MYNT® EYE S SDK Guide

Release 2.2.2

JohnZhao

Dec 16, 2018

CONTENTS

CHAPTER

ONE

MYNT® EYE

1.1 产品介绍

小觅双目摄像头（MYNT® EYE）标准版，利用摄像头和运动传感器的互补性，可为视觉 SLAM 的研究提供精度更高、成本更低、部署简单且可以实现人脸和物体识别的视觉研发硬件。就目前而言，”双目+IMU”的研发方式仍然是SLAM 研发的最前沿方向。

作为一款针对立体视觉计算应用进行深入研发的硬件产品，小觅双目摄像头（MYNT® EYE）标准版可广泛应用于视觉定位导航（vSLAM）领域，包括：无人车和机器人的视觉实时定位导航系统、无人机视觉定位系统、无人驾驶避障导航系统、增强现实（AR）、虚拟现实（VR）等；同时，双目也可应用于视觉识别领域，包括：立体人脸识别、三维物体识别、空间运动追踪、三维手势与体感识别等；亦可应用于测量领域，包括：辅助驾驶系统（ADAS）、双目体积计算、工业视觉筛检等。

结合自研的帧同步、自动曝光及白平衡控制等摄像头技术，小觅双目摄像头（MYNT® EYE）标准版可以输出高精度同步的图像源，帮助降低算法研发难度，加快算法研发效率。同时，标准版产品标配六轴传感器（IMU）和红外主动光探测器（IR）。其中，六轴传感器（IMU）可为视觉定位算法的研究提供数据的互补和校正，适用于视觉惯性里程计（VIO）的算法研究，帮助提升定位精度；红外主动光探测器（IR）可以帮助解决室内白墙和无纹理物体的识别难题，提升图像源的识别精度。

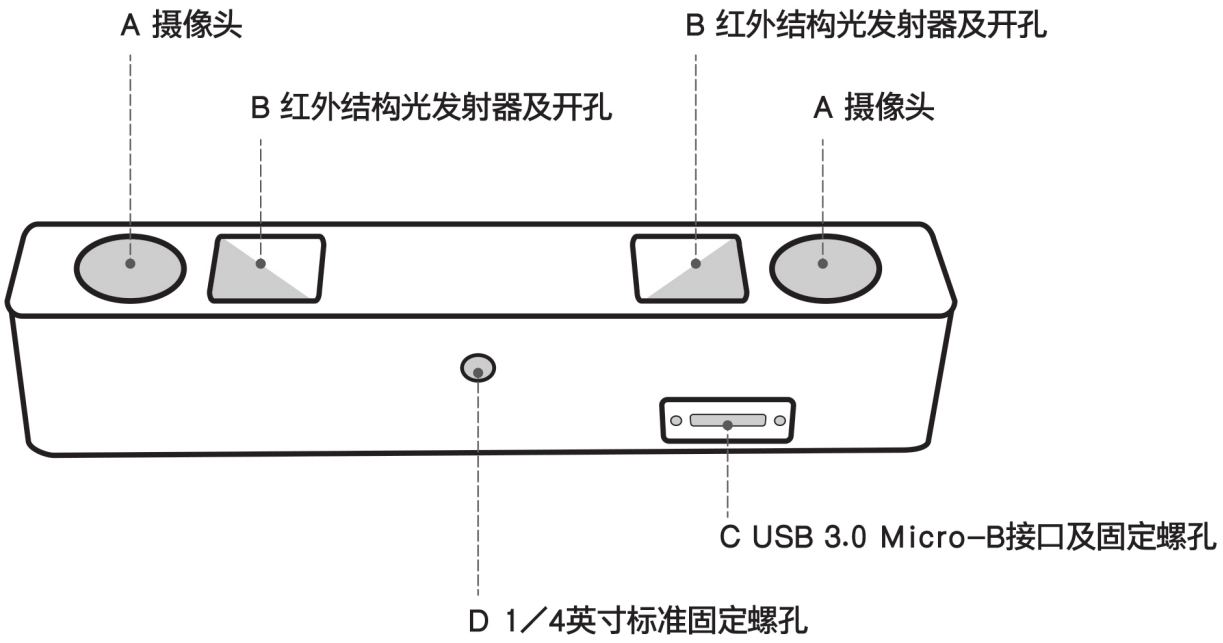
为保证摄像头产品输出数据质量，产品出厂时，我们已对双目进行标定。同时，产品通过权威顶尖实验室高温高湿持续工作、高温高湿持续操作、低温动态老化、高温工作、低温存储、整机冷热冲击、正弦振动、随机振动等多项硬件稳定性测试，保证品质的稳定和可靠。除了产品和技术的研发，亦可直接应用于产品量产，加速从研发到产品化的过程。

