

2018-10-30

用户手册

双目惯性模组产品用户使用说明

INDEMIND

目录

1.产品简介.....	4
2.尺寸结构.....	4
3.产品数据.....	5
3.1.产品形态.....	5
3.2.硬件技术指标.....	5
3.3.传感器选型.....	6
3.4.软件技术指标.....	6
3.5.产品规格.....	6
3.6.软件规格.....	7
4.SDK 使用介绍及插件接入规范.....	7
4.1SDK 系统结构.....	7
4.2SDK 使用说明.....	8
4.3 插件调用流程.....	9
4.3.1 插件文件夹结构.....	9
4.3.2 接口类型.....	9
4.3.3 接口调用时序.....	10
4.3.4 插件发布.....	10
4.3.5 开发示例.....	10
4.4 SDK 系统标定数据.....	10
4.5 插件开发规范.....	11
4.6 插件接口介绍.....	11
4.6.1 获取设备信息.....	11
4.6.2 获取标定参数.....	12
4.6.3 获取图像数据.....	13
4.6.4 获取 IMU 数据-原始数据.....	14
4.6.5 获取 IMU 数据-补偿数据.....	14
4.6.6 获取 SLAM 位姿结果.....	15
4.6.7 获取深度解算结果.....	15
4.6.8 保存设备信息和参数.....	16
4.6.9 保存相机标定参数.....	16
5.平台支持.....	16
6.SDK 安装.....	16
6.1Windows SDK 安装.....	16
6.2Linux SDK 安装.....	20
7.依赖项说明.....	23
8.SLAM 说明.....	23
9.DEMO 介绍.....	23
9.1 功能介绍.....	23
9.2 json 介绍.....	24
10.图像、IMU 数据采集软件说明.....	24
10.1 用户软件介绍.....	24
10.2 软件下载与使用.....	24

10.2.1 客户端.....	25
10.2.1 源码.....	25

1. 产品简介

INDEMIND 成立于 2017 年，专注于计算机视觉技术与嵌入式计算平台研发，公司核心技术团队成员均来自计算机视觉领域的顶级技术人员。INDEMIND 双目视觉惯性模组的硬件提供完整的双目+IMU 传感器标定参数，满足不同行业研发人员的开发需求。

INDEMIND 双目视觉惯性模组运用摄像头+IMU 多传感器融合架构，使摄像头与 IMU 传感器优势互补，解决摄像头在遮挡、低纹理环境、光照变化、快速运动等复杂场景中的定位困难问题，实现位姿精度更高、环境适应性更强、动态性能更稳定、成本更低的 SLAM 硬件方案，内置高精度时间同步机制，实现时间同步精度微妙级，为视觉 SLAM 算法提供精确稳定的图像源+IMU 数据，可为视觉 SLAM 研究、智能机器人、AR/VR/MR、无人机避障、室内外导航定位等产品研发或技术研究提供高精度、低成本的研发硬件。

INDEMIND 双目视觉惯性模组采用全局快门的 2*1280*800@50FPS 高清摄像头，可提供水平 120°、垂向 75°视场角，结合高帧率 6 轴 IMU 传感器，为 SLAM 算法提供强有力前端数据采集能力。

结合 INDEMIND 自研的 Vi-SLAM 算法，实现定位稳定性 1-2mm (RMS)、绝对定位精度 < 1%、姿态稳定性 0.1 度 (RMS)、绝对姿态精度小于 1 度等一系列超行业主流水平的定位效果，有效的节约算法开发周期及成本，让开发者可以迅速调试及部署。

2. 尺寸结构

尺寸结构如表 1 所示：

表 1

总体尺寸 (mm)	板子尺寸 (mm)	基线长度 (mm)
140*25	95*25	120

图 1 为模组实拍图：

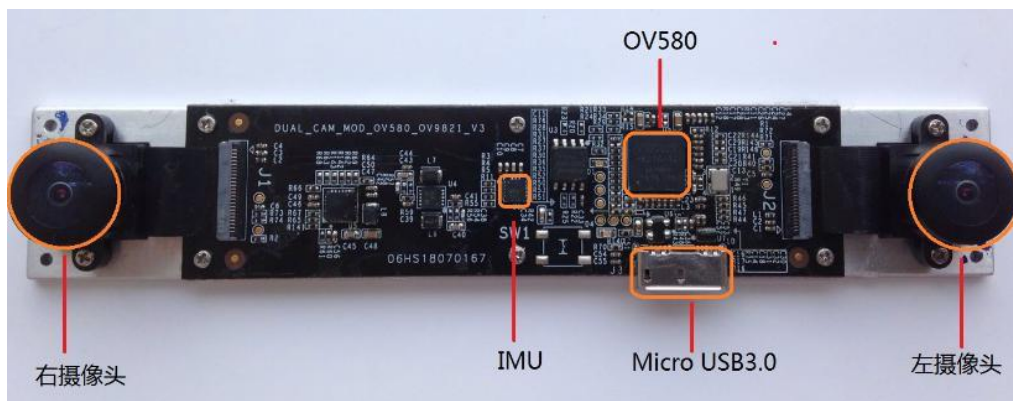


图 1

左、右摄像头：左、右摄像头为传感器镜头，使用中请注意保护，以避免成像质量下降。

IMU：高频率 IMU 硬件，结合双目摄像头，实现精准定位，相机与 IMU 安装精密，切勿拆卸。

OV580：图像、IMU 数据获取芯片。

Micro USB3.0：使用中，插上 Micro USB3.0 数据线后，保持连接处自然受力，不弯折，以避免使用中损坏接口，或导致数据连接不稳定。

IMU Coordinate System (IMU 坐标系统)：IMU 坐标系统为右手系，坐标轴方向如图 2 所示：

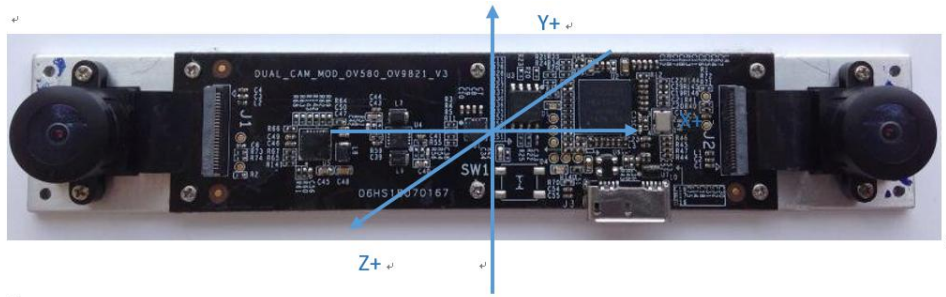


图 2

3. 产品数据

3.1. 产品形态

- 1) 双目+IMU 硬件模组；
- 2) USB3.0 线；
- 3) 标定参数；
- 4) SDK：
 - a) SLAM 算法；
 - b) 深度解算算法。
- 5) 产品使用说明文档；
- 6) 数据采集软件。

3.2. 硬件技术指标

表 2

双目 + IMU 模组	工作距离	0.1m-10m
	摄像头类型	全局快门
	视场角	水平 120°，垂直 75°
	分辨率	1280*800
	帧率	50 FPS
	基线长度	12cm
	曝光控制	自动增益；自动曝光；自动白平衡
	位姿输出频率	1000Hz
	IMU 更新率	1000Hz
	颜色模式	mono

