

Optor Visual-Inertial Camera

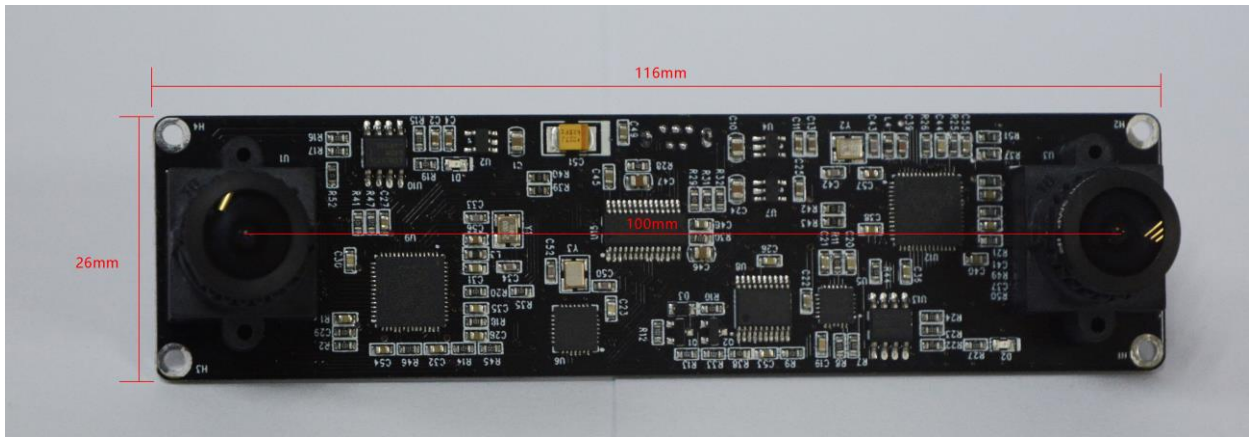
使 用 手 册

v 0.1

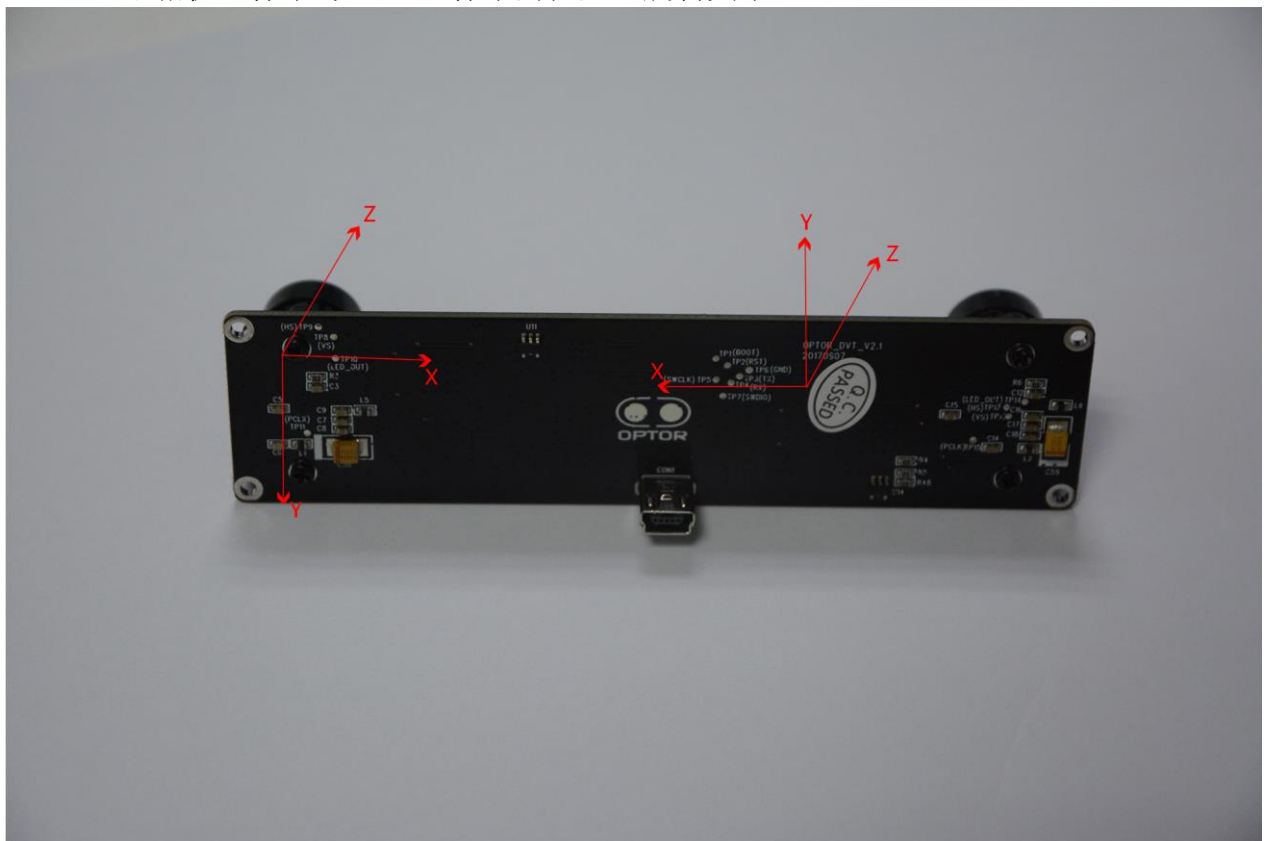
optor-惯性相机是一款专门为视觉算法开发者设计的通用视觉传感器。具有丰富的硬件控制接口以及数据接口，为科研人员提供可靠的图像数据以及惯导数据，降低开发门槛。

