploy.sh 修改“libdir”可以修改 push lib 路径。

## 2.1.2 Android box requirements

- OS 在 Android 7 及以上；
- Android kernel 支持 hid 和 uvc 两个模块；
- Android OS 是 userdebug 版本；
- 如果只需要 SLAM 功能，USB2.0 可以满足要求；如果需要全功能（SLAM/RGB/TOF/Audio），必须要 USB3.0 及以上。

## 2.1.3 Android 环境测试验证

我们以 Android 原生系统 box（root 权限）为例，介绍如何测试验证 Xvisio Android SDK。

Box 开机后，通过 usb 口连接 PC，执行如下命令 将 Xvisio SDK push 到 box 中：

```
adb root
```

```
adb remount
```

```
cd android_xxx_sdk
```

```
adb push libs /data/test/
```

```
adb push bin /data/test/
```

Xvisio device 连接 box 后执行如下命令：

*lsusb*（确认是否可以枚举到 f408 设备，如果枚举到，说明 device 已成功连接 box）

```
adb shell
```

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

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

另开一个终端：*adb shell*，*cd /data/test/bin/arm64-v8a/*，

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

参考测试命令依次输入指令（输入 1 获取 3DOF 数据，输入 2 获取 6DOF 数据.....），以上是 Android SDK 的测试验证步骤，如果最终能够正常获取 3DOF，6DOF 以及其他 sensor 数据，说明 Android 环境没有问题，后续开发者可以将 SDK 集成到 Android 系统中。