3.3.传感器选型

表 3

器件	型号	参数
摄像头	OV9281	50FPS - 1280*800 分辨率
IMU	ICM20602	1000Hz

3.4.软件技术指标

表 4

SLAM	定位误差	稳定性 1-2mm (RMS)，绝对定位精度<1%；（参考）
	姿态精度	稳定性 0.1° (RMS)，绝对姿态精度<1° (RMS)。（参考）
	视觉频率	25FPS
	IMU 频率	1000Hz
数据融合	视觉+IMU	开启

3.5.产品规格

表 5

型号:	双目视觉惯性模组
尺寸:	板子长宽: 95mm * 25mm; 总体长宽: 140mm*25mm
帧率:	50 FPS
分辨率:	1280 x 800
深度分辨率:	Base on CPU/GPU Up to 640*400@25FPS
像素尺寸:	3.0 x 3.0 μ m (1280*800)
快门速度:	5 millisec
基线:	120.0 mm
镜头:	Replacable Standard M12
视角:	D:140° H:120° V:75°
焦距:	2.09mm
滤镜:	无
运动感知:	6 Axis IMU
色彩模式:	Monochrome (单色)

扫描模式:	Global shutter
功耗:	1.05W @ 5V DC from USB
IMU 频率:	1000Hz
输出数据格式:	Raw data
接口:	USB 3.0
重量:	40g
UVC MODE:	YES

3.6.软件规格

表 6

支持操作系统:	Windows (Windows10) 、Linux (Ubuntu 18.04)
SDK 地址:	https://github.com/indemind
开发者支持:	SDK
标定文件:	IMU 标定、摄像头标定、IMU/摄像头外参标定
整合:	OpenCV/ CUDA
适用的距离:	0.1-10m+

4. SDK 使用介绍及插件接入规范

4.1 SDK 系统结构

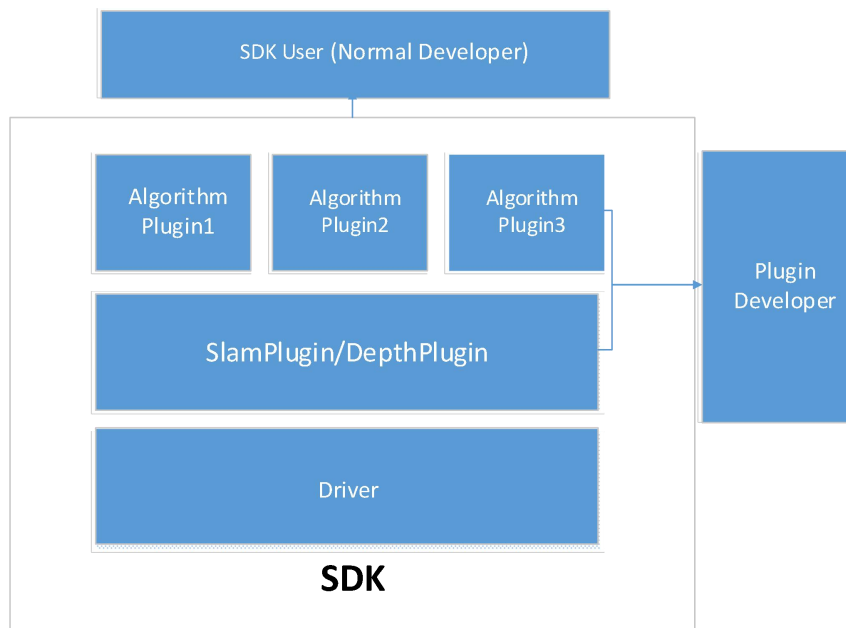
以下用到的名称定义:

- 1) 插件开发者: 开发算法插件的开发人员或小组
- 2) SDK 使用者: 使用 SDK 进行二次开发应用的用户
- 3) SDK 动态库: 即 indem.dll/so

以下为本 SDK 基本结构图及对应的开发者。

对普通 SDK 用户, 无需关心底层硬件和算法实现, 只需要利用 SDK 实现业务功能即可, 本 SDK 提供了 SLAM (采用视觉+IMU 多传感器融合架构, 含前端追踪、后端优化、闭环及重定位等功能) 和深度图解算功能。

对开发者用户, SDK 提供底层驱动的数据、硬件的标定参数、SLAM/深度图解算等信息, 这些信息以接口的形式传递给开发者, 同时开发者可以基于本 SDK 进行算法开发, 并已插件的形式接入 SDK。



流程图 1

4.2 SDK 使用说明

SDK 开发包随本文档一起提供，开发包中分为 linux 和 windows 版本。其中 windows 版本依赖库放在 SDK-bin.zip 中，Linux 版依赖库放在 SDK-lin.zip 中，头文件放在 include 文件夹中，demo 文件夹存放了示例代码。

要使用 SDK，首先需要创建 SDK 对象：

```
CIMRSDK* pSDK = new CIMRSDK();
```

为了获得原始 IMU 及摄像头数据，按照如下方式设置回调函数：

```
void IMUCallback(double time, float accX, float accY, float accZ, float gyrX, float
gyrY, float gyrZ, void* pParam) {}
pSDK->RegistModuleIMUCallback(IMUCallback, param);
void ImageCallback(double time, unsigned char* pLeft, unsigned char* pRight, int
width, int height, int channel, void* pParam) {}
pSDK->RegistModuleIMUCallback(ImageCallback, param);
```

要获取 Slam 解算后的位姿，需要设置如下回调：

```
void ModulePoseCallback(int, void* pData, void* pParam) {
}
//pSDK->RegistModulePoseCallback(HMDPoseCallback, NULL);
```

要获取 depthimage 插件的左目去畸变图像：

```
void DepthImageCallback(int ret, void* pData, void* pParam) {
}
//pSDK->AddPluginCallback("depthimage", "depth", DepthImageCallback, NULL);
```

后续将提供更多插件以增强 SDK 的功能。

4.3 插件调用流程

4.3.1 插件文件夹结构

SDK 系统在启动时，会遍历 `indem.dll` 路径下的 `plugin` 文件夹，加载所有的插件。这些插件以文件夹为组织单位，他们依赖的第三方库都放在这个文件夹下。例如，一个 `depthimage` 插件的组织结构如下：

```
-----plugin
|----depthimage
|----depthimage.dll
|----其他插件文件夹
```

SDK 会使用“文件夹名.dll”或者“文件夹名.so”作为插件的载入入口。

4.3.2 接口类型

插件以 C++ 开发，对插件开发者提供一份接口头文件 `AlgorithmPlugin.h`，其中 `INTERFACE_MAJOR_VERSION` 为当前插件接口的版本号，当接口升级的时候提供的新接口会升级该宏，以兼容旧版本的插件。

接口分为基本的逻辑接口和验证性接口。逻辑接口提供算法调用方面的逻辑操作，验证性接口主要提供插件相关信息及第三方插件开发者的验证性需求等。

逻辑接口及功能简要描述如下：

```
indem::IAlgorithm* AlgorithmFactory();
```

创建算法插件实例。该函数是动态库载入接口，通过该接口创建对应的插件实例，之后 SDK 系统可以通过它进行插件的初始化等操作。

```
virtual const char* Name() = 0;
```

插件名，插件的唯一标识，不能跟其他插件同名。

```
virtual bool Init(CamaraParams pParams) = 0;
```

插件的初始化

```
virtual void AddPoseAsync(double time, const Pose& pose) = 0;
```

SDK 系统会以默认的频率调用该接口，将模组的最终位姿（即融合后结果）传给插件

```
virtual void AddImageAsync(double time, unsigned char* pLeft, unsigned char* pRight, int width, int height, int channel) = 0;
```

SDK 系统会以默认的频率(Hz)调用该接口，模组的图像实时传给插件

```
virtual int AddCallback(const char* name, PluginCallback pCallback, void* pParam) = 0;
```