产品推出至今，获得了国内外诸多知名企业认可，除了已经面世的标准版“S”系列，未来还将推出内置深度计算能力的“D”系列产品。客户可根据目标平台的算力、运算架构等因素，选择合适的产品。

以成为全球顶尖的立体视觉计算技术解决方案供应商为目标，我们一直在不断优化与完善软硬件，让小觅双目摄像头成为最优的立体视觉技术研发与应用平台。

1.2 尺寸与结构

外壳(mm)	PCBA板(mm)
165x31.5x29.6	149x24



A. 摄像头：摄像头传感器镜头，在使用中请注意保护，以避免成像质量下降。

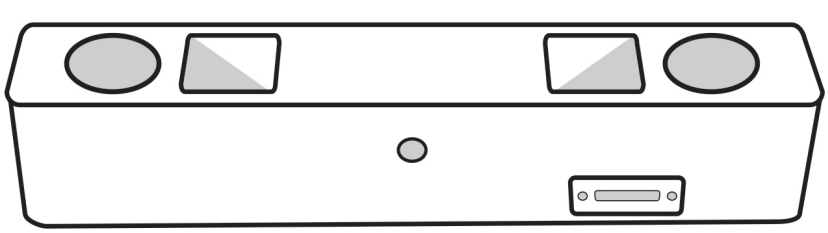
B. 红外结构光发射器及开孔：通过红外结构光可有效解决白墙等无纹理表面的视觉计算。(非 IR 版，此孔保留，但内部无结构光发射装置)

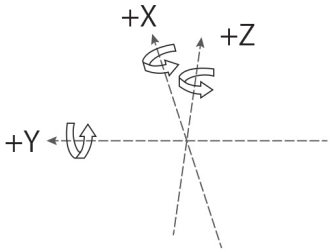
C. USB Micro-B 接口及固定孔：使用中，插上 USB Micro-B 数据线后，请使用接口端的螺丝紧固接口，以避免使用中损坏接口，也保证数据连接的稳定性。

D. 1/4 英寸标准固定螺孔：用于将双目摄像头固定于摄影三角架等装置。

1.3 IMU 坐标系统

IMU 坐标系统为右手系，坐标轴方向如下：




+X
+Z
+Y

CHAPTER

TWO

MYNT® EYE SDK

2.1 支持平台

SDK 是基于 CMake 构建的，用以跨 Linux, Windows, macOS 等多个平台。SDK提供两种安装方式：下载安装以及源码编译安装。

已测试可用的平台有：

- Windows 10
- Ubuntu 16.04 / 14.04
- Jetson TX2

Tip: ubuntu 14.04暂不支持直接下载安装，只能通过源码编译安装。

Warning: 由于硬件传输速率要求，务必使用 USB 3.0 接口。另外，虚拟机因其大多存在 USB 驱动兼容性问题，不建议使用。

2.2 Ubuntu SDK PPA 安装

Ubuntu 16.04
✓

2.2.1 PPA安装

```
$ sudo add-apt-repository ppa:slightech/mynteye2
$ sudo apt-get update
$ sudo apt-get install mynteye2
```

2.2.2 运行样例

Tip: samples 路径: /opt/mynteye/samples; tools 路径: /opt/mynteye/tools

```
$ cd /opt/mynteye/samples
$ ./api/camera_a
```