## 2.2 Windows SDK 环境搭建

### 2.2.1 XVSDK 安装

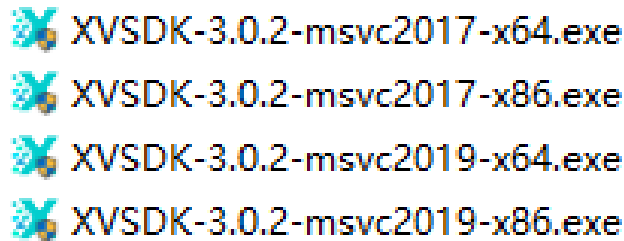


图 2-2 XVSDK 安装

根据环境选择所需安装包文件，双击运行，下一步直至安装完成。

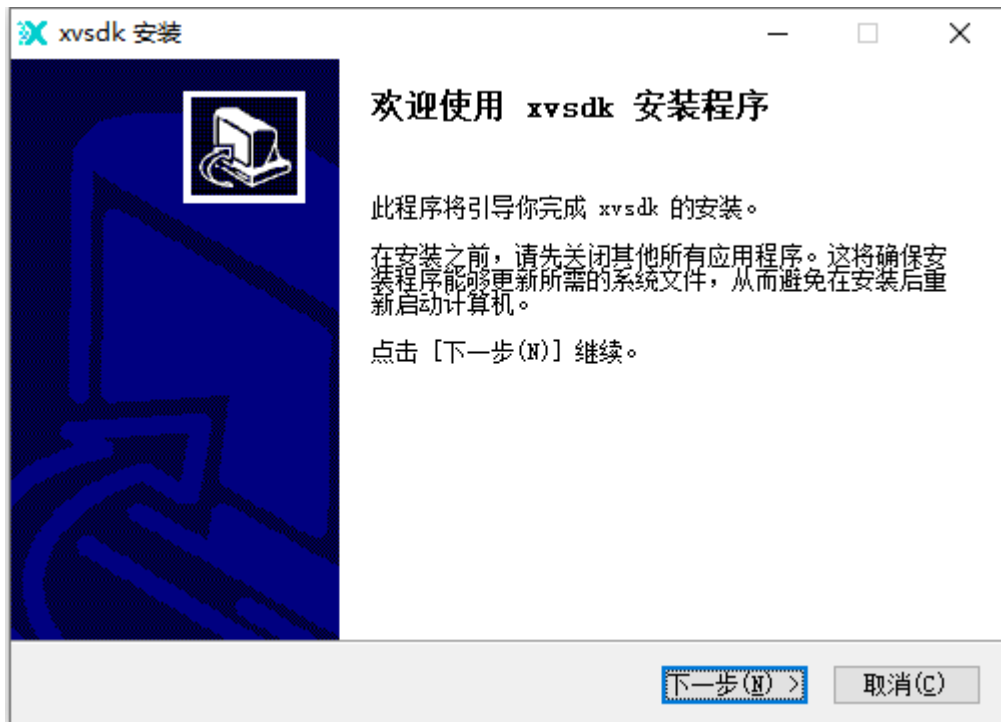


图 2-3 安装 1



图 2-4 安装 2

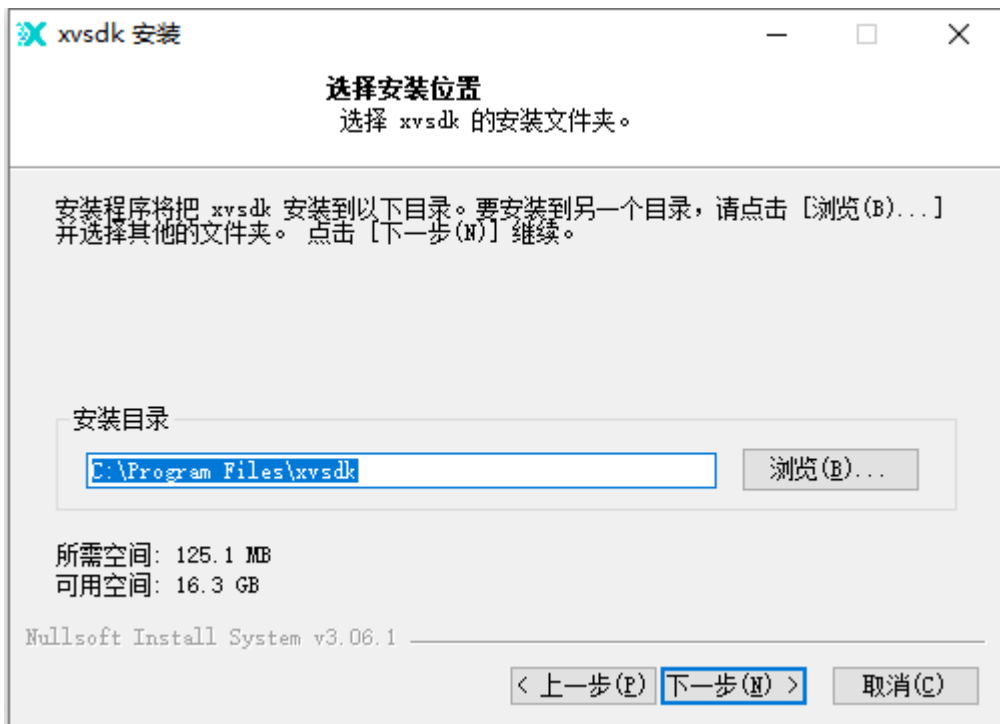


图 2-5 安装 3

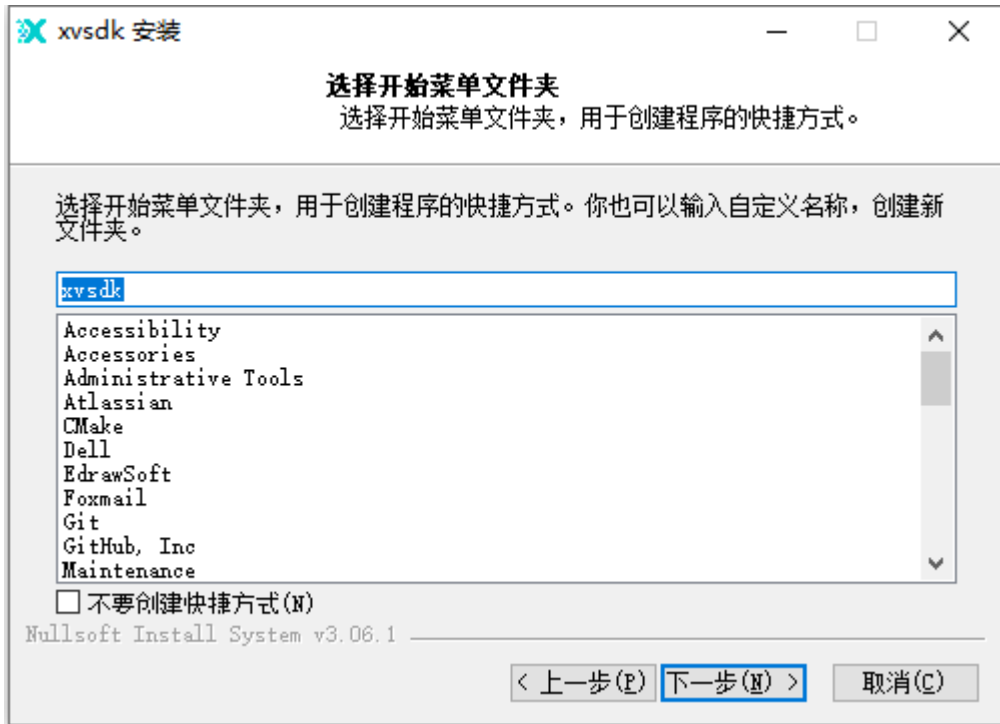


图 2-6 安装 4

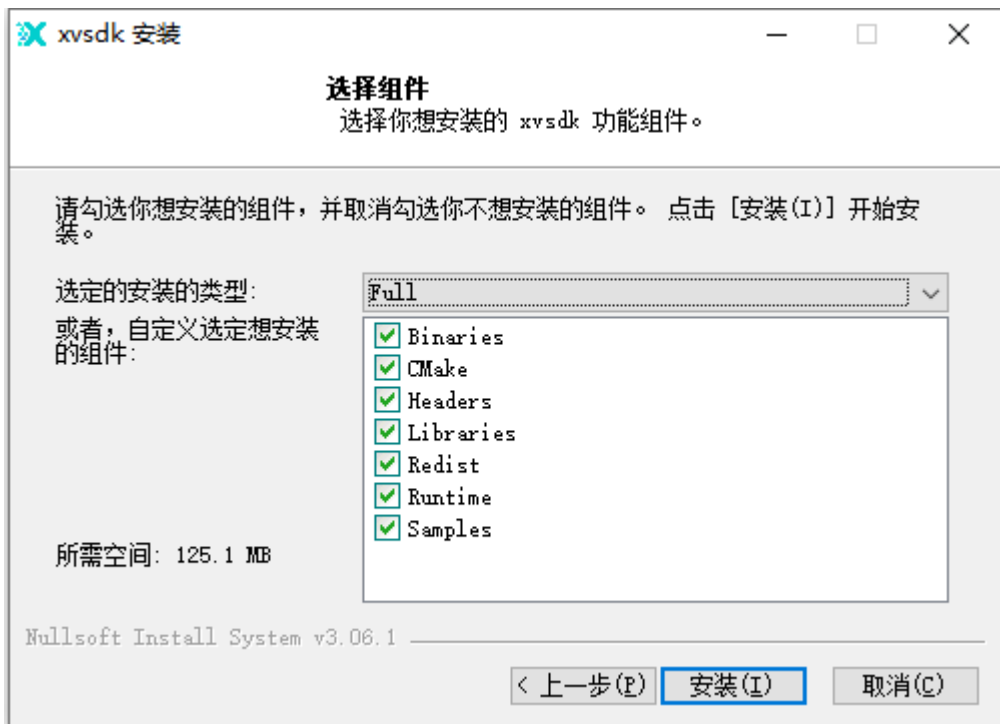


图 2-7 安装 5





图 2-8 安装 6

## 2.2.2 驱动安装

- 1) 打开 Windows 设备管理器，连接模组，可以看到以下三个设备。

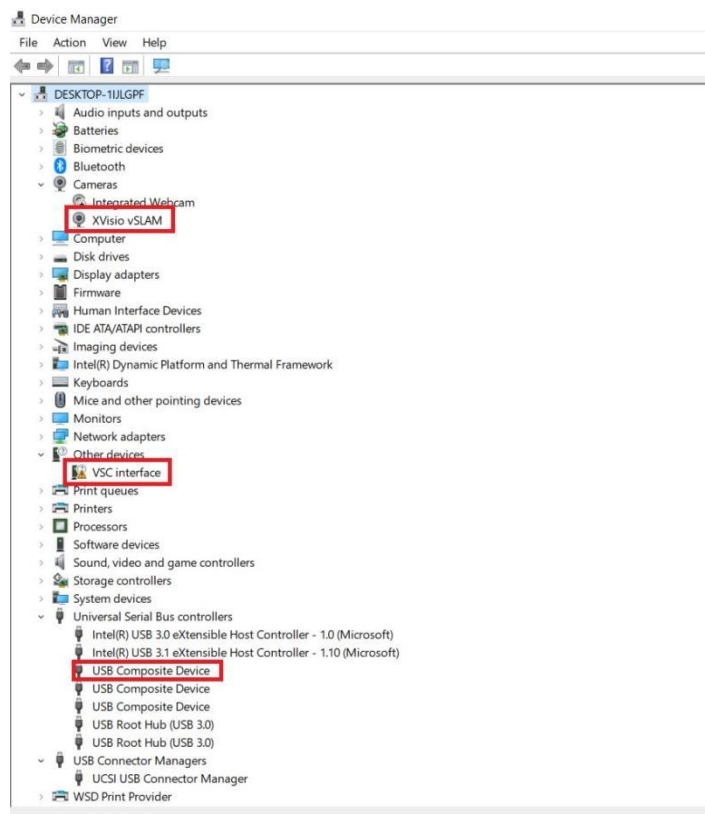


图 2-9 设备管理器

- 2) 下载并打开 Zadig 软件 (<https://zadig.akeo.ie/downloads/zadig-2.4.exe>)  
选择: VSC interface -> libusb-win32, 然后单击 Install Driver.