为插件装载回调函数。该回调函数最终将对 SDK 用户公开，即发布

```
virtual bool InvokeCommand(const char* commandName, _IN_ void* pIn, _OUT_ void* pOut);
```

执行指定的操作命令

```
virtual void Release() = 0;
```

插件资源释放

验证性接口及功能简要描述如下：

```
virtual PluginInfo GetPluginInfo()=0;
```

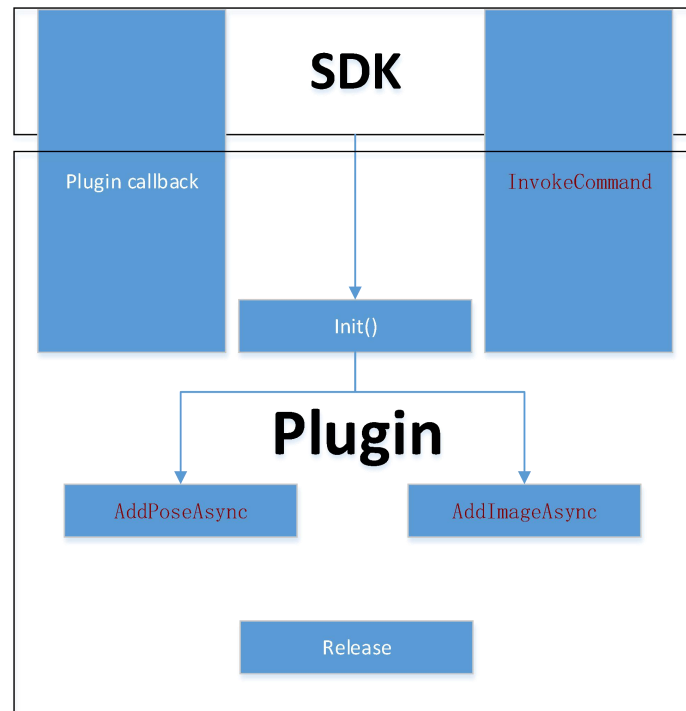
获取插件基本信息，详细参考 DEMO。

4.3.3 接口调用时序

SDK 首先载入动态库，调用 AlgorithmFactory 方法，创建算法插件实例。之后调用插件实例的 Init 方法，将硬件标定参数传递给算法实例。至此，插件初始化完毕。

之后 SDK 会以硬件自身的频率调用 AddPose 函数和 AddImage 函数，将结算后得到的位姿和图像传给插件。

SDK 能够打通插件开发者与 SDK 使用者之间的通道。如下图所示，SDK 通过 AddCallback 向插件添加数据回调函数，插件开发者有能力将算法结果通过回调函数以自定义频率传给 SDK 用户。SDK 用户可以通过 SDK 直接调用 InvokeCommand，执行算法插件特有的操作。



流程图 2

4.3.4 插件发布

插件开发测试完成后，将依赖项及插件动态库拷贝到以插件名命名的文件夹内，将该文件夹拷贝到 plugin 文件夹下即可。SDK 系统会自动进行调用。

另外，为了让 SDK 使用者知道插件提供了哪些功能，插件开发者需要提供：

- 1) 回调函数信息，包括：回调函数名、回调函数的数据类型信息。
- 2) 可用的操作，包括：命令名、输入参数类型、输出参数类型。

4.3.5 开发示例

模组提供 seethrough 插件示例一份，演示了如何向 SDK 添加一个算法，参见提供 DEMO。

4.4 SDK 系统标定数据

插件初始化的时候会将设备标定参数传入进来，以执行必要的操作。其中数据结构如下所示：

```

struct CamaraParams {
    int _width;          //图像宽
    int _height;         //图像高
    int _channel;        //通道数

    double _Kl[9];       //3X3 左相机内参矩阵
    double _Kr[9];       //3X3 右相机内参矩阵
    double _Dl[4];       //4X1 左相机畸变差校正参数
    double _Dr[4];       //4X1 右相机畸变差校正参数
    double _Pl[12];      //3X4 基线校正后左相机投影矩阵
    double _Pr[12];      //3X4 基线校正后右相机投影矩阵
    double _Rl[9];       //3X3 基线校正后左相机旋转矩阵
    double _Rr[9];       //3X3 基线校正后右相机旋转矩阵
    double _TSCl[16];    //4X4 左相机系到传感器坐标系的变换
    double _TSCr[16];    //4X4 右相机系到传感器坐标系的变换
    /* 加计参数, 3X4 矩阵, 每个元素如下
    *   Ax0  11  12  13
    *   Ay0  21  22  23
    *   Az0  31  32  33
    */
    double _Acc[12];
    /* 陀螺参数, 3X4 矩阵, 每个元素如下
    *   Gx0  11  12  13
    *   Gy0  21  22  23
    *   Gz0  31  32  33
    */
    double _Gyr[12];
};

```

4.5 插件开发规范

每次升级, INTERFACE_MAJOR_VERSION 版本号都会加 1。SDK 会根据版本号来确定是否使用新接口。每次升级内容及历史随文档记录同时更新。

以下接口函数应当定义为异步的, 且需要在 1ms 之内返回:

```
virtual void AddPoseAsync(double time, const Pose& pose) = 0;
```

```
virtual void AddImageAsync(double time, unsigned char* pLeft, unsigned char* pRight, int
width, int height, int channel) = 0;
```

4.6 插件接口介绍

4.6.1 获取设备信息

获取插件相关信息部分是由用户通过继承 virtual PluginInfo GetPluginInfo()=0; 重写该函数用于获取插件基本信息。

插件信息获取函数定义如下：

```
ImrModuleDeviceInfo GetModuleInfo();
```

插件基本信息结构定义如下：

```
struct ImrModuleDeviceInfo {
    char _id[32];                //硬件 ID
    char _designer[32];           //开发者
    char _firmware_version[32];  //固件版本
    char _hardware_version[32];  //硬件版本
    char _lens[32];              //摄像头型号
    char _imu[32];               //IMU 型号
    char _viewing_angle[32];     //视场角
    char _baseline[32];          //基线长度
};
```

示例如下：

- 1) 厂家信息；
 - a) INDEMIND
- 2) 固件版本；
 - a) 1.1.1
- 3) 硬件版本；
 - a) 1.1.1
- 4) 镜头型号；
 - a) OV9281
- 5) IMU 型号；
 - a) ICM20602
- 6) 视场角；
 - a) H: 120° , V: 75°
- 7) 基线长度。
 - a) 120mm
- 8) 设备 ID
 - a) 0x730103E8

更详细的说明请参考 demo 调用示例。

4.6.2 获取标定参数

对于这一部分，需要首先继承 IAlgorithmPlugin 基类，然后初始化 Init 接口中，模组的硬件标定参数 CamaraParams 会传入进来，该参数提供了图像/IMU 的标定信息。具体说明请参考 SDK 说明文档或参考 demo 调用示例，其具体的标定参数有：

图像标定信息：

- _width 图像宽
- _height 图像高
- _channel 通道数
- _Kl[9] 3X3 左相机内参矩阵
- _Kr[9] 3X3 右相机内参矩阵
- _Dl[4] 4X1 左相机畸变差校正参数,鱼眼畸变
- _Dr[4] 4X1 右相机畸变差校正参数,鱼眼畸变