2.3 Windows SDK exe 安装

Windows 10
✓

2.3.1 下载并安装 SDK

Tip: 下载地址： mynteye-s-2.2.2-win-x64-opencv-3.4.3.exe Google Drive 百度网盘 。

安装完 SDK 的 exe 安装包后，桌面会生成 SDK 根目录的快捷方式。

进入 `<SDK_ROOT_DIR>\bin\samples\tutorials` 目录，双击 `get_stereo.exe` 运行，即可看到双目实时画面。

2.3.2 生成样例工程

首先，安装好 Visual Studio 2017 和 CMake 。

接着，进入 `<SDK_ROOT_DIR>\samples` 目录， 双击 `generate.bat` 即可生成样例工程。

p.s. 样例教程，可见 SDK 主页给出的 Guide 文档。

2.3.3 如何于 Visual Studio 2017 下使用 SDK

进入 `<SDK_ROOT_DIR>\projects\vs2017` ，见 `README.md` 说明。

2.4 Ubuntu SDK 源码安装

Ubuntu 16.04	Ubuntu 14.04
✓	✓

Tip: 如果是其他 Linux 发行版，不是用的 `apt-get` 包管理工具，那你准备依赖时不能 `make init` 自动安装，得自己手动安装了。必要安装项如下：

Linux	How to install required packages
Debian based	sudo apt-get install build-essential cmake git libv4l-dev
Red Hat based	sudo yum install make gcc gcc-c++ kernel-devel cmake git libv4l-devel
Arch Linux	sudo pacman -S base-devel cmake git v4l-utils

2.4.1 获取代码

```
sudo apt-get install git
git clone https://github.com/slightech/MYNT-EYE-S-SDK.git
```

2.4.2 准备依赖

```
cd <sdk>
make init
```

- OpenCV

Tip: OpenCV 如何编译安装，请见官方文档 Installation in Linux 。或参考如下命令：

```
[compiler] sudo apt-get install build-essential
[required] sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev␣
↪libavformat-dev libswscale-dev
[optional] sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-
↪dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

$ git clone https://github.com/opencv/opencv.git
$ cd opencv/
$ git checkout tags/3.4.1

$ mkdir _build
$ cd _build/

$ cmake \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_INSTALL_PREFIX=/usr/local \
\
-DWITH_CUDA=OFF \
\
-DBUILD_DOCS=OFF \
-DBUILD_EXAMPLES=OFF \
-DBUILD_TESTS=OFF \
-DBUILD_PERF_TESTS=OFF \
..

$ make -j4
$ sudo make install
```

2.4.3 编译代码

Tip: 如果 OpenCV 安装到了自定义目录或想指定某一版本，编译前可如下设置路径：

```
# OpenCV_DIR 为 OpenCVConfig.cmake 所在目录
export OpenCV_DIR=~/opencv
```

不然， CMake 会提示找不到 OpenCV 。如果不想依赖 OpenCV ， 请阅读 *OpenCV 不依赖* 。

编译并安装：

```
cd <sdk>
make install
```

最终，默认会安装在 <sdk>/_install 目录。

2.4.4 编译样例

```
cd <sdk>
make samples
```

运行样例：

```
./samples/_output/bin/api/camera_a
```

教程样例，请阅读 *MYNT® EYE 数据* 和 *MYNT® EYE 控制* 。

2.4.5 编译工具

```
cd <sdk>
make tools
```

安装脚本依赖：

```
cd <sdk>/tools/
sudo pip install -r requirements.txt
```

工具和脚本的使用，后续会有介绍。

2.4.6 结语

工程要引入 SDK 的话，CMake 可参考 samples/CMakeLists.txt 里的配置。不然，就是直接引入安装目录里的头文件和动态库。

2.5 Windows SDK 源码安装

Windows 10
✓

Tip: Windows 不直接提供 Visual Studio *.sln 工程文件，需要用 CMake 来构建生成。一是 CMake 跨平台、易配置、可持续维护，二是第三方代码（glog, OpenCV）也都是用的 CMake 构建。

Tip: 目前暂未提供二进制安装程序，需要你从源码编译。也是配置开发环境的过程。

2.5.1 前提条件

CMake（提供构建）

- CMake，用于构建编译（必要）。
- Git，用于获取代码（可选）。
- Doxygen，用于生成文档（可选）。

安装好上述工具后，在命令提示符（Command Prompt）里确认可运行此些命令：

```
>cmake --version
cmake version 3.10.1

>git --version
git version 2.11.1.windows.1

>doxygen --version
1.8.13
```