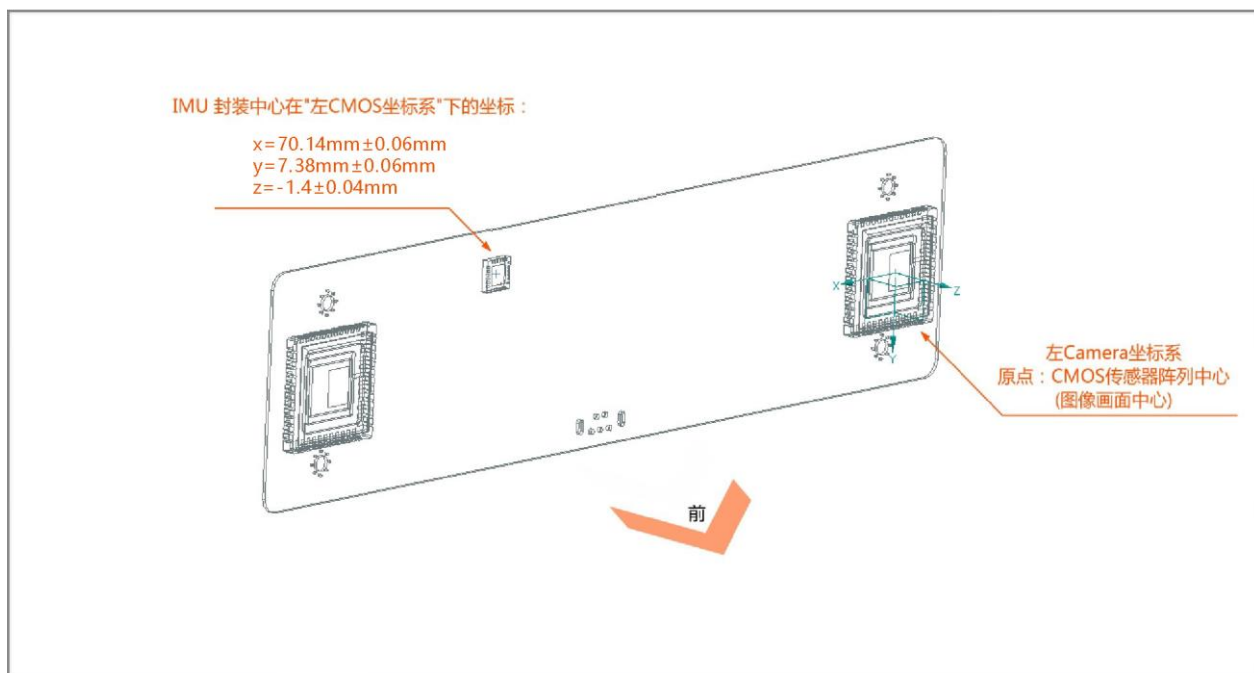
1. 相机硬件规格

1.1 物理尺寸

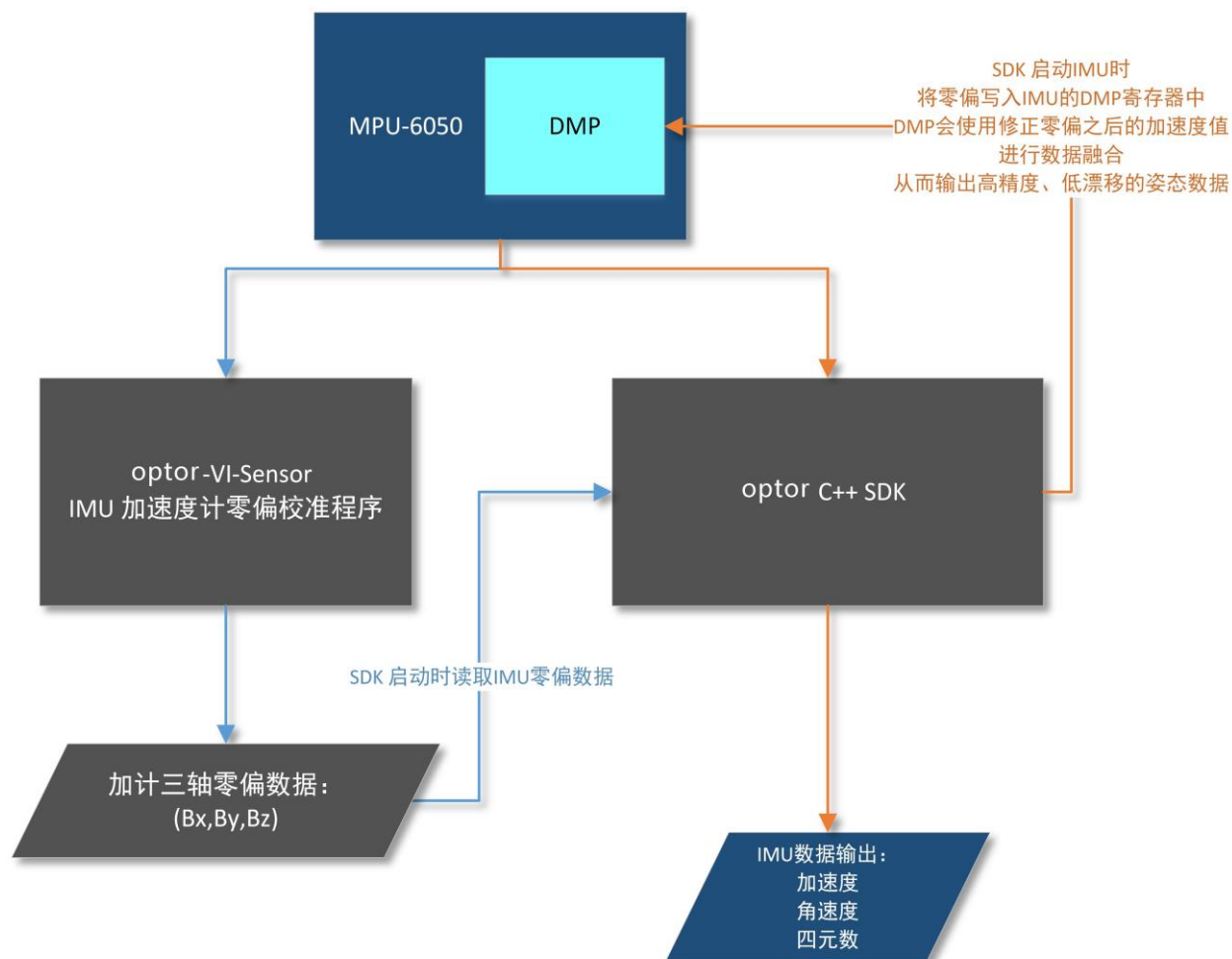


1.2 左眼相机坐标系与 IMU 坐标系的位置、旋转关系





1.3 硬件性能&规格

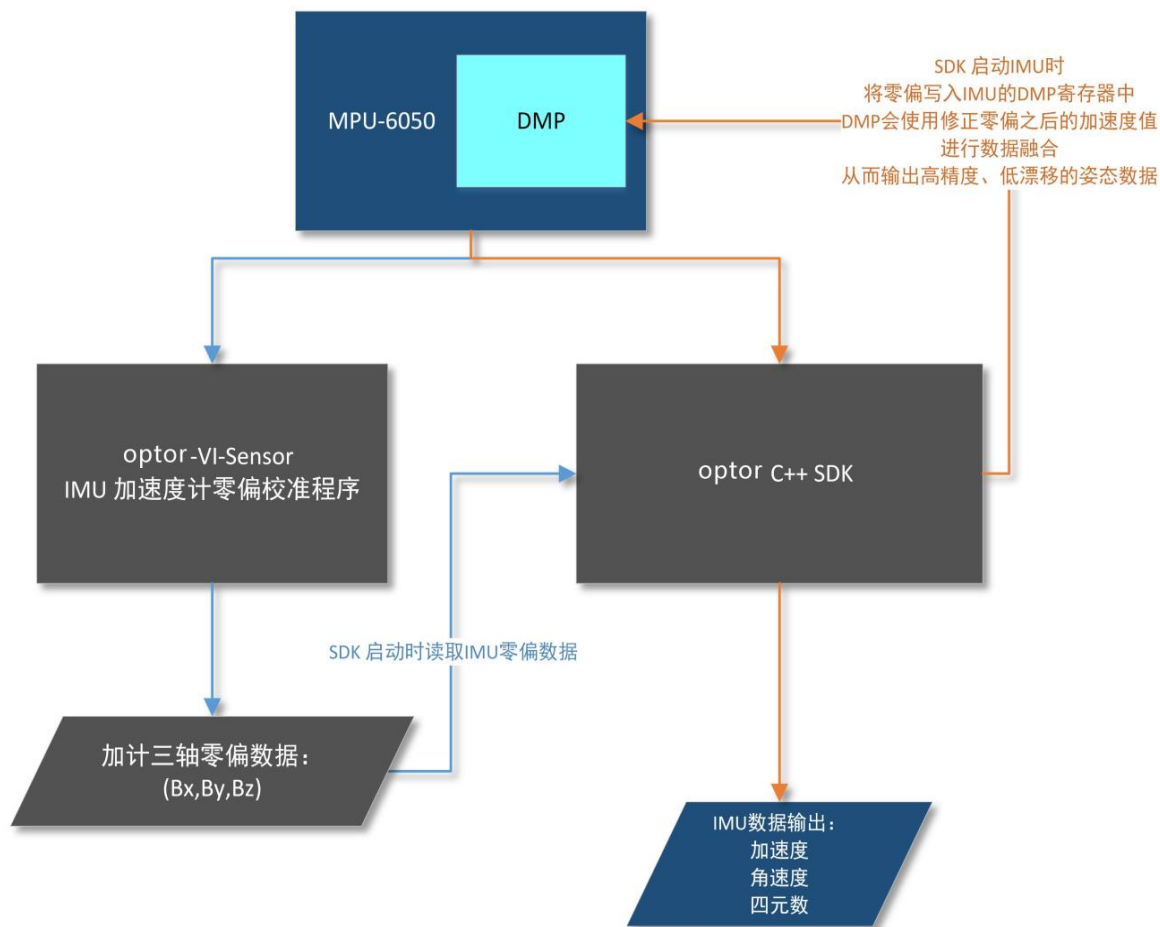


2. 产品特点

Loitor 惯性相机 - 硬件规格

	CMOS	IMU
型号	MT9V034	MPU-6050
曝光方式	全局快门	-
控制器IC	CY68013	STM-32
帧率	24-65 fps	200 fps
图像分辨率	320x240 640x480 752x480	-
硬件固件更新方式	可通过Windows端固件更新程序进行相机固件升级	
双目基线长度	10cm	-
镜头接口规格	M12 镜头接口	-
镜头规格	2.1mm/150度广角 + 6mm/60度窄视角（@752x480分辨率）	-
数据接口	USB 2.0	
数据传输延迟	1000ms/当前帧率	100us
帧同步方式	SDK软件自动控制触发左右眼帧同步，精确至40微秒	微秒级时间戳