- `_Pl[12]` 3X4 基线校正后左相机投影矩阵
- `_Pr[12]` 3X4 基线校正后右相机投影矩阵
- `_Rl[9]` 3X3 基线校正后左相机旋转矩阵
- `_Rr[9]` 3X3 基线校正后右相机旋转矩阵
- `_baseline`（基线），单位：m
- `_TSCI[16]` 4X4 左相机系到传感器坐标系的变换
 - `_RSCI`（前三行三列，旋转矩阵）
 - `_tSCI`（第四行前三列，杆臂，单位：m）
- `_TSCr[16]` 4X4 右相机系到传感器坐标系的变换
 - `_RSCI`（前三行三列，旋转矩阵）
 - `_tSCI`（第四行前三列，杆臂，单位：m）

IMU 标定信息：

- `_Gyro compensation parameter 3X4`（陀螺补偿参数）
 - `_Gyro bias`（第 1 列，陀螺零偏），单位：°/s
 - `_Gyro compensation parameter`（第 2/3/4 列，陀螺比例因子及正交矩阵乘积）
- `_ACC compensation parameter 3X4`（加速度计补偿参数）
 - `_Acc bias`（第 1 列，加速度计零偏），单位：g
 - `_Acc compensation parameters`（第 2/3/4 列，加速度计比例因子及正交矩阵乘积）

参考-加速度计补偿模型（陀螺仪模型等同加速度计模型）：

$$X = Ax0 + Kxx * Ax + Kyx * Ay + Kzx * Az$$

$$Y = Ay0 + Kxy * Ax + Kyy * Ay + Kzy * Az$$

$$Z = Az0 + Kxz * Ax + Kyz * Ay + Kzz * Az$$

式中：X, Y, Z 分别表示加表 xyz 轴输出原始数据；Ax, Ay, Az 分别表示标准输入信息；Ax0, Ay0, Az0 分别是加表 xyz 轴的零偏；Kxx, Kyy, Kzz 分别为加表的比例因子；Kyx, Kzx, Kxy, Kzy, Kxz, Kyz 分别是加表的交叉耦合误差。

代码在 demo 中如下图 code_2 所示。

```
memcpy(params._Dl, g_config->GetHeadDl().data, 4 * sizeof(double));
memcpy(params._Dr, g_config->GetHeadDr().data, 4 * sizeof(double));
memcpy(params._Kl, g_config->GetHeadKl().data, 9 * sizeof(double));
memcpy(params._Kr, g_config->GetHeadKr().data, 9 * sizeof(double));
memcpy(params._Pl, g_config->GetHeadPl().data, 12 * sizeof(double));
memcpy(params._Pr, g_config->GetHeadPr().data, 12 * sizeof(double));
memcpy(params._Rl, g_config->GetHeadRl().data, 9 * sizeof(double));
memcpy(params._Rr, g_config->GetHeadRr().data, 9 * sizeof(double));
memcpy(params._TSCI, g_config->GetLeftCameraTSC().data, 12 * sizeof(double));
memcpy(params._TSCr, g_config->GetRightCameraTSC().data, 12 * sizeof(double));
const cv::Mat& acc = g_config->GetHeadAcc();
params._Acc[0] = acc.at<double>(0, 0); params._Acc[3] = acc.at<double>(0, 1); params._Acc[6] = acc.at<double>(0, 2); params._Acc[9] = acc.at<double>(0, 3);
params._Acc[1] = acc.at<double>(1, 0); params._Acc[4] = acc.at<double>(1, 1); params._Acc[7] = acc.at<double>(1, 2); params._Acc[10] = acc.at<double>(1, 3);
params._Acc[2] = acc.at<double>(2, 0); params._Acc[5] = acc.at<double>(2, 1); params._Acc[8] = acc.at<double>(2, 2); params._Acc[11] = acc.at<double>(2, 3);
const cv::Mat& gyr = g_config->GetHeadGyr();
params._Gyr[0] = gyr.at<double>(0, 0); params._Gyr[3] = gyr.at<double>(0, 1); params._Gyr[6] = gyr.at<double>(0, 2); params._Gyr[9] = gyr.at<double>(0, 3);
params._Gyr[1] = gyr.at<double>(1, 0); params._Gyr[4] = gyr.at<double>(1, 1); params._Gyr[7] = gyr.at<double>(1, 2); params._Gyr[10] = gyr.at<double>(1, 3);
params._Gyr[2] = gyr.at<double>(2, 0); params._Gyr[5] = gyr.at<double>(2, 1); params._Gyr[8] = gyr.at<double>(2, 2); params._Gyr[11] = gyr.at<double>(2, 3);
//在 indemind.dll 目录下有 plugin 文件夹
//plugin 文件夹下存放子文件夹，每个子文件夹存放着插件动态库及其依赖项
//SDK 按照文件夹名字动态加载里头的同名 dll，作为入口
```

图 code_2

4.6.3 获取图像数据

注册图像回调函数，然后调用该回调函数，来获取所需要的数据。具体说明请参考 demo 调用示例，代码在 demo 中如下图 code_3, code_4 所示。

```

/*
    获取相机原始数据回调函数
*/
void GetCameraCallback(double time, unsigned char* pLeft, unsigned char* pRight, int width, int height, int channel, void* pParam) {
    assert(time != NULL);
    cv::Mat Limg(height, width, CV_8UC1, pLeft); //此处用户获取到原始图像数据
    cv::Mat Rimg(height, width, CV_8UC1, pRight);
    //imwrite("D:/indemTest.jpg", img);
}

```

图 code_3

```

//注册模组图像数据获取回调函数
pSDK->RegistModuleCameraCallback(GetCameraCallback, NULL);

```

图 code_4

图像数据为灰度图，时间单位为 ms。

4.6.4 获取 IMU 数据-原始数据

与获取图像数据类似通过注册回调函数来获取 IMU 数据。具体说明请参考 SDK 说明文档或参考 demo 调用示例，代码在 demo 中如下图 code_5 所示。

```

void IMUDataCallbackFunction(imrIMUData* data) {
    IMU temp;
    temp.imu_time = data->timeStamp;
    temp.acc[0] = data->acc[0];
    temp.acc[1] = data->acc[1];
    temp.acc[2] = data->acc[2];
    temp.gyp[0] = data->gyr[0];
    temp.gyp[1] = data->gyr[1];
    temp.gyp[2] = data->gyr[2];
    {
        std::lock_guard<std::mutex> lck(temp);
        temp.current_imgTime = imgTime;
    }

    std::lock_guard<std::mutex> lck(imu);
    Imu_Queue.push(temp);
}

```

```

imrConnectDevice(handlfd, m_idiInfo._devInfo[0]._id, 0, IMUDataCallbackFunction);

```

图 code_5

获取 IMU 原始数据：时间，单位：ms；陀螺 XYZ，陀螺单位：°/s；加速度计 XYZ，加速度计单位：g（g0 取 9.8019967）。

4.6.5 获取 IMU 数据-补偿数据

该部分需要通过 SDK 调用 IMU 回调函数获取补偿后的 IMU 数据，具体说明请参考 SDK 说明文档或参考 demo 调用示例，代码在 demo 中如下图 code_6 所示。

```

void sdkImuCallback(double time, float accX, float accY, float accZ, float gyrX, float gyrY, float gyrZ, void* pParam)
{
    //std::cout << "=====IMU" << std::endl;
    //std::cout << "imu_time" << time << std::endl;
    if (last_imuTime >= time)
    {
        std::cout << "=====IMU" << std::endl;
    }
    last_imuTime = time;
    IMU temp;
    temp.imu_time = time;
    temp.acc[0] = accX;
    temp.acc[1] = accY;
    temp.acc[2] = accZ;
    temp.gyp[0] = gyrX;
    temp.gyp[1] = gyrY;
    temp.gyp[2] = gyrZ;
    std::lock_guard<std::mutex> lck(imu);
    Imu_Queue.push(temp);
}

```

```

pSDK->RegistModuleIMUCallback(sdkImuCallback, NULL);