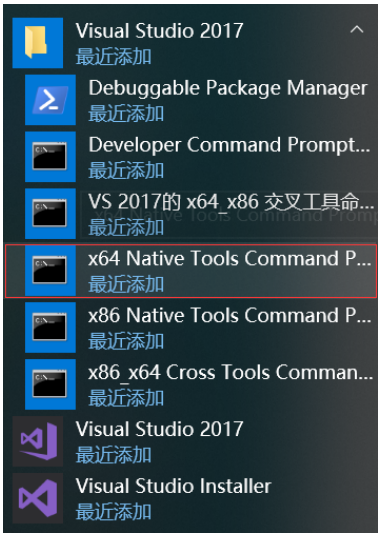
Visual Studio（提供编译）

- Visual Studio
  - Visual Studio 2017
  - Visual Studio 2015
- Windows 10 SDK

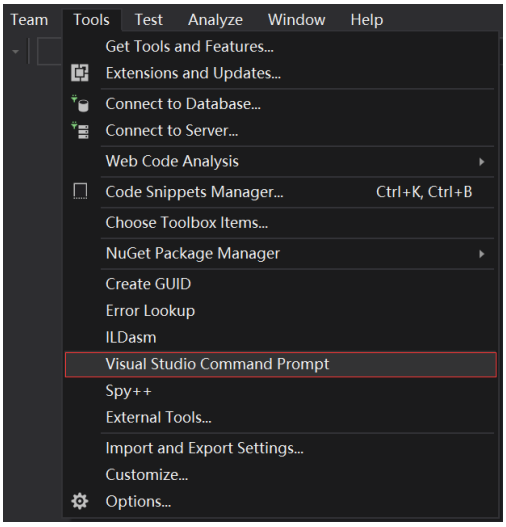
安装好 Visual Studio 后，在其 Visual Studio Command Prompt 里确认可运行如下命令：

```
>cl
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26429.4 for x86

>msbuild
Microsoft (R) 生成引擎版本 15.7.179.6572
```