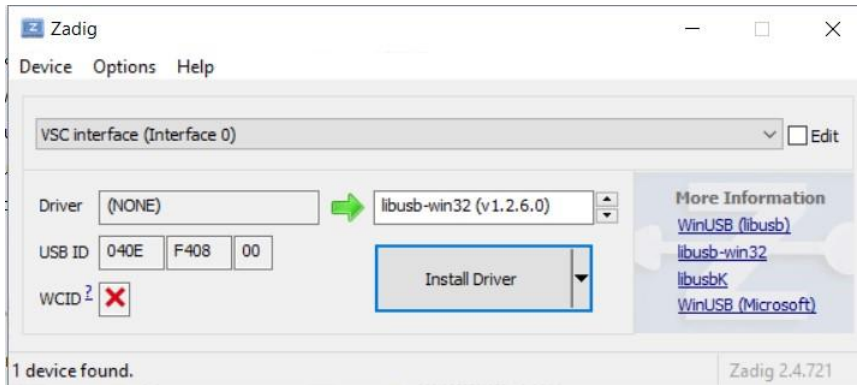


图 2-10 zadig

- 3) 安装成功后, 会显示以下内容。

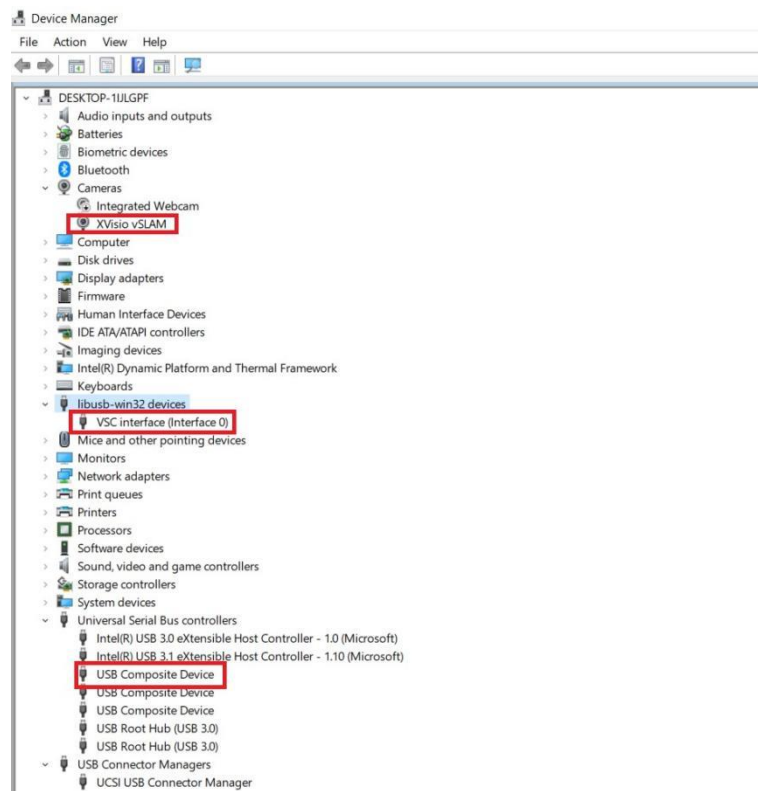


图 2-11 设备管理器

- 4) 安装过后, 插入模组, 打开安装路径 C:\Program Files\xv sdk\bin 下的 all\_stream.exe。能正常获取数据, 显示图像即安装成功。

```

== 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=97385642 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

```

图 2-12 正常获取数据

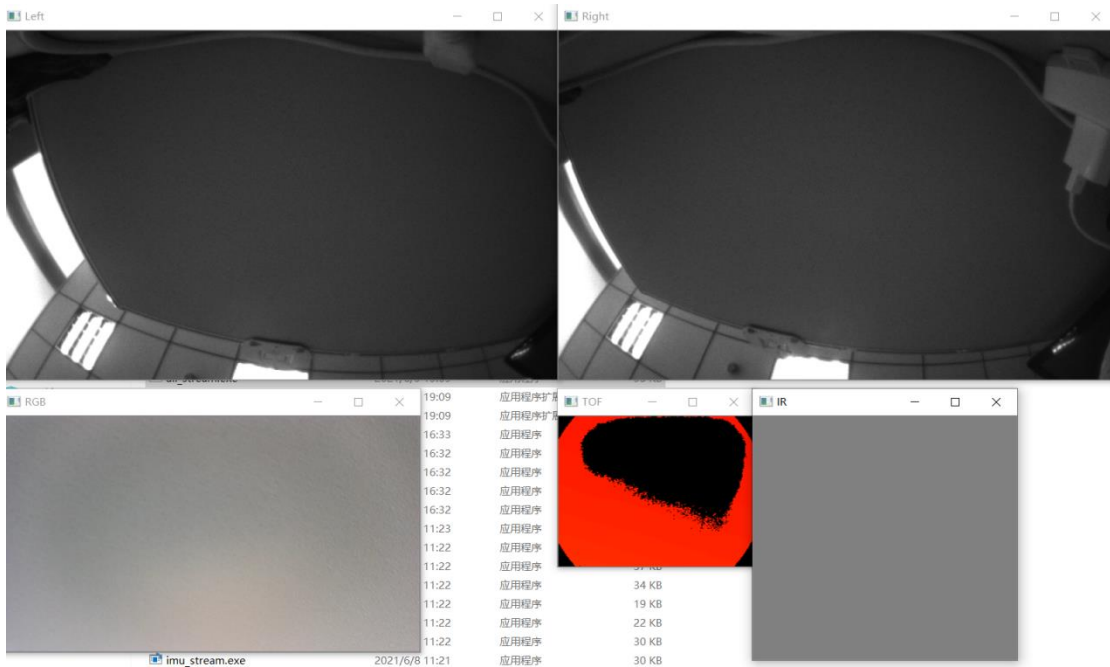


图 2-13 正常出图

### 2.2.3 Windows sdk 目录结构:

如下图所示， Windows SDK 主要包含以下目录:

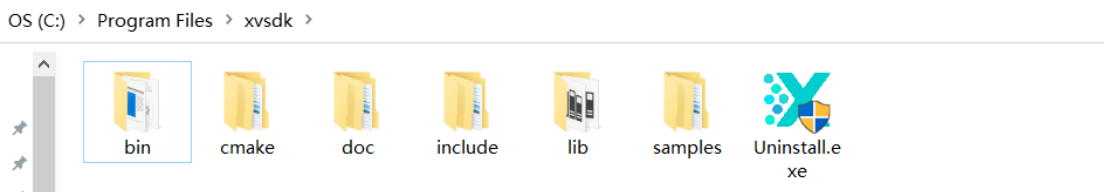


图 2-14 Windows SDK

bin: 包含 windows 工具及动态链接库。

cmake: 包含 cmake 配置文件。

doc: 包含 xvsdk 类, 接口及结构体相关 html 文档。(需提前安装 Doxygen 工具)

include: 包含 xv-sdk.h 及 xv-types.h 头文件

lib: 包含 xvsdk 静态库

samples: 包含 xvsdk 示例代码

## 2.3 Ubuntu SDK 环境搭建

### 2.3.1 Ubuntu sdk 安装:

支持的分发版本:

- Ubuntu
  - 16.04 'Xenial'
  - 18.04 'Bionic'
  - 20.04 'Focal'

通过 deb 包安装 XVSDK:

```
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
```

安装完成后, 文件存放在系统路径/usr/中, 在任意位置打开终端输入工具名称即可运行。(注: demo-api 以及 pipe\_srv 需注意在同一目录下运行。)