```

图 code_6

获取 IMU 补偿数据：时间，单位：s；陀螺 XYZ，陀螺单位：°/s；加速度计 XYZ，加速

度计单位：g（g0 取 9.8019967）。

4.6.6 获取 SLAM 位姿结果

注册算法回调函数，SDK 系统会以默认的频率(1kHz)调用该接口，将设备的最终位姿传给插件。具体说明请参考 SDK 说明文档或 demo 调用示例，代码在 demo 中如下图 code_7，code_8 所示。

```
/*
  获取模组算法数据回调函数
  param pData 设备数据，例如ImrHMDPose, ImrTrig
  param pParam 用户定义参数
*/
void GetResultData(int, void* pData, void* pParam)
{
    assert(pParam != NULL);
}
```

图 code_7

```
//注册模组算法数据获取回调函数
pSDK->RegistModulePoseCallback(GetResultData, NULL);
```

图 code_8

SLAM 输出数据：时间，单位：s；位移数据，单位：m；姿态数据：四元数（W,X,Y,Z）

4.6.6.1 SLAM 打开

启用自带的 SLAM，需要将 MRCONFIG 结构体中的 bSlam 设置为 true，这代表开启了 SLAM。若需要调用 SLAM 的结果时，设置 SLAM 的回调函数。

```
config.bSlam = true;
```

```
pSDK->RegistModuleCameraCallback(GetModuleCameraCallback, NULL);
```

图 code_9

4.6.6.2 关闭 SLAM

当运行 SDK，但是不需要 SDK 自带 SLAM 运行时，将 MRCONFIG 结构体中的 bSlam 设置为 false。

4.6.7 获取深度解算结果

该部分需要用户自己继承 IAlgorithmPlugin 类之后，写自己需要的相应算法，然后打包成 lib 文件，通过插件的形式实现调用。具体说明请参考 SDK 说明文档或参考 demo 调用示例。（注意 demo 中将给出深度解算作为参考，用户可以根据需要进行修改）

深度解算算法的打开/关闭同 4.6.6 节中的 SLAM，通过控制回调函数，进行算法的打开及关闭。深度解算结果：深度图，单位：mm

如图 code_11，code_12 所示：

```
pSDK->AddPluginCallback("depthimage", "depth", DepthImageCallback, this);
```

图 code_11

```
void DepthImageCallback(int ret, void* pData, void* pParam) {
    ImrDepthImageTarget* Depth = reinterpret_cast<ImrDepthImageTarget*>(pData);
    std::cout << "-----" << Depth->deeptr[2] << std::endl;
}
```

图 code_12

4.6.8 保存设备信息和参数

在调用获取设备信息参数之后用户可自行完成该部分的功能，对数据进行存储。具体说明请参考 SDK 说明文档或参考 demo 调用示例。

4.6.9 保存相机标定参数

在调用获取标定参数之后用户可自行完成该部分的功能，对数据进行存储。具体说明请参考 SDK 说明文档或参考 demo 调用示例。

5. 平台支持

本 SDK 支持 Windows10 和 Linux（Ubuntu 18.04）平台。

需求：64 位操作系统；USB3.0 接口；运行 INDEMIND SDK 中 Vi-SLAM、深度解算算法，计算平台性能建议>i5 7500 CPU、NVIDIA 1060 显卡。

6. SDK 安装

6.1 Windows SDK 安装

该部分，我们为用户提供了所需要的相关库文件，用户只需要下载之后包含相关的库文件等即可使用。在使用 Visual Studio 进行开发的过程中，我们推荐使用 VS2015 及其以上版本，2015 以下版本未经过测试。如有任何疑问，欢迎大家上 github 提出问题。

安装必须硬件和软件

需要显卡 Geforce GTX 1050 以上，并且把显卡驱动安装好，CUDA 版本 9.0。没有 GPU，如深度解算等功能无法正常运行。

具体如下：

1) 程序下载

首先在指定网站 <https://github.com/indemind/SDK-Win64> 下载 SDK，如图 SDK_1，Linux 版本下载地址：<https://github.com/indemind/SDK-Linux>。

bin	2018/11/5 14:22	文件夹	
demo	2018/10/13 16:57	文件夹	
doc	2018/11/5 14:23	文件夹	
include	2018/11/5 9:57	文件夹	
indem.lib	2018/10/31 18:05	对象文件库	18 KB
README.md	2018/11/2 18:05	MD 文件	1 KB

图 SDK_1

SDK 主要包含：demo，doc，include、bin 四个文件夹和 indem.lib 程序库。

demo 文件夹下放置的是 DEMO 程序；

doc 文件夹下放置的是说明文档和模组的相关测试报告；

include 文件夹下面放置的是需要用到的头文件；

bin 文件夹下放置的是所依赖的各部分的 dll 文件以及配置所用的 json、yaml 文件；

indem.lib 文件是 SDK 开发所依赖的 lib 程序库。

如果要在自己的工程中添加 SDK，需要保证自己的项目是 x64 平台。否则需要对自己项目进行升级才能使用。

2) 工程创建

如果是新建工程，可以参考如下步骤：

打开 VS2015（我们在这里较为推荐使用 VS 的该版本），如下图 SDK_2。



图 SDK_2

在左侧导航栏中点击创建新项目，并指定相应的文件夹和路径，如图 SDK_3。

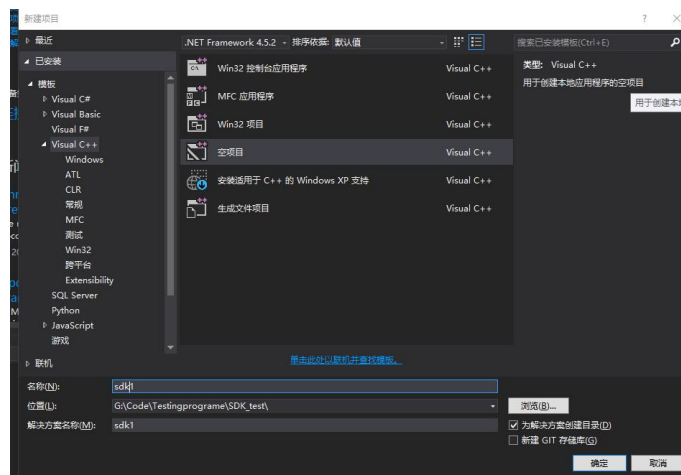


图 SDK_3

点击确定，这样我们就生成了一个空的工程。因为我们要使用刚刚下载下来的安装包中提供的 lib 文件，所以在这里我们需要给工程添加相应的头文件和 lib 文件。因为提供的程序均为 x64 平台下执行代码，所以，不管在 Debug，还是 Release 配置下，平台均需要选择 X64。鼠标右击该工程名字，查看项目属性，如图 SDK_4(以下均以 Release x64 环境为例)。