Tip: Visual Studio Command Prompt 可以从开始菜单打开，



也可以从 Visual Studio 的工具菜单里打开，

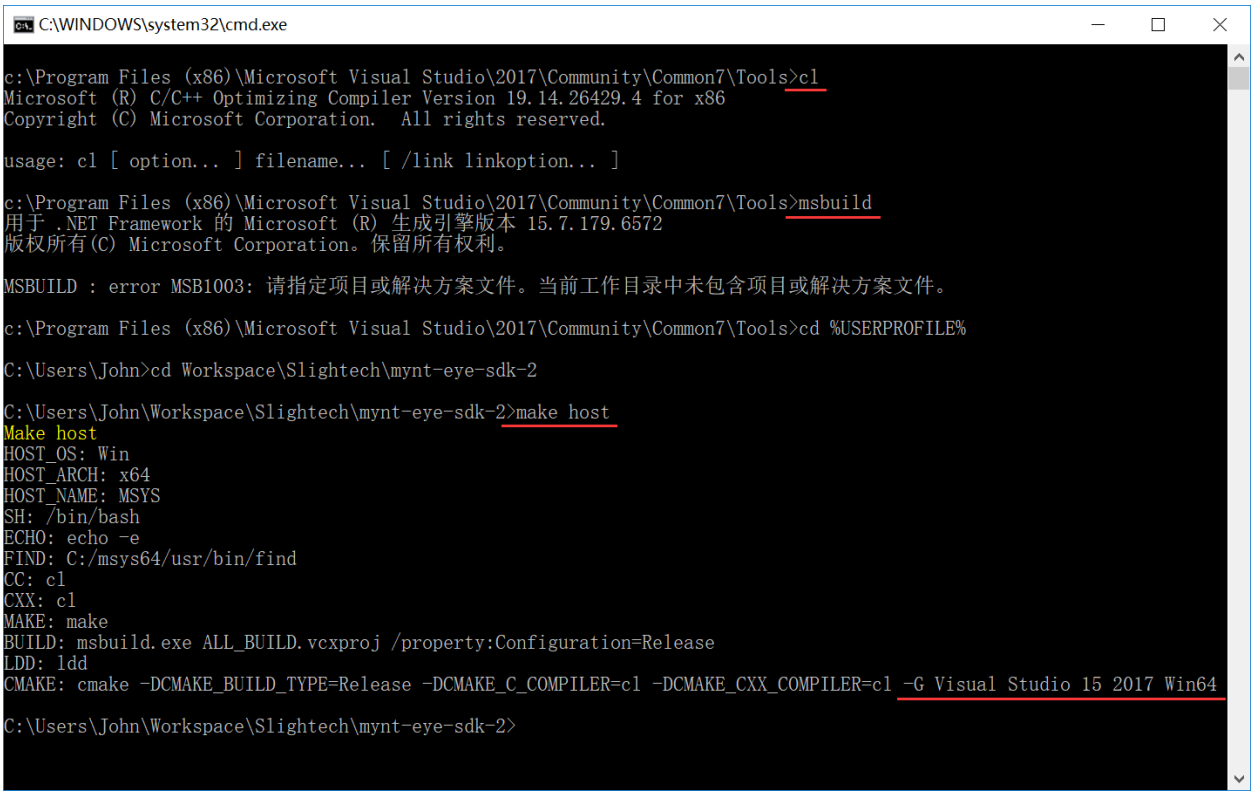


但如 Visual Studio 2015 工具菜单里可能没有，可以自己添加个。

打开 Tools 的 External Tools... ，然后 Add 如下内容：

Field	Value
Title	Visual Studio Command Prompt
Command	`C:\Windows\System32\cmd.exe`
Arguments	`/k "C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\Tools\VsDevCmd.bat"`
Initial Directory	`$(SolutionDir)`

Visual Studio Command Prompt 里就可以用编译命令 `cl` `link` `lib` `msbuild` 等，



MSYS2（提供 Linux 命令）

- MSYS2
  - 国内镜像
  - pacman

安装后，确认系统环境变量 PATH 里添加了如下路径：

```
C:\msys64\usr\bin
```

然后，打开 MSYS2 MSYS ，执行更新并安装 make ：

```
$ pacman -Syu
$ pacman -S make
```

最终，命令提示符（Command Prompt）里可运行如下命令：

```
>make --version
GNU Make 4.2.1
```

2.5.2 获取代码

```
git clone https://github.com/slightech/MYNT-EYE-S-SDK.git
```

2.5.3 准备依赖

```
>cd <sdk>
>make init
Make init
Init deps
Install cmd: pacman -S
Install deps: git clang-format
pacman -S clang-format (not exists)
error: target not found: clang-format
pip install --upgrade autopep8 cpplint pylint requests
...
Init git hooks
ERROR: clang-format-diff is not installed!
Expect cmake version >= 3.0
cmake version 3.10.1
```

- OpenCV

Tip: OpenCV 官方提供了 exe 进行安装。如果想从源码编译，请见官方文档 Installation in Windows 。或参考如下命令：

```
>git clone https://github.com/opencv/opencv.git
>cd opencv
>git checkout tags/3.4.1

>cd opencv
>mkdir _build
>cd _build

>cmake ^
-D CMAKE_BUILD_TYPE=RELEASE ^
-D CMAKE_INSTALL_PREFIX=C:/opencv ^
-D WITH_CUDA=OFF ^
-D BUILD_DOCS=OFF ^
-D BUILD_EXAMPLES=OFF ^
-D BUILD_TESTS=OFF ^
-D BUILD_PERF_TESTS=OFF ^
-G "Visual Studio 15 2017 Win64" ^
..

>msbuild ALL_BUILD.vcxproj /property:Configuration=Release
>msbuild INSTALL.vcxproj /property:Configuration=Release
```

2.5.4 编译代码

Tip: 如果 OpenCV 安装到了自定义目录或想指定某一版本，编译前可如下设置路径：

```
# OpenCV_DIR 为 OpenCVConfig.cmake 所在目录
set OpenCV_DIR=C:\opencv
```