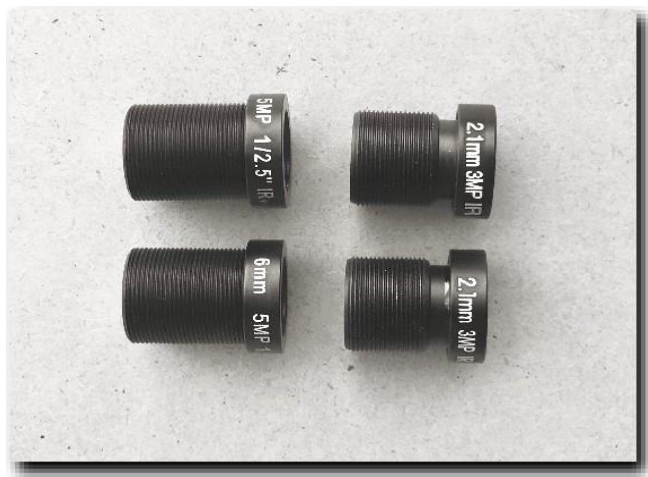
2.1 包含 IMU 零偏标定程序，使用零偏值初始化 DMP 算法，使 IMU 输出数据具有较高的精度以及比正常情况更小的姿态漂移



2.2. 双目光学参数出厂时已经精确标定



2.3. 镜头座内螺纹经过特殊处理，保障镜头在长途运输 过程里不会松动、引起相机内参变化；同时能够更换镜头。



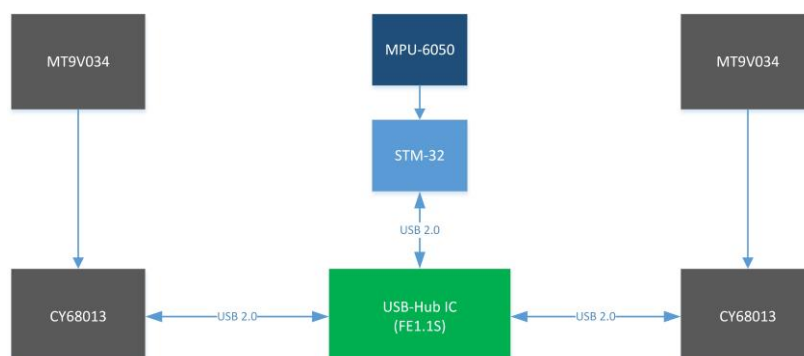
2.4. SDK 无需编译，没有特殊依赖库（仅依赖 libusb）

2.5. 具有稳定可靠的 ROS 驱动

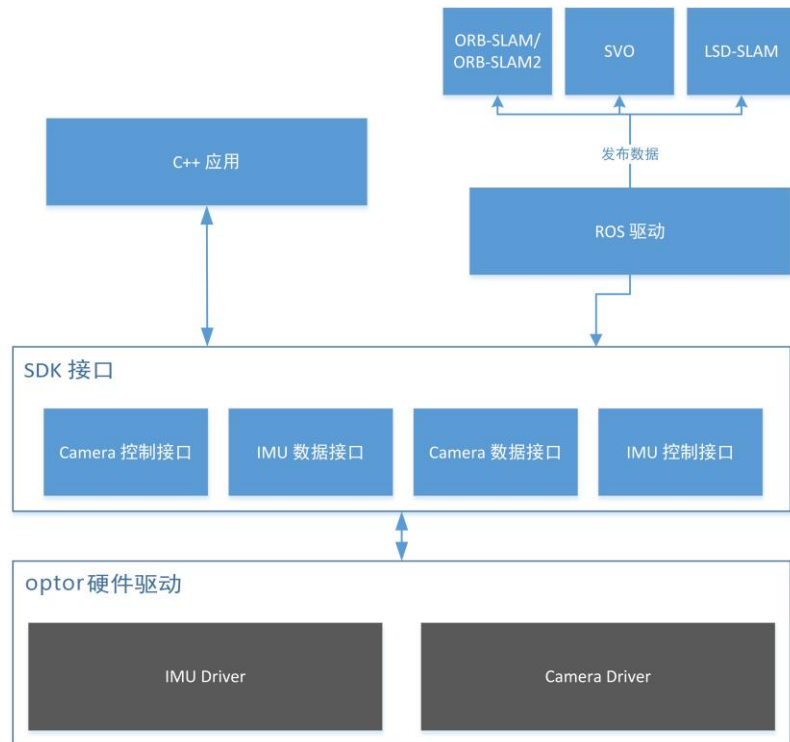
2.6 支持 Ubuntu16 以及 14

3. 软硬件架构

3.1 硬件架构



3.2 SDK 架构



4. SDK 的目录结构、Demo 程序的编译步骤

4.1 SDK 目录结构

```

yjzb@yjzb-desktop:~/yzx/optor/optor_VI_Sensor_SDK_V1.0 14:51:45
$ tree
.
├── imucaldata.txt
├── IMU_Calib
│   ├── imucal -----> IMU 标定程序
│   └── imucaldata.txt -----> IMU 标定结果
├── loitor_PCCam_cliper.stl
├── mac_calib_pattern.png
├── optor_PCCam_cliper.stl
├── optor-vi-install.sh
├── optor_vi-install.sh
├── pattern.pdf
├── ROS -----> ROS Package 目录
│   └── optor_stereo_visensor -----> 用于复制到/catkin_ws/src 下
│       ├── CMakeLists.txt
│       ├── include
│       │   └── optor_stereo_visensor_ros
│       │       ├── loitorcam.h
│       │       ├── loitorimu.h
│       │       ├── loitorusb.h
│       │       └── stereo_visensor_cam.h
│       ├── install.sh
│       ├── launch
│       │   └── optor_stereo.launch
│       ├── optor_VISensor_Setups.txt
│       ├── package.xml
│       └── src
│           ├── loitorcam.cpp
│           ├── loitorimu.cpp
│           ├── loitorusb.cpp