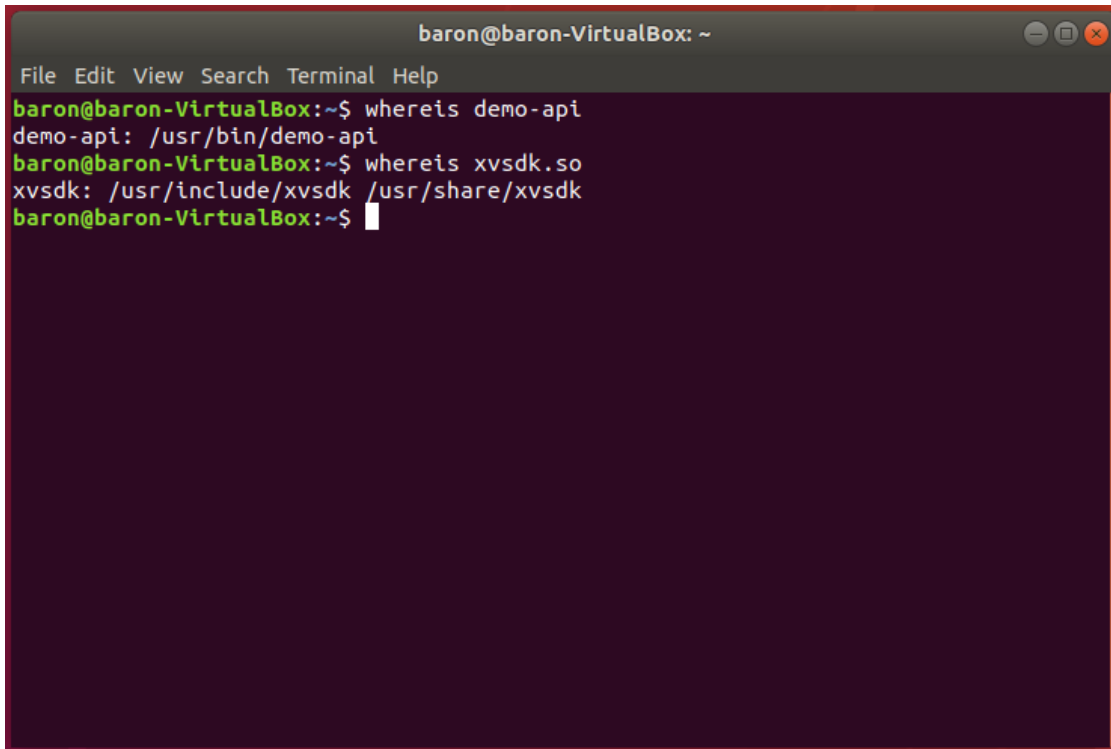
```
baron@baron-VirtualBox: ~  
File Edit View Search Terminal Help  
baron@baron-VirtualBox:~$ demo-api  
create server recv pipe  
create server send pipe  
█
```

图 2-15 运行 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
```

图 2-16 正常运行

如需查找文件位置，可使用 whereis 命令查询。



```
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 xv sdk.so  
xv sdk: /usr/include/xv sdk /usr/share/xv sdk  
baron@baron-VirtualBox:~$
```

图 2-17 查找文件位置

### 2.3.2 Ubuntu sdk 目录结构:

bin: 包含 windows 工具及动态链接库。

include: 包含 xv-sdk.h 及 xv-types.h 头文件

lib: 包含 xv sdk 静态库

share/doc: 包含 xv sdk 类, 接口及结构体相关 html 文档。(需提前安装 Doxygen 工具)

share/xv sdk: 包含 xv sdk 示例代码

share/ros-wrapper: 包含 ros sdk 环境

## 3.Xvisio SDK 开发

本章节依次介绍如何基于 Xvisio SDK 开发 SLAM、CSLAM、平面检测、3DOF 等功能, 同时介绍如何获取 RGB、TOF、eyetracking 等 camera 数据, 如何集成 Audio 功能。具体示例 code 可以参考“examples\demo-api\demo-api.cpp”。

## 3.1 Xvisio SLAM

### 3.1.1 SLAM 模式设置

Xvisio SLAM 支持两种模式：一种是 mix mode（SLAM 算法一部分运行在 device 端，一部分运行在 host 端），另外一种是 edge mode（SLAM 算法完全跑在 device 端）。

```
//enable edge mode
```

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

```
//enable mix mode
```

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

### 3.1.2 SLAM 中心点

Device 的 6DOF 中心点在 imu 上（6DOF 的 xyz 表示 imu 的 translation; pitch,yaw,roll 表示 imu 的 rotation），SLAM 启动后会基于 device 的重力方向建立坐标系，坐标系原点在 SLAM 启动时 imu 上（SLAM 启动时，xyz value 值为 0; pitch,yaw,roll 表示当时 imu（device）的 rotation），如下图所示：

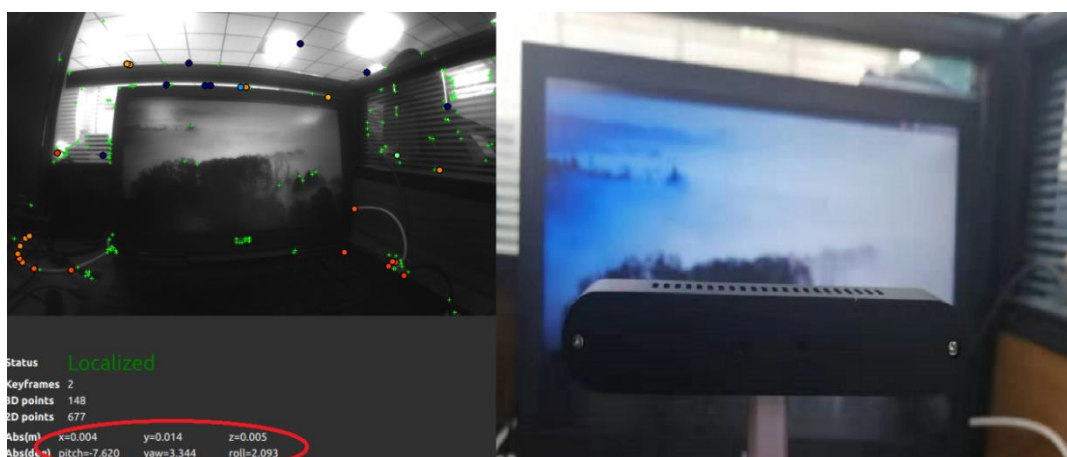


图 3-1 device 水平放置启动



图 3-2 device 倾斜放置启动

### 3.1.3 6DOF 数据结构

6DOF 数据支持 3 种格式（欧拉角，旋转矩阵，四元数）表示 rotation，6DOF 结构体如下所示：