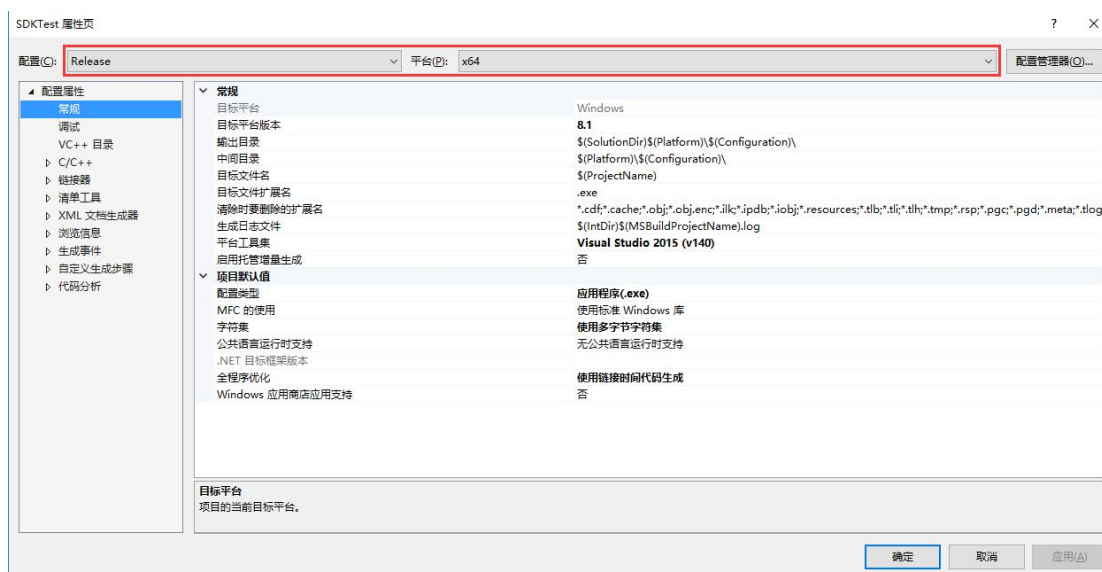


图 SDK 4

在”VC++目录 -> 包含目录”中包含刚刚下载的 SDK-Win64 文件夹中的 include 文件夹路径,以及在”VC++目录 -> 库目录”中包含 indem.lib 程序库所在的路径,如图 SDK_5,SDK_6 所示。