│           └── stereo_visensor_cam.cpp
└── SDK -----> C++ SDK 目录
    ├── build
    │   ├── camtest -----> C++ 目标程序
    │   ├── CMakeCache.txt
    │   ├── CMakeFiles
    │   │   └── 3.5.1
    │   │       ├── CMakeCCompiler.cmake
    │   │       ├── CMakeCXXCompiler.cmake
    │   │       ├── CMakeDetermineCompilerABI_C.bin
    │   │       ├── CMakeDetermineCompilerABI_CXX.bin
    │   │       ├── CMakeSystem.cmake
    │   │       ├── CompilerIdC
    │   │       │   ├── a.out
    │   │       │   └── CMakeCCompilerId.c
    │   │       ├── CompilerIdCXX
    │   │       │   ├── a.out
    │   │       │   └── CMakeCXXCompilerId.cpp
    │   └── camtest.dir
    └── build.ninja

```

4.2 C++ 示例程序的编译步骤

此程序是对 optor SDK 最简单用法的示范，请先确认你的系统已经成功安装了 OpenCV。如果没有安装 OpenCV，可以在 OpenCV 的官网下载最新版本。选好你的工作空间，例如: /home/workspace/

1. 复制 /optor_VI_Sensor_SDK_V1.0/SDK 到 /home/workspace/

2. 打开命令行，进入管理员权限

```
sudo -s
```

3. 进入 SDK 目录下

```
cd /home/workspace/optor_VI_Sensor_SDK_V1.0/SDK
```

4. 执行 CMake

```
cmake .
```

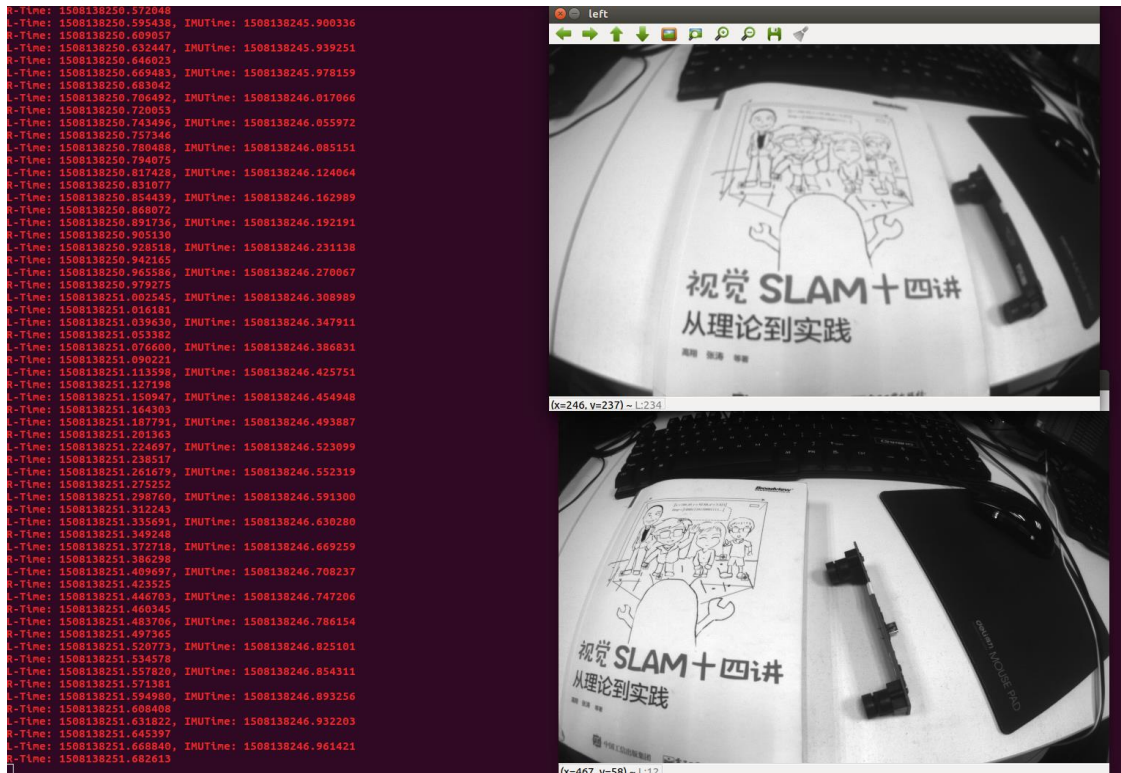
5. 编译示例程序

make

6. 执行 Demo 程序

./camtest2

7. 程序运行结果:



8. 如果遇到串口无法打开导致程序退出的问题，请参见 11.1 的解决办法。

4.3 ROS Package 的编译

此 Package 依赖的 ROS Package:

cv_bridge
image_transport
opencv2 roscpp
rospy sensor_msgs
std_msgs

1. 找到你的 catkin workspace，并进入 /src 目录，例如：

```
cd /home/workspace/catkin_ws/src
```