```
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;
};
```

**hostTimestamp** 的 time base 是 host 端，以 host 端 boot 开始计时（从 0 开始计



时)；

**deviceTimestamp** 的 time base 是 device 端，以 device boot 开始计时（从 0 开始计时）；

**confidence** 是 6dof 的可信赖 level,值为 0 表示 lost。

### 3.1.4 6DOF 获取

Mix mode 支持两种方式获取 6DOF 数据：一种是 callback 方式，另外一种 是主动获取。edge mode 目前只支持 callback 方式获取 6DOF。

#### 1) callback 方式

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

callback 的帧率一般由 imu 的帧率决定（6DOF 使用 imu 做 fusion）；如果想从 callback 获取到带预测的 6DOF 数据，可以在 ini 文件中设置 filter\_prediction value。

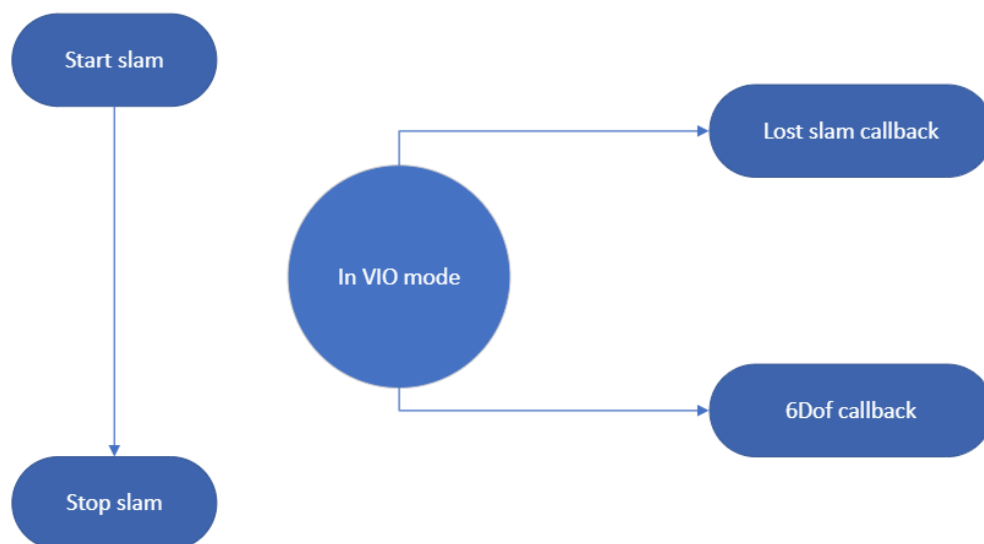


图 3-3 callback 方式

#### 2) 主动获取

*bool getPose(Pose &pose, double prediction);*

开发者可以调用这个接口，主动获取 6DOF 数据，同时可以设置 prediction time，但建议小于 0.016（16ms，如果 prediction 过大，6DOF 预测不一定准确）。

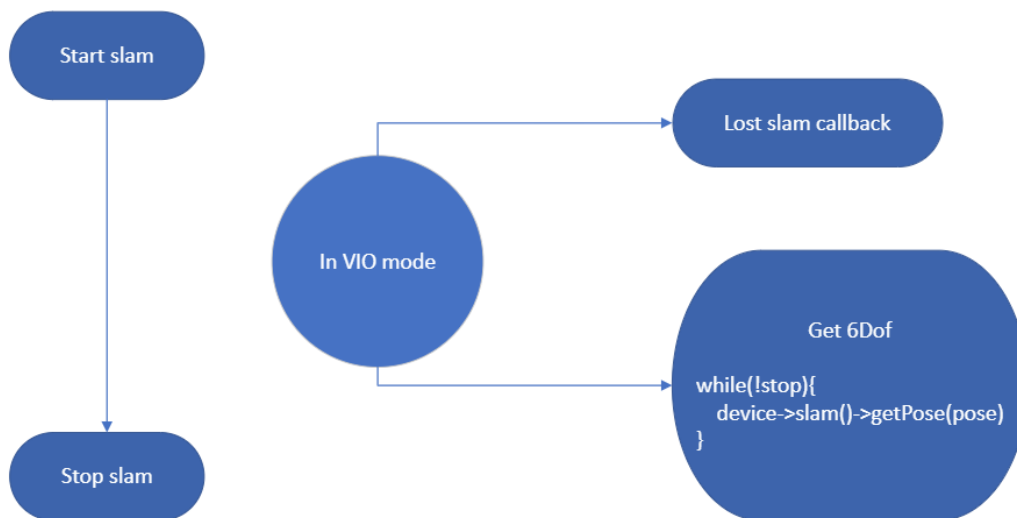


图 3-4 主动获取

### 3.2 VIO 模式

实现 VIO 功能，主要使用下面几个接口：

*bool start();*

*bool stop();*

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

*bool unregisterCallback(int callbackId);*

*bool getPose(Pose &pose, double prediction) ;*

VIO 模式不包含 map loopclosure，随着里程计数增加，累计误差也会增加。正常使用时，一般建议用户使用 CSLAM 模式，先建图，然后运行 SLAM。下面以主动获取 6DOF 为例，介绍如何通过以上接口运行 SLAM，接口调用流程如下所示：

- 1) 注册 lost callback;
- 2) 调用 start(), 开启 SLAM;
- 3) 调用 getPose () 获取 6DOF;
- 4) 调用 stop(), 停止 SLAM。

具体 code 参考 demon-api.cpp (case 2, case3)。

### 3.3 CSLAM 模式

实现 CSLAM 功能，主要使用下面几个接口：

```
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);
```

建图的步骤：

要建立一个好的地图，你必须首先考虑如何使用地图。地图必须包含应用程序所需的视点。如果最终应用程序在另一个房间，则没有理由在当前房间中记录地图。类似地，如果最终应用程序将显示地面上的一些虚拟对象，则没有理由将地图记录到使用摄像头向上看的房间中。应用程序要移动的路径应该是记录地图的最终路径。为了保证一个好的循环闭合，在同一条路径上走两次：例如，从一个起点开始，走开，回到起点，然后再次在同一条路径上走开，然后回到起点结束。在记录过程中，在同一条路径上行走两次，可以保证在循环闭合检测的不同记录视点之间有很好的重叠。在做这个地图记录时，重要的是避免快速移动或面对没有特征的区域。

#### Sample:

具体 code 参考 `demon-api.cpp` (case 6, case7)，下面是 API 调用流程和示意图，按两种使用场景介绍：

#### 场景 1：建图后切换到 CSLAM

API 调用流程：

- 1) 调用 `start()`，注册 6dof callback

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

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

- 2) 开始建图

- 3) 建图结束后，调用 `saveMapAndSwitchToCslam`，保存地图并切换到 `cslam`（如果使用 `callback` 方式获取 6DOF，需要注册 `done_callback`，

localized\_on\_reference\_map)

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

4) 调用 stop(), 停止 cslam

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

流程图如下:

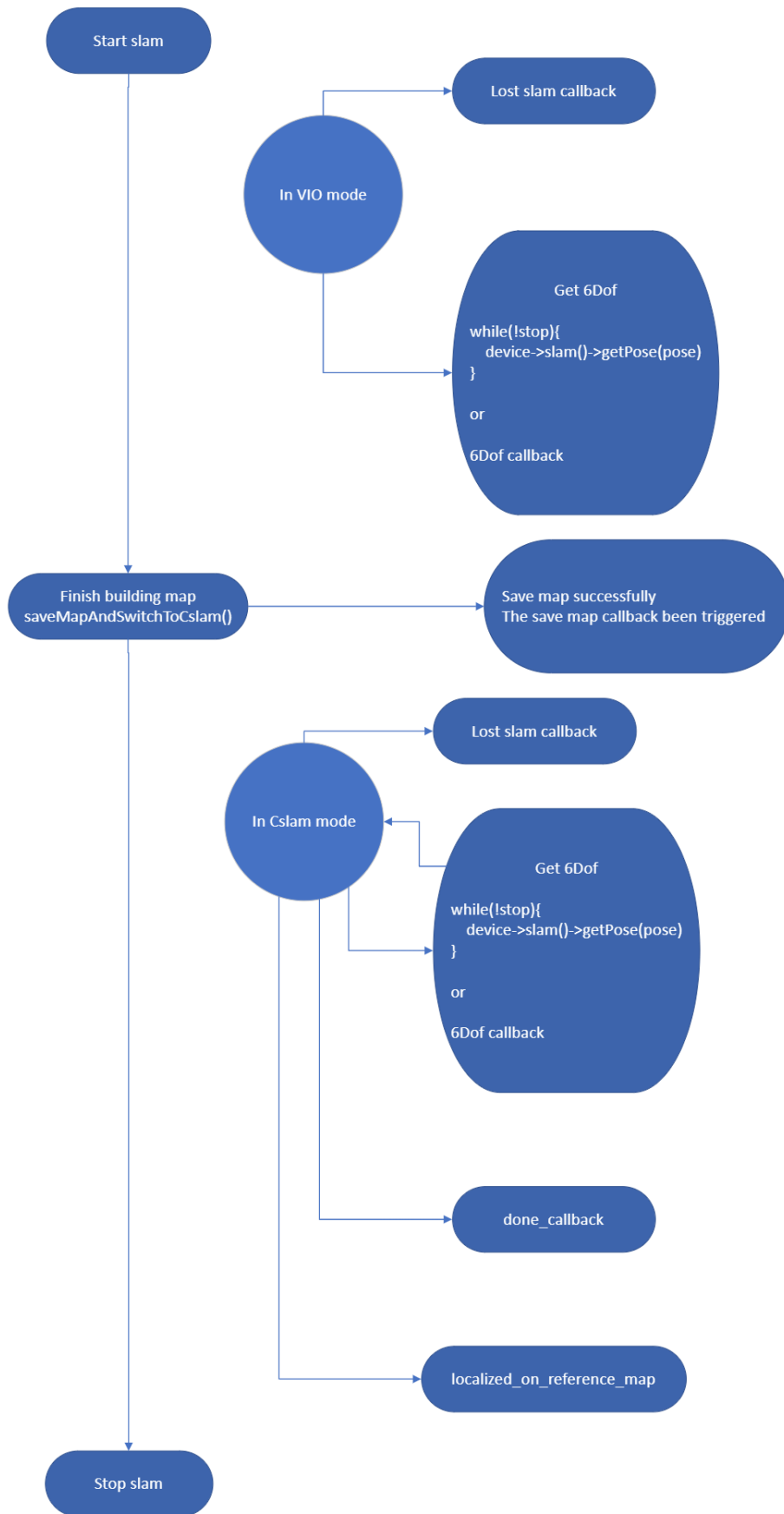


图 3-5 调用流程

## 场景 2: Load 地图后切换到 CSLAM

- 1) 调用 `start()` , 注册 6DOF callback (如果使用 `callback` 方式获取 6DOF,需要注册)

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

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

- 2) 调用 `loadMapAndSwitchToCslam` , 加载地图并切换到 `cslam`(如果使用 `callback` 方式获取 6DOF,需要注册 `done_callback`, `localized_on_reference_map`)