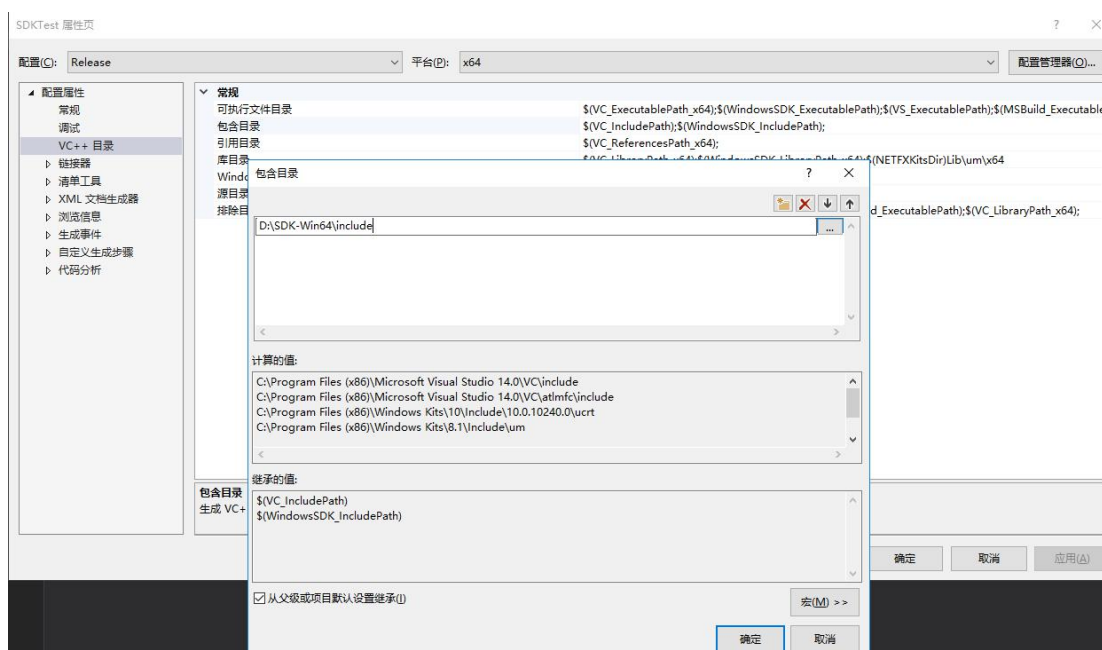


图 SDK_5

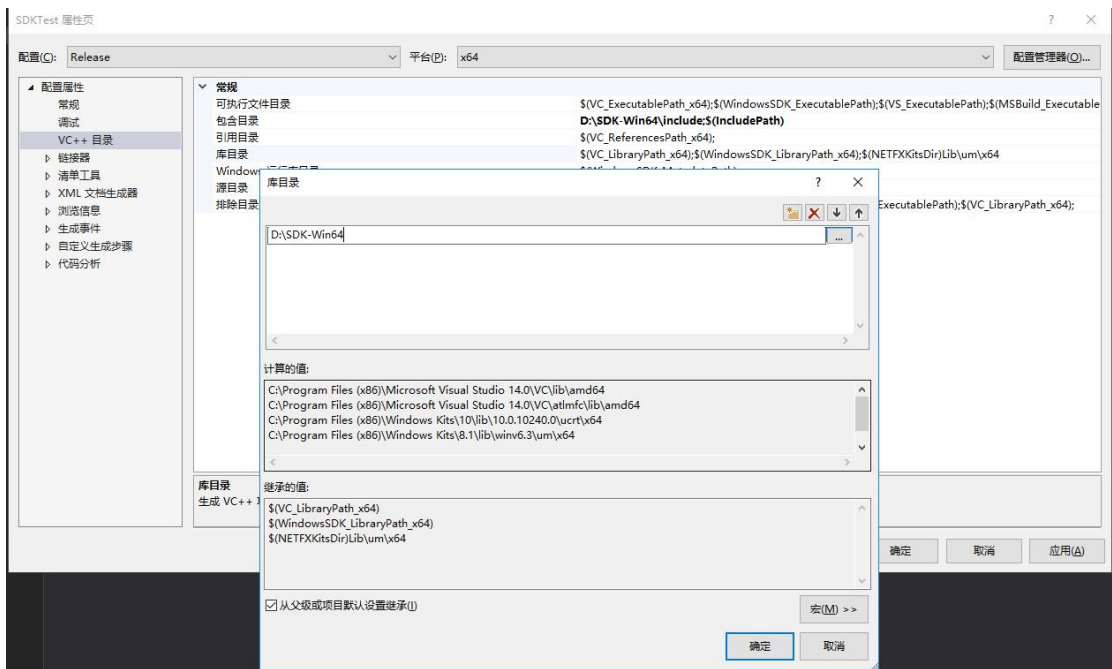


图 SDK_6

然后点击链接器—输入—附加依赖项，将具体的.lib 文件添加到此处，如图 SDK_7 所示。

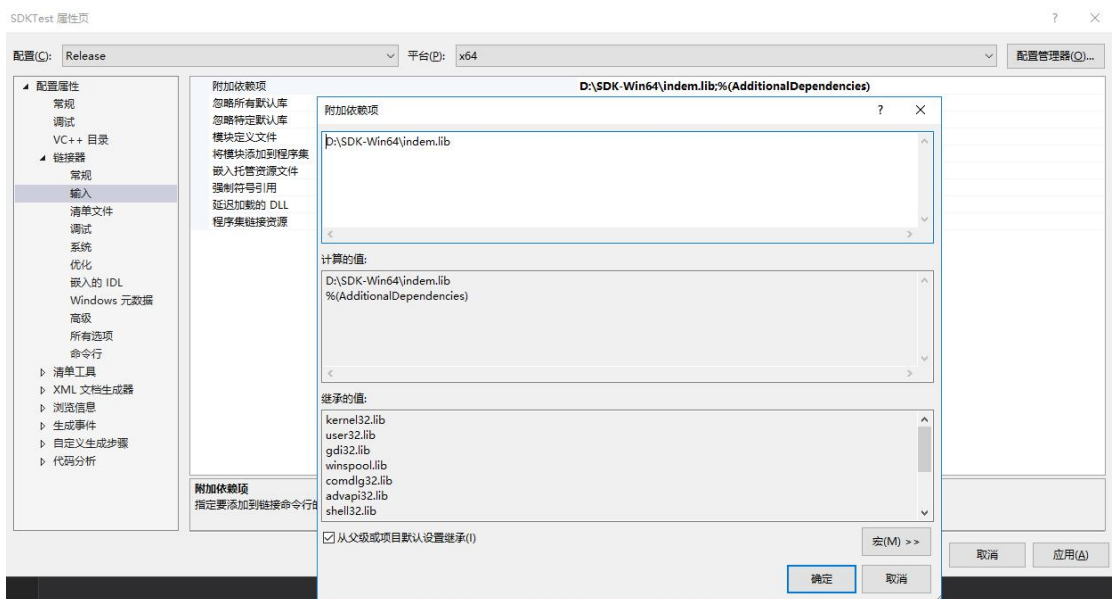


图 SDK_7

最后在程序进行重新生成前的最后一步修改，将编辑器上方的解决方案配置及平台修改为配置时设定的平台，如图 SDK_8



图 SDK_8

这样一个基本环境就搭建成功了，我们可以根据需要来使用其中的方法，重写其中的方法，或者是按照需要来调用其方法了。这里需要注意的是，调用相应的方法时，需要包含有该方法定义的头文件，我们在 SDK 文件夹下的 include 文件夹中给出了我们提供的方法定义的头文件，如图 SDK_9 所示。



名称	修改日期	类型	大小
bin	2018/11/5 16:29	文件夹	
demo	2018/11/5 14:54	文件夹	
doc	2018/11/5 14:54	文件夹	
include	2018/11/5 14:54	文件夹	
indem.lib	2018/10/31 18:05	对象文件库	18 KB
README.md	2018/11/2 18:05	MD 文件	1 KB

图 SDK_9

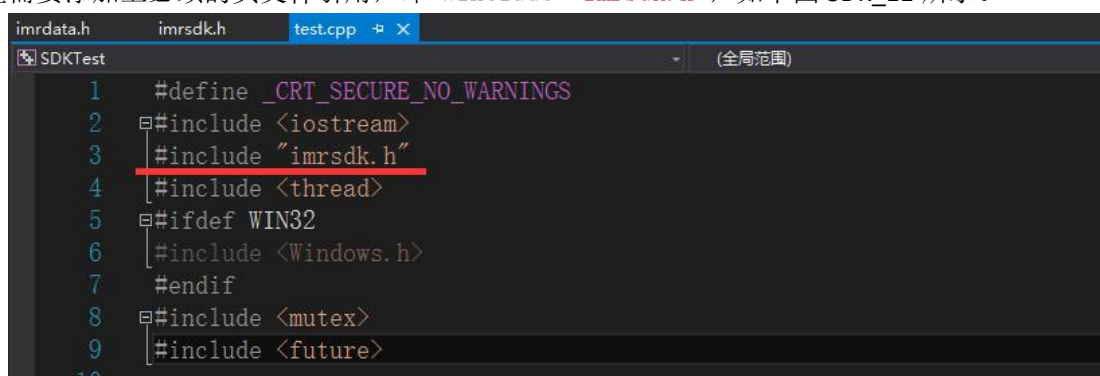
在创建工程的时候,可以选择将相应需要的头文件即.h 文件放置到所新建的工程目录下,如图 SDK_10 所示。



名称	修改日期	类型	大小
Debug	2018/11/5 16:00	文件夹	
x64	2018/11/5 16:02	文件夹	
imrdata.h	2018/11/5 9:57	H 文件	3 KB
imrsdk.h	2018/11/1 14:25	H 文件	7 KB
SDKTest.vcxproj	2018/11/5 16:00	VC++ Project	6 KB
SDKTest.vcxproj.filters	2018/11/5 16:00	VC++ Project Fil...	1 KB
test.cpp	2018/11/5 16:26	C++ 文件	4 KB

图 SDK_10

之后创建自己的程序文件即.cpp 文件,然后在该.cpp 文件中就可以开始我们的编码了,这里需要添加上必须的头文件引用,即 `#include "imrsdk.h"`,如下图 SDK_11 所示。



```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <iostream>
3  #include "imrsdk.h"
4  #include <thread>
5  #ifdef WIN32
6  #include <Windows.h>
7  #endif
8  #include <mutex>
9  #include <future>
10

```

图 SDK_11

然后创建我们自己的 main 函数,去初始化相应的参数就可以了,具体的初始化方法,如果你是初学者可以参照 Demo 中给的示例来进行理解与开发,如果你是有经验的开发者,可以查找 imrsdk.h 文件进行所需要函数定义的查询,来进行下一步开发。

运行时将 bin 文件夹里的内容拷贝到所在项目路径下即可。模组设备调试时需要使用 USB3.0 接口。

6.2Linux SDK 安装

1) 安装必须硬件和软件

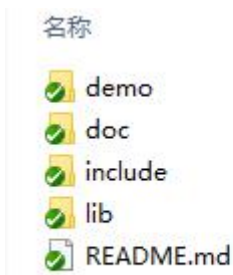
需要显卡 Geforce GTX 1050 以上,并且把显卡驱动安装好,CUDA 版本 9.0。没有 GPU,如深度解算等功能无法正常运行。

2) 下载 Linux 版本 SDK

首先在指定网站下载 Linux 版本 SDK,地址: <https://github.com/indemind/SDK-Linux>。

3) 解压 SDK

下载完解压出以下文件：



SDK 主要包含：demo，doc，include、lib 四个文件夹。

demo 文件夹下放置的是 DEMO 程序；

doc 文件夹下放置的是说明文档；

include 文件夹下面放置的是需要用到的头文件；

lib 包括 **Ubuntu 1604** 和 **Ubuntu 1804** 系统库文件，库文件夹下放置的是所依赖的各部分的链接文件.so 与.a 文件以及配置所用的.yaml 与.json 文件；

4) 编译 demo

编译前的准备

a) 安装 cmake

```
sudo apt-get install cmake
```

b) 安装 google-glog + gflags

```
sudo apt-get install libgoogle-glog-dev
```

c) 安装 BLAS & LAPACK

```
sudo apt-get install libatlas-base-dev
```

d) 安装 SuiteSparse and CXSparse

```
sudo apt-get install libsuitesparse-dev
```

e) 编译器

使用 **Ubuntu 16.04** 编译 demo 程序需要使用 **GCC5.4** 版本，否则可能链接失败。

使用 **Ubuntu 18.04** 编译 demo 程序需要使用 **GCC7.3** 版本，否则可能链接失败。

编译 demo

解压后的文件在 Linux 上显示以下：

```
root@d12:/home/sdk# ls
demo doc include lib
```

现在在 **Ubuntu 18.04** 上使用 **GCC7.3** 编译 demo 为例，操作如下：

通过命令行进入 demo 目录里面使用命令 `mkdir build` 创建一个 build 目录，用来放编译 demo 程序。

```
CMaKeLists.txt Indem.cpp
root@d12:/home/sdk/demo# mkdir build
root@d12:/home/sdk/demo# ls
build CMaKeLists.txt Indem.cpp
```

进入 build 目录

```
cd build
```

```
cmake ..
```

```
make
```

编译 demo，操作截图以下

```

root@dl2:/home/sdk/demo# cd build/
root@dl2:/home/sdk/demo/build# ls
root@dl2:/home/sdk/demo/build# cmake ..
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sdk/demo/build
root@dl2:/home/sdk/demo/build# make