2. 将/optor_VI_Sensor_SDK_V1.0/ROS/optor_stereo_visensor 完整的复制到

/catkin_ws/src 下

3. 进入 catkin_ws

```
cd /home/workspace/catkin_ws
```

4. 编译

```
catkin_make
```

5. 如果编译成功，则 ROS Node 可以正常运行；否则需要检查 ROS 系统中是否已经安装了 optor ROS Package 所需的依赖包

4.4 ROS Package: 如何启动

1. 运行之前，更新 ROS 环境变量

`source /devel/setup.bash`

2. 一旦步骤 4.3 成功执行安装之后，就可以启动 ROS 节点，命令如下：

`roscore & rosrun optor_stereo_visensor stereo_visensor_node`

`SETTINGS_FILE_PATH`

你需要把其中的 `SETTINGS_FILE_PATH` 替换成“`optor_VISensor_Setups.txt`”的真实路径，例如：

`/home/di-tech/workspace/optor_VI_Sensor_SDK_V1.0/SDK/`

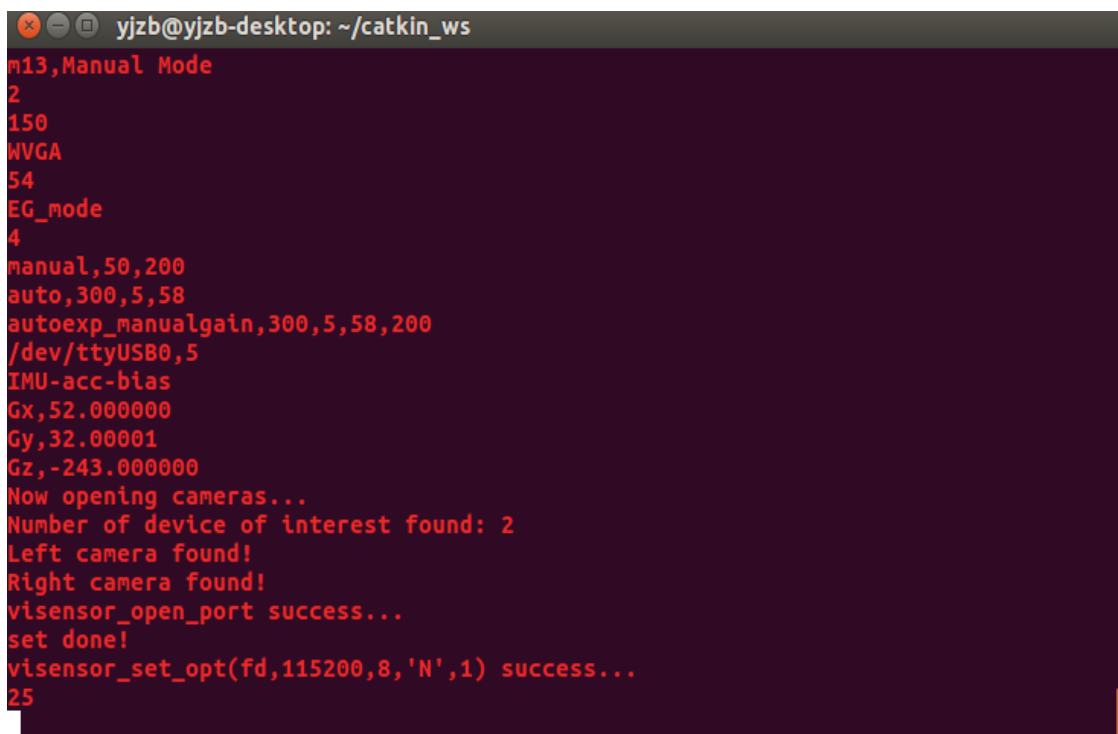
`optor_VISensor_Setups.txt`

或者直接到 `catkin_ws/src/optor_stereo_visensor` 目录下启动

节点，命令：`roscore & rosrun optor_stereo_visensor`

`stereo_visensor_node`

3. 如果出现类似以下的输出结果，则运行成功：

A terminal window titled 'yjzb@yjzb-desktop: ~/catkin_ws' displays the output of the 'roscore & rosrun optor_stereo_visensor stereo_visensor_node' command. The output is in red text on a dark background. It shows various sensor parameters like 'm13, Manual Mode', '2', '150', 'WVGA', '54', 'EG_mode', '4', 'manual, 50, 200', 'auto, 300, 5, 58', 'autoexp_manualgain, 300, 5, 58, 200', '/dev/ttyUSB0, 5', 'IMU-acc-bias', 'Gx, 52.000000', 'Gy, 32.000001', 'Gz, -243.000000', and 'Now opening cameras...'. It then reports 'Number of device of interest found: 2', 'Left camera found!', 'Right camera found!', 'visensor_open_port success...', 'set done!', and 'visensor_set_opt(fd, 115200, 8, 'N', 1) success...'. The terminal ends with '25'.

4. 如果你的 ROS 运行权限不是 root（而是普通用户），可能会遇到串口打开权限的问题而导致 ROS 驱动自动退出，此时可以参照 11.1 的方案解决此问题