不然， CMake 会提示找不到 OpenCV 。如果不想依赖 OpenCV ， 请阅读 *OpenCV 不依赖* 。

编译并安装：

```
cd <sdk>
make install
```

最终，默认会安装在 <sdk>/_install 目录。

2.5.5 编译样例

```
cd <sdk>
make samples
```

运行样例：

```
.\samples\_output\bin\api\camera_a.bat
```

教程样例，请阅读 *MYNT® EYE 数据* 和 *MYNT® EYE 控制* 。

Tip: 所有编译出的样例程序 exe 都会有个相应的 bat 。 bat 会临时设定下系统环境变量，然后再运行 exe 。所以建议执行 bat 运行程序。

如果直接运行 exe 的话，可能会报 dll 找不到。说明你需要将 <sdk>_install\bin %OPENCV_DIR%\bin 加入到系统环境变量 PATH 里。

OpenCV 如何设定环境变量，可见官方文档 Set the OpenCV environment variable and add it to the systems path 。

2.5.6 编译工具

```
cd <sdk>
make tools
```

工具和脚本的使用，后续会有介绍。

Tip: 脚本为 Python 实现，需要先安装 Python 及其包管理工具 pip ，然后再如下安装依赖：

```
cd <sdk>\tools
pip install -r requirements.txt
```

注：MSYS2 里也带了 Python ，但测试未能安装上 matplotlib 。

2.5.7 结语

工程要引入 SDK 的话，CMake 可参考 samples/CMakeLists.txt 里的配置。不然，就是直接引入安装目录里的头文件和动态库。

2.6 MacOS 安装 x

TODO

2.7 ROS 安装

ROS Kinetic	ROS Indigo
✓	✓

2.7.1 环境准备

- ROS

ROS Kinetic (Ubuntu 16.04)

```
wget https://raw.githubusercontent.com/oroca/oroca-ros-pkg/master/ros_install.sh && \
chmod 755 ./ros_install.sh && bash ./ros_install.sh catkin_ws kinetic
```

ROS Indigo (Ubuntu 14.04)

```
wget https://raw.githubusercontent.com/oroca/oroca-ros-pkg/master/ros_install.sh && \
chmod 755 ./ros_install.sh && bash ./ros_install.sh catkin_ws indigo
```

2.7.2 编译代码

```
cd <sdk>
make ros
```

2.7.3 运行节点

```
source wrappers/ros/devel/setup.bash
roslaunch mynt_eye_ros_wrapper mynteye.launch
```

运行节点，同时打开 RViz 预览：

```
source wrappers/ros/devel/setup.bash
roslaunch mynt_eye_ros_wrapper display.launch
```

2.7.4 测试服务

运行节点，有提供获取设备信息服务，如下测试：

```
$ source wrappers/ros/devel/setup.bash
$ rosrun mynt_eye_ros_wrapper get_device_info.py
LENS_TYPE: 0000
SPEC_VERSION: 1.0
NOMINAL_BASELINE: 120
HARDWARE_VERSION: 2.0
IMU_TYPE: 0000
```

(continues on next page)

(continued from previous page)

```
SERIAL_NUMBER: 0610243700090720
FIRMWARE_VERSION: 2.0
DEVICE_NAME: MYNT-EYE-S1000
```

2.7.5 常见问题 - ROS Indigo

make ros 时 libopencv 找不到

```
make[3]: *** No rule to make target `/usr/lib/x86_64-linux-gnu/libopencv_videostab.so.
↪2.4.8', needed by `/home/john/Workspace/MYNT-EYE-S-SDK/wrappers/ros/devel/lib/
↪libmynteye_wrapper.so'.  Stop.
```

Solution 1) 安装 OpenCV 2:

```
sudo apt-get update
sudo apt-get install libcv-dev
```

Solution 2) 安装 OpenCV 3 并重编 `cv_bridge`:

```
sudo apt-get install ros-indigo-opencv3

git clone https://github.com/ros-perception/vision_opencv.git
mv vision_opencv/cv_bridge/ MYNT-EYE-S-SDK/wrappers/ros/src/
```