```
device->slam()->loadMapAndSwitchToCslam(mapStream,    cslamSwitchedCallback,  
cslamLocalizedCallback);
```

- 3) 调用 `stop()` , 停止 CSLAM

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

流程图如下:

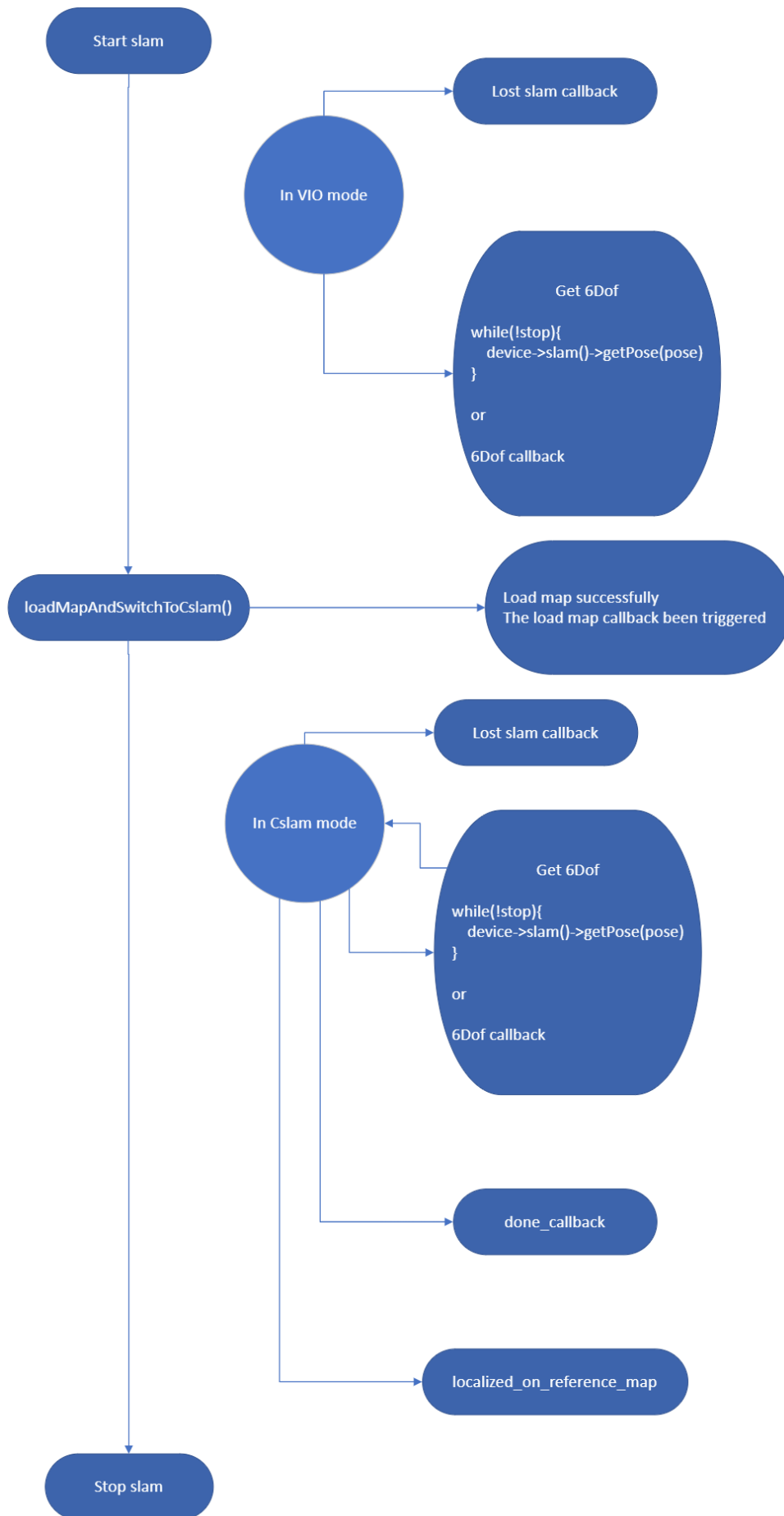


图 3-6 调用流程

### 3.4 3DOF (Rotation)

如果 APP 只需要 3DOF 数据，SDK 可以只获取 imu 数据然后转换为 3DOF，3DOF 结构体以及接口如下：

```
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:

具体 code 参考 demon-api.cpp (case 1, case2)。

### 3.5 平面检测(TOF/stereo)

Plane detection 的接口如下：

```
//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;
};

```

平面检测需要和配合 SLAM 一起工作，平面的位置都是基于 SLAM 坐标系下（SLAM 启动时以 device-imu 所在的位置为原点建立世界坐标系）；

`std::vector<Vector3d> points;` 表示平面的 3D 特征点，依次连接这些 3D 特征点，可以得到真实检测到的平面区域，`int id` 表示平面 ID；具体示意见下图，总共检测到 3 个平面，通过连接 3D 特征点，得到具体平面区域，所有平面都是基于 SLAM 世界坐标系统下。