5. 获取数据时间戳

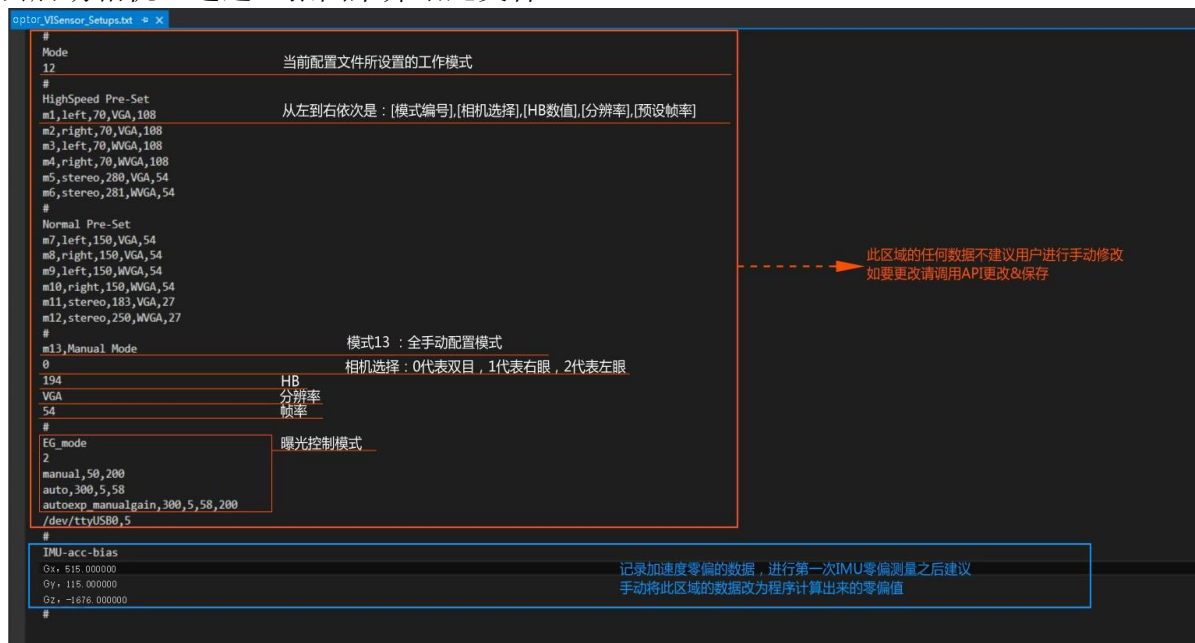
optor ROS 驱动所发布的图像数据以及 IMU 数据均已添加精确时间戳，可以通过 `msg.header.stamp` 得到。

5. 相机配置文件

如果你观察 `camtest2.cpp` 文件的 `main()` 函数的用法，就会发现 SDK 的使用实际上非常简单：

1. `visensor_load_settings("optor_VISensor_Setups.txt");` // 加载相机配置文件
2. `visensor_Start_Cameras();` // 启动相机
3. `visensor_Start_IMU();` // 启动 IMU

在第一行，我们要求系统必须加载相机配置文件，因为这个文件里包含了对相机以及 IMU 硬件进行设置的诸多参数。只有系统找到了这个文件，才能够以正确的寄存器参数去启动相机。通过一张图来介绍此文件：



如图所示，黄色区域不建议做任何修改。蓝色区域用于记录 IMU 零偏，具体设置方法会在步骤 8 中给出。

6. visensor_API 使用说明

6.1 相机控制

1. void visensor_set_auto_EG(int E_AG);

设置曝光控制模式：

E_AG=0 为完全手动曝光，需要手动设置曝光值

E_AG=1 时为带有上下限制的自动曝光（下限为 0，上限 255）E_AG=2 时为

上一种模式的基础上增加了手动设置固定增益值的功能，具体见

visensor_set_gain()

E_AG=3 时为全自动增益&曝光模式

2. void visensor_set_exposure(int _man_exp);

设置手动曝光值，范围 0-255

3. void visensor_set_gain(int _man_gain);

设置手动增益值，范围 0-255

4. void visensor_set_max_autoExp(int max_exp);

设置自动曝光模式曝光值上限，范围 0-255

5. void visensor_set_max_autoExp(int max_exp);

设置自动曝光模式曝光值下限，范围 0-255

6. void visensor_set_resolution(bool set_wvga);

设置分辨率模式，set_wvga=true 时为 752*480 模式，否则为 640*480 模式

7. void visensor_set_fps_mode(bool fps_mode);

设置帧率模式：

fps_mode=true 时为高帧率模式。此模式在 WVGA 分辨率下根据 HB 不同，从 40fps—50fps

此模式在 VGA 分辨率下根据 HB 不同，从 50fps—65fps fps_mode=false 时为低帧率模式。

此模式在 WVGA 分辨率下根据 HB 不同，从 22fps—27fps，此模式在 VGA 分辨率下根据 HB 不同，从 22fps—25fps

8. void visensor_set_current_HB(int HB);

设置当前 HB 值，从 70-255

9. void visensor_set_desired_bin(int db);

设置自动曝光时的“欲达到亮度”，CMOS 会根据此数值进行自动曝光调

节 db 越大代表想要越亮的图像，范围 0-48

10. void visensor_set_cam_selection_mode(int _visensor_cam_selection);

相机通道选择，可选择单独左眼、单独右眼、双目模式

_visensor_cam_selection=0 时为双目模式

_visensor_cam_selection=1 时为右眼模式

_visensor_cam_selection=2 时为左眼模式

11. void visensor_set_current_mode(int _mode);

手动改变当前相机工作模式编号

12. int visensor_Start_Cameras(); | void visensor_Close_Cameras();

启动、安全关闭相机。

其中，visensor_Start_Cameras() 函数必须在 1-11 控制函数之后调用，也就是说如果需要通过 API 改变相机设置，必须在 visensor_Start_Cameras 之前发生，否则设置不会生效。

13. void visensor_save_current_settings();

将当前的模式设置保存到配置文件里，此函数必须在 1-11 控制函数之后调用。

6.2 图像数据读取接口

此系列函数为阻塞式函数，在得到新的图像数据之前不会返回。

1. visensor_imudata visensor_get_stereoImg(char* left_img,char* right_img);

取出当前的左右眼图像（单通道），返回值是一个 visensor_imudata 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

2. visensor_imudata visensor_get_stereoImg(char* left_img,char*

right_img,timeval &left_stamp,timeval &right_stamp);

取出当前的左右眼图像（单通道）并返回拍摄时刻的时间戳

返回值是一个 visensor_imudata 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

3. visensor_imudata visensor_get_leftImg(char* left_img);