然后，重新 `make ros`。

2.7.6 结语

关于如何使用，请阅读 *ROS 如何使用*。

2.8 OpenCV 不依赖

SDK 提供了三层接口，其 OpenCV 依赖情况如下：

- `api`，上层接口，依赖 OpenCV。
- `device`，中间层接口，不依赖 OpenCV。
- `uvc`，底层接口，不依赖 OpenCV。

如果不想使用 OpenCV，你可编辑 `<sdk>/cmake/Option.cmake` 里的 `WITH_API` 选项，设为 `OFF` 就能关闭 `api` 层代码编译：

```
option(WITH_API "Build with API layer, need OpenCV" ON)
```

`device` 层接口使用样例，请见 device/camera.cc。

CHAPTER

THREE

MYNT® EYE 固件

3.1 固件与 SDK 适配性

Firmwares	SDK Version
MYNTEYE_S_2.0.0_rc.img	2.0.0-rc (2.0.0-rc ~ 2.0.0-rc2)
MYNTEYE_S_2.0.0_rc2.img	2.0.0-rc2 (2.0.0-rc ~ 2.0.0-rc2)
MYNTEYE_S_2.0.0_rc1.img	2.0.0-rc1
MYNTEYE_S_2.0.0_rc0.img	2.0.0-rc0 (2.0.0-rc1 ~ 2.0.0-alpha1)
MYNTEYE_S_2.0.0_alpha1.1.img	2.0.0-alpha1 (2.0.0-rc1 ~ 2.0.0-alpha1)
MYNTEYE_S_2.0.0_alpha1.img	2.0.0-alpha1 (2.0.0-rc1 ~ 2.0.0-alpha1)
MYNTEYE_S_2.0.0_alpha0.img	2.0.0-alpha0

`Firmwares` 表明固件文件名称。其在 MYNTEYE_BOX 的 `Firmwares` 目录内。

`SDK Version` 表明该固件适配的 SDK 版本，括号内指可用版本范围。

3.2 固件如何进行升级

固件升级，需要使用我们提供的固件升级程序：MYNT EYE TOOL。

固件及MYNT EYE TOOL的安装包，都在 MYNTEYE_BOX 的 `Firmwares` 目录内。文件结构如下：

```
Firmwares/
├─Checksum.txt                  # file checksum
├─MYNTEYE_S_2.0.0_rc0.img       # firmware
├─...
└─setup.zip                    # MYNTEYE TOOL zip
```

固件升级程序，目前仅支持 Windows ，所以需要你在 Windows 下进行操作。步骤如下：

3.2.1 下载准备

- 下载并解压 `setup.zip`。
- 下载固件，如 `MYNTEYE_S_2.0.0_rc0.img` 。
  - 请见 固件与 *SDK* 适配性 选择适合当前 SDK 版本的固件。
  - 请见 `Checksum.txt` 找到下载固件的校验码，如下获取并比对：
    * 命令提示符（CMD）里运行 `certutil -hashfile <*.img> MD5` 。

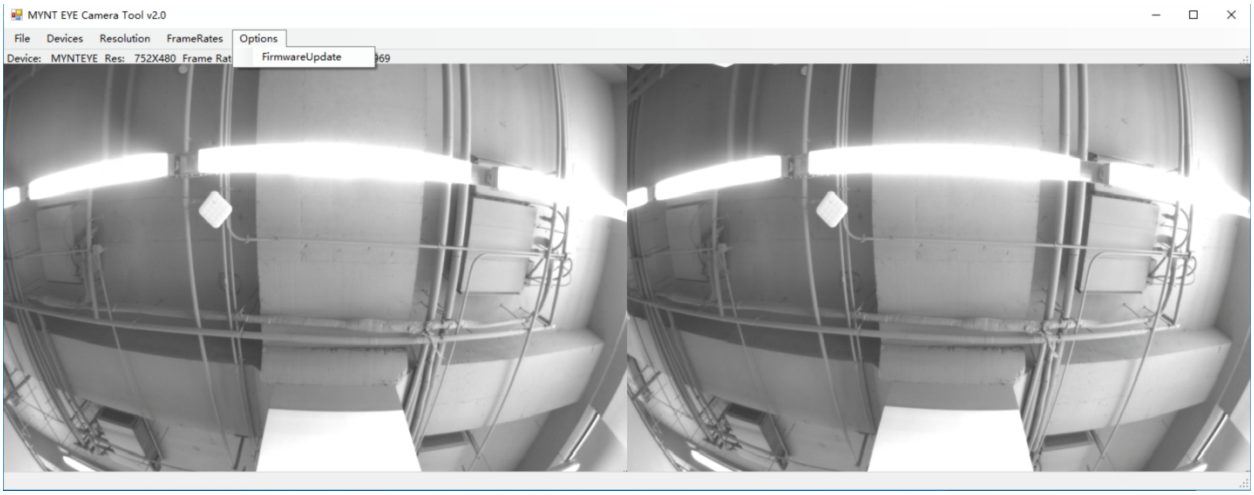
* 校验码不正确的话，说明下载有误，请重新下载！

3.2.2 安装MYNT EYE TOOL

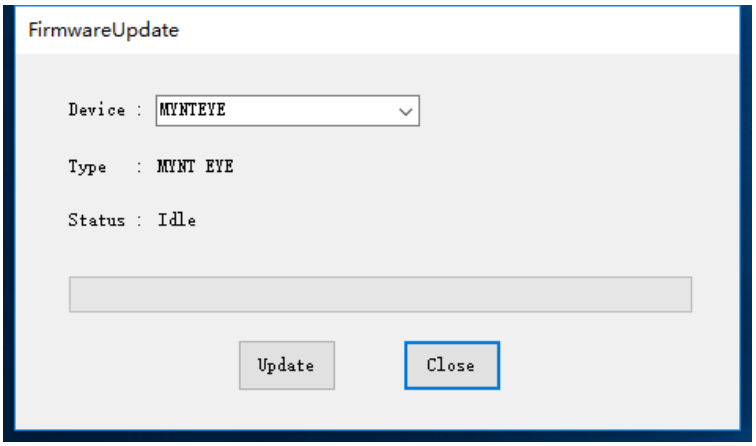
- 双击 `setup.msi` 安装固件升级程序。

3.2.3 升级固件

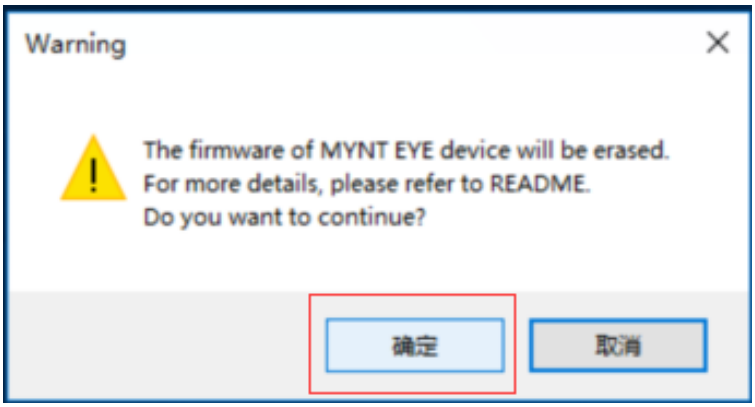
- USB3.0 口插上 MYNT® EYE 设备。
- 打开MYNT EYE TOOL，选择 `Options/FirmwareUpdate` 。

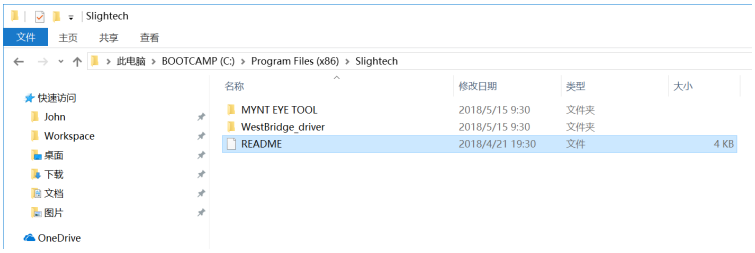


- 点击 `Update` 。