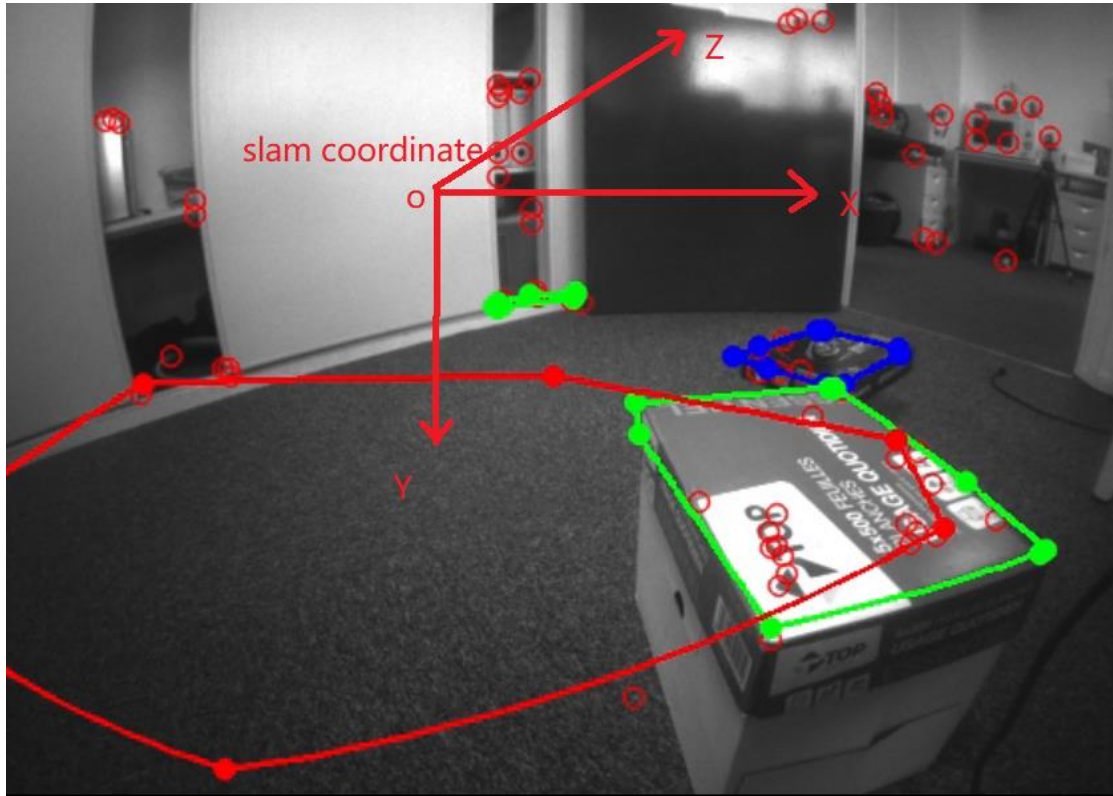


图 3-7 世界坐标系

**Sample:**

TOF 平面检测 `demon-api.cpp` ( case13, case14 ) ; stereo 平面检测 `demon-api.cpp` ( case23, case24 ) 。

### 3.6 地图共享

Map share 的接口如下:

```

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

```

地图共享是 CSLAM 的进阶功能，也是先建图，再切换到带地图的 SLAM 功能，但相对于 CSLAM 区别是这个地图可以共享给其他 device 使用，应用场景是多人能够同时在一个场景下实时交互（如对战游戏，共同办公），如下图所示，其核心是使用者们都基于同一个世界坐标系统下（使用同一个 map）。



图 3-8 地图共享

**Sample:**

建图 `demon-api.cpp` (case19, case20)；使用 `mapdemon-api.cpp` (case21, case22)。

### 3.7 HID 接口调用

Device 很多控制命令都基于 Hid 通道，hid 读写接口如下：

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

**Sample:**

介绍如何调用 hid cmd 接口：

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

### 3.8 标定参数 API

SDK 提供了 fisheye, RGB, TOF, display 的标定参数的 API，所有 camera 的外参坐标系都是以 imu 为原点，右手坐标系；fisheye 内参支持两种类型，Polynomial Distortion Model (rgb, tof) 和 Unified camera model (fisheye)。

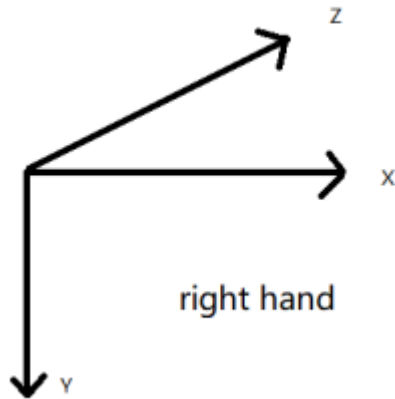


图 3-9 右手坐标系

接口如下:

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

**Sample:**

Fisheye 标定参数 demon-api.cpp (case37); RGB 标定参数 demon-api.cpp (case31); TOF 标定参数 demon-api.cpp (case38);

### 3.9 热插拔处理

Hotplug 的接口如下:

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

### 3.10 获取 Fishseye 数据

Fisheye 频率:

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

Fisheye 的接口如下:

```
virtual int registerCallback(std::function<void (T)>) = 0;  
virtual bool unregisterCallback(int callbackId) = 0;
```

### 3.11 获取 RGB 数据

RGB 的接口如下：

```
virtual int registerCallback(std::function<void (T)>) = 0;  
virtual bool unregisterCallback(int callbackId) = 0;
```

**Sample:**

RGB 数据获取 demon-api.cpp (case9) ；

### 3.12 获取 TOF 数据

TOF 模式介绍：

频率：单频 (SF) | 双频 (DF)

模式：IQ | M2 | edge

帧率：5-30 帧

不同模式对应帧率参考（请注意，由于性能影响，fps 设置项在某些模式下不会根据实际设置模式输出）：

IQ DF: 30FPS

IQ SF: 30FPS

M2 DF: 4.5FPS

M2 SF: 13FPS

M2 DF: 3.5FPS

M2 SF: 7FPS

TOF 的接口如下：

```
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:**

RGB 数据获取 demon-api.cpp ( case11,case39,case40 ) ;

### 3.13 获取 RGBD 数据

RGBD 的接口如下:

```
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:**

注意: 获取 RGBD 数据需同时开启 colorCamera 和 TOFCamera。

RGB 数据获取 demon-api.cpp ( case9 ) ;

### 3.14 CNN 功能介绍

3.14.1 CNN 的接口如下:

```
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:**

CNN 功能 demon-api.cpp ( case42 ) ;

### 3.14.2 Xvisio AI 部署介绍

➤ 依赖项:

OpenVINO Toolkit

<https://docs.openvinotoolkit.org/latest/index.html>

可至 OpenVINO 官方网站下载，并安装好 OpenVINO 以及其依赖的工程环境，本示例安装在/opt/intel/。

➤ 部署流程:

- 1) 获取模型和转换模型，获取.blob 模型文件。

本文档选择 OpenVINO Model Zoo 已经训练好的模型进行下载。

[https://github.com/openvinotoolkit/open\\_model\\_zoo/blob/master/models/intel/index.md](https://github.com/openvinotoolkit/open_model_zoo/blob/master/models/intel/index.md)

使用 OpenVINO Toolkit 自带的模型下载工具进行下载

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

(这里下载好的模型是 OpenVINO IR(.bin & .xml)格式，以 FP16 格式为例，复制 FP16 文件夹到 workspace)

```
cp -r FP16 /home/workspace/example/
cd /home/workspace/example/
```

然后把 OpenVINO IR(.bin & .xml)转换为 Myraid VPU(.blob)，示例如下:

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

(注: 更多命令参数配置可到 OpenVINO 官方网站进行查询。)

- 2) 配置 config.json

config.json 的作用是让 xv sdk 能够正确获取模型的名字以及相应的解析方法，从而获得正确的结果。

对于姿态检测，示例如下:

```
{
  "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/xv sdk-release-
3.1.0/xv sdk/install/bin/CNN_2x8x_r14_5.blob", (该路径为 blob 文件的绝对路径)
"source":"RGB",
"CNN_input_flip_stereo":"true",
"CNN_input_flip_RGB":"false",
"CNN_input_flip_TOF":"false"
}

```

- 3) 配置好 blob 与 json 文件后，将 blob 与 json 文件放置于 demo-api 代码中配置的路径中（默认为 demo-api 目录/data/down/），运行 xv sdk 的 demo-api，选择 42 号设置不同摄像头的 CNN 功能。

## 3.15 手势功能介绍

### 3.15.1 MNN 部署：

手势控制操作前，需将 MNN 模型文件（包含：hand\_landmark.mnn, palm\_detection\_pb.mnn）push 到默认目录/etc/xvisio/gesture/中，如需设置新的路径，可在 push 过模型文件后，通过 setConfigPath 配置新的 MNN 模型文件存储路径。

手势的 21 个特征点分布图：



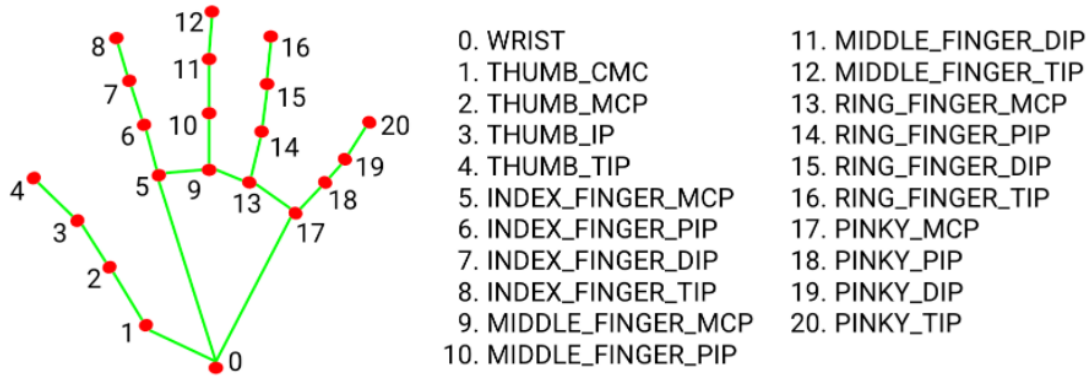


Fig 2. 21 hand landmarks.

### 3.15.2 手势的接口如下：

```

struct keypoint {
    float x = -1;
    float y = -1;
    float z = -1;
};

struct GestureData {
    int index[2] = {-1,-1}; //手势 index 值，分左右手，先左后右
    keypoint position[2]; //手势的坐标数组，分左右手，先左后右。默认为0号特征点位置。
    keypoint slamPosition[2]; //切换 rgb 相机坐标系到 slam 坐标系后的手势坐标数组，分左右手，先左后右。默认为0号特征点位置。
    double hostTimestamp = std::numeric_limits<double>::infinity(); //以 host 端 boot 开始计时（从0开始计时）；
    std::int64_t edgeTimestampUs = std::numeric_limits<std::int64_t>::min(); //以 device boot 开始计时（从0开始计时）；
    float distance; //预留位，用来保存动态手势移动距离。
    float confidence; //预留位，用来保存可信度。
};

virtual void setConfigPath(std::string config) = 0; //设置 MNN 配置文件路径接口。

virtual int registerDynamicGestureCallback(std::function<void (GestureData

```

`const &)> = 0;` //注册动态手势回调，获取动态手势 index 值。

`virtual bool UnregisterDynamicGestureCallback(int callbackID) = 0;` //反注册动态手势回调。

`virtual int registerKeypointsCallback(std::function<void (std::shared_ptr<const std::vector<keypoint>>> callback) = 0;` //注册手势 21Dof 回调，基于 rgb 坐标系，2D 坐标。

`virtual bool unregisterKeypointsCallback(int callbackId) = 0;` //反注册手势 21Dof 回调

`virtual int registerSlamKeypointsCallback(std::function<void (std::shared_ptr<const std::vector<keypoint>>> callback) = 0;` //预留位，用于获取基于 slam 坐标系的手势 21Dof 坐标值，3D 坐标。

`virtual bool unregisterSlamKeypointsCallback(int callbackId) = 0;` //预留位，取消基于 slam 坐标系的回调。

### 3.15.3 示例代码:

代码参考 demo-api 示例代码 case 43-47 测试项。

## 3.16 获取 SGBM 数据

SGBM 的接口如下:

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

`virtual bool unregisterCallback(int callbackId) = 0;`

`virtual Mode mode() const = 0;` //配置 SGBM 模式接口, 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 默认启动配置信息:

`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:**

SGBM 数据获取 demon-api.cpp (case58, 59) ;

### 3.17 Python-wrapper 功能

提供 python 接口，获取 slam, imu, fisheye, rgb, tof, sgbm 等模块的图像及参数数据。

Python-wrapper 部署 (仅支持 windows 版本) :

1. 安装 windows 版本 xv sdk, 勾选 python-wrapper 模块

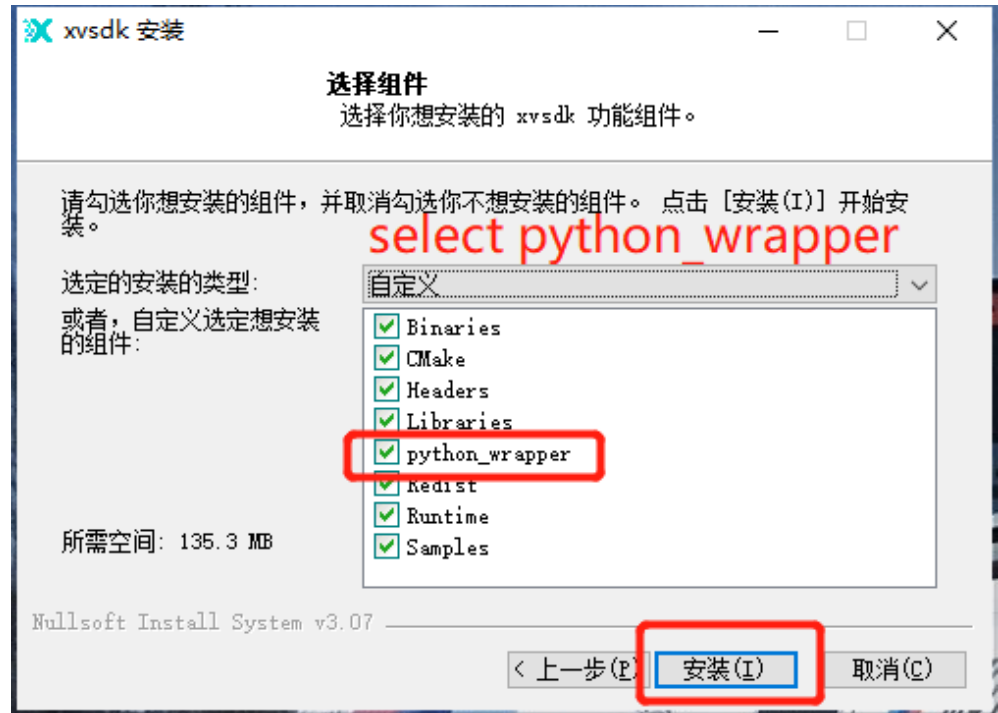


图 3-10 XVSDK

2. 安装路径\bin\python-wrapper 下包含 PythonDemo.py
3. 安装 python3.9，连接眼镜或模组使用 cmd 运行 python PythonDemo.py