```

如果编译成功的话，会生成一个 TestIndem 可执行文件。

5) 执行 demo 程序

把刚才编译的可执行文件 TestIndem 拷贝到刚才解压 SDK 的 lib 目录下的 1804 目录下，在 lib/1804 目录下使用 ./TestIndem.sh 命令启动程序。

TestIndem 和 TestIndem.sh 需要可执行权限。使用命令 chmod 777 TestIndem 和 chmod 777 TestIndem.sh 进行修改。

为了提高系统稳定性，请运行时使用超级用户（root 权限）运行，或者使用“sudo ./程序名”运行，例如 DEMO 运行“sudo ./TestIndem.sh”。

如果用户需要自己写 demo 程序，请参照此 demo，把 CMakeLists.txt 写好就可以进行编译。

如果用户自定义生成 demo 的可执行文件的名称，还需要在 TestIndem.sh 里用其替换掉 TestIndem。

7) 调用 SDK 的接口说明

a) 创建 SDK 对象

```
CIMRSdk* pSDK = new CIMRSdk();
```

b) 设置使用的 SLAM

```
MRCONFIG config = { 0 };
```

```
config.bSlam = true; //true 开启 SLAM,false 不开启 SLAM
```

c) 获取模组图像数据

```
pSDK->RegistModuleCameraCallback(SdkCameraCallBack,NULL);
```

d) 获取 IMU 数据

```
pSDK->RegistModuleIMUCallback(sdkImuCallBack,NULL);
```

e) 获取 SLAM 结果

```
pSDK->RegistModulePoseCallback(sdkSLAMResult,NULL);
```

f) 获取深度图

```
pSDK->AddPluginCallback("depthimage", "depth", DepthImageCallback, NULL);
```

g) 释放资源

```
pSDK->Release();
```

```
delete pSDK;
```

注意：不要在以上回调函数里做延时比较多的操作，比如把数据写入文件等，否则会造成

数据丢失，影响数据正确性等后果。

8) 显示深度图可以使用 **opencv** 来处理，参考如下代码，如下：

在 Indem.cpp 里增加 `#include <opencv2/opencv.hpp>`

```
void DepthImageCallback(int ret, void* pData, void* pParam) {
    ImrDepthImageTarget* Depth = reinterpret_cast<ImrDepthImageTarget*>(pData);
    //std::cout << "DepthImageCallback==" << std::setprecision(10) << Depth->_image_w << ", _image_h" << Depth->_image_h << std::endl;
    float maxp = 0;
    for (int mi = 0; mi < Depth->_image_w * Depth->_image_h; mi++)
    {
        if (Depth->_deepptr[mi] > maxp)
            maxp = Depth->_deepptr[mi];
    }
    cv::Mat dep(Depth->_image_h, Depth->_image_w, CV_8UC1);
    for (int mv = 0; mv < Depth->_image_h; mv++)
    {
        for (int mu = 0; mu < Depth->_image_w; mu++)
        {
            dep.at<uchar>(mv, mu) = (uchar)fmax(Depth->_deepptr[mv*Depth->_image_w + mu] * 255.f / maxp, 0.f);
        }
    }
    cv::imshow("depthimage.png", dep);
    cv::waitKey(1);
}
```

7. 依赖项说明

设备提供的 SLAM 需要的共依赖的库有 g2o、ceres、lapack、glog、OpenCV、CUDA。

g2o 和 ceres 都是用来 SLAM 优化的依赖项，若是没有该部分将无法正常运行算法优化。

OpenCV（Windows SDK 中 OpenCV 版本为 3.1.0；Linux SDK 中 OpenCV 版本为 3.4.3）是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。

Lapack 用来支持求解科学与工程计算中最常见的数值线性代数问题，如求解线性方程组、线性最小二乘问题、特征值问题和奇异值问题等的依赖项。

Glog 是一个 C++ 语言的应用级日志记录框架，提供了 C++ 风格的流操作和各种助手宏，主要是用于支持日志相关操作的依赖项。

CUDA 是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题，SDK 中深度图的解算依赖 GPU 加速，CUDA 版本为 9.0。

8. SLAM 说明

SLAM (Simultaneous Localization and Mapping), 即时定位与地图构建。SDK 提供的 Vi-SLAM 包含前端特征提取，匹配，后端优化，闭环、建图和重定位功能。

Vi-SLAM 支持 Windows 及 Linux 平台，有效的节约算法开发周期及成本，让开发者可以迅速调试及部署，直接应用于机器人、无人机、AGV、AR/VR 等领域。

Vi-SLAM 工作过程中的特征点匹配显示状态控制在文件 “slam_imp.yaml” 中，displayImages: true 表示打开特征点匹配显示，displayImages: false 表示关闭特征点匹配显示，该状态默认开启。

9. DEMO 介绍

9.1 功能介绍

在给出的 demo 示例小程序中有具体调用示范，用户可根据该程序的示范，修改用户想要的程序。Demo 中主要提供的实例程序功能分别为：

a) 驱动获取 IMU 原始数据，数据格式：时间单位：ms；陀螺单位：°/s；加速度计单

- 位: g (g_0 取 9.8019967); 当前数据输出格式为: 时间戳、陀螺数据、加速度计数据、当前图像时标;
- b) 驱动获取图像数据, 数据格式: 时间单位: ms
 - c) SDK 获取 IMU 补偿数据, 数据格式: 时间单位: s; 陀螺单位: $^{\circ}/s$; 加速度计单位: g (g_0 取 9.8019967)
 - d) SDK 获取图像数据: 时间单位: s
 - e) SDK 获取 SLAM 结算结果, 数据格式: 时间单位: s; 位移数据: m; 姿态数据: 四元数 (W,X,Y,Z)
 - f) SDK 获取深度解算结果, 数据格式: 深度图, 单位: mm

9.2 json 介绍

为进一步方便用户基于 INDEMIND SDK 进行二次开发, SDK 开放 SLAM、深度解算等算法的参数配置文件, 下面对参数配置文件 (json 文件) 进行介绍 (部分说明), 如图 json_1, json 文件的路径为: SDK\SDK-win

```

{
  "running": {
    "log": 0, //关闭log文件
    "type": 1, //设备类型
    "algorithm": {
      "Detection": {
        "open": false, //深度图结算开启标志
        "params": {
          "freq": 200, //频率 单位ms
          "hasmap": true, //深度图初始化配置是否有地图
          "saveoctomap": false, //深度图初始化配置是否存地图
          "useoctomap": true //深度图初始化配置是否使用地图
        }
      }
    }
  }
}

```

图 json_1

10. 图像、IMU 数据采集软件说明

10.1 用户软件介绍

模组数据采集软件采集图像及 IMU 数据, 并保存为 EuRoC 数据集格式, 方便用户进行离线测试使用。

具体使用方法请参考模组数据采集软件使用手册, 用户手册中会有详细说明。

10.2 软件下载与使用

根据用户群的不同, 提供了对应的 WINDOWS 版本 (建议使用 WIN10 及以上版本), 以及 LINUX 版本 (包括: 16.04、18.04) 的客户端。

另外, 提供了数据采集程序的源码, 方便用户在不同条件下, 根据所处环境选择或添加新功能提供支持。

10.2.1 客户端

WINDOWS-64

https://github.com/INDEMIND/ModuleInfo_Win64

LINUX (16.04、18.04)

https://github.com/INDEMIND/ModuleInfo_Linux

10.2.1 源码

WINDOWS-64 / LINUX (16.04、18.04)

https://github.com/INDEMIND/ModuleInfo_Source