只取出左眼图像

返回值是一个 visensor_imudata 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

4. visensor_imudata visensor_get_leftImg(char* left_img,timeval &left_stamp);

只取出左眼图像并返回拍摄时刻的时间戳

返回值是一个 visensor_imudata 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

5.visensor_imudata visensor_get_rightImg(char* right_img);

只取出右眼图像

返回值是一个 visensor_imudata 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

6. visensor_imudata visensor_get_rightImg(char* right_img,timeval

`&right_stamp);`

只取出右眼图像并返回拍摄时刻的时间戳

返回值是一个 `visensor_imudata` 类型对象，其数据是距离本次返回的图像的拍摄时刻最近的时间点采集到的 IMU 数据，时间同步误差在 2.5ms 以内(1/2 个 IMU 采样周期)

6.3 IMU 控制

1. `visensor_Start_IMU();`

打开串口，启动 IMU

2. `void visensor_Close_IMU();`

安全关闭 IMU

3. `void visensor_set_imu_portname(char* input_name);`

手动设置 IMU 串口名称

6.4 IMU 数据

1. `void visensor_set_imu_bias(float bx,float by,float bz);`

手动设置 IMU 零偏

2. `visensor_imudata visensor_imudata_pack`

此变量为全局变量，记录当前的 IMU 数据(包含时间戳)，其结构可以参考

`loitorimu.h`

7. HB一行消隐参数手动设定

HB (Horizontal Blanking) 行消隐，是 CMOS MT9V034 的一项重要寄存器参数，它的大小可以影响帧率，如果设置不恰当(过大或者过小)也会导致图像卡顿、丢帧甚至 USB 连接失效的情况。

HB 越大，每一帧传输所需要的时间就越长，帧率越低；如果你的 USB 总线能力有限，建议将 HB 设置到 250 左右。

HB 越小，采集帧率就越高，但是对于 USB 的传输压力就越大。比较小的 HB 值（比如 120-194）更适合 USB 传输能力较强的 PC 机。

如果你发现相机存在丢帧、卡顿的情况，建议通过 API 修改 HB 值，然后再调用 `visensor_save_current_settings()` 保存设置。

8. IMU 加速度计零偏标定程序

1. 从命令行进入 IMU_Calib 文件夹，启动

`./imucalib`

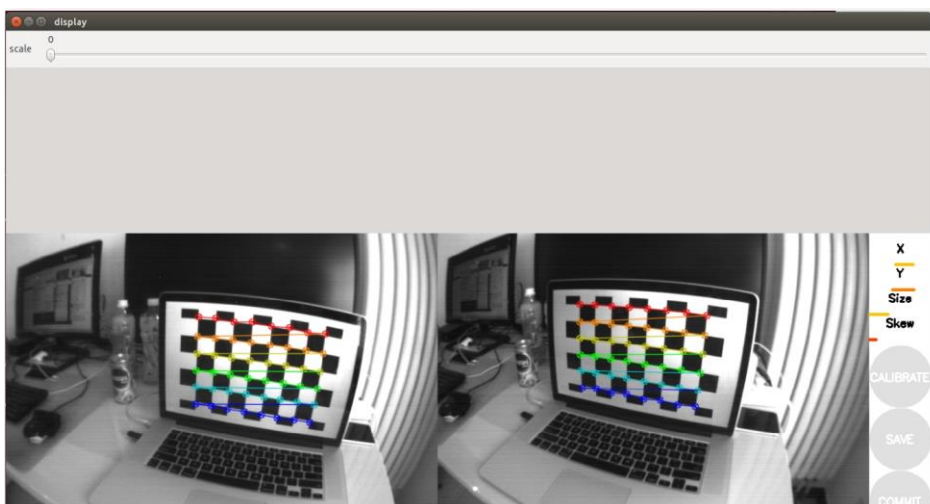
按照提示进行零偏标定。

2. 标定完成后会自动写入配置文件：`imucaldata.txt`

再手动将配置文件中的三个参数依次拷贝到 `Loitor_VISensor_Setups.txt` 中的蓝色区域即可。

9. Camera 光学标定

我们采用 ROS `camera_calibration` 对相机进行了出厂标定，其参数已经写入文件：“相机内参.txt”，具体使用方法详见相机标定教程中的参数复制部分：)



10. 编译 ORB-SLAM2 以及使用 optor 相机测试的步骤

10.1 按照 ORB-SLAM2 的官方指导编译、配置 ORB-SLAM2

10.2 由于光学标定已经完成，编译完成之后，首先运行 optor ROS Node，再按默认配置运行 ORB-SLAM2 即可

11. 可能遇到的问题

11.1 串口文件无法打开此问题可能有

以下几方面原因造成：

1. 串口名称不对；此时需要重新查看 USB 设备的串口路径，并将路径写入 `optor_VISensor_Setups.txt` 中相应部分
2. 普通用户下无法打开串口文件（权限问题）：

Linux 下的设备使用都需要使用 `sudo` 或 `root` 用户才能打开，为了能让普通用户也能使用串口，在命令行中输入：

```
sudo gpasswd -a [yourusername] dialout
```

其中 `yourusername` 为你的当前用户名；注销并且重新登录，即可解决此问题。

11.2 普通用户下无法启动相机

若出现此问题，请在命令行中以普通用户运行“`optor-vi-install.sh`”

```
sudo ./optor-vi-install.sh
```

11.2 IMU 上电时需要等待 8-10 秒，初始化陀螺仪零偏，保持相机静止，否则会出现肉眼可见的姿态漂移

建议加入 optor 售后支持群：590927713 及时反馈遇到的问题，并且得到帮助 :)
GitHub 代码托管：github.com/optor-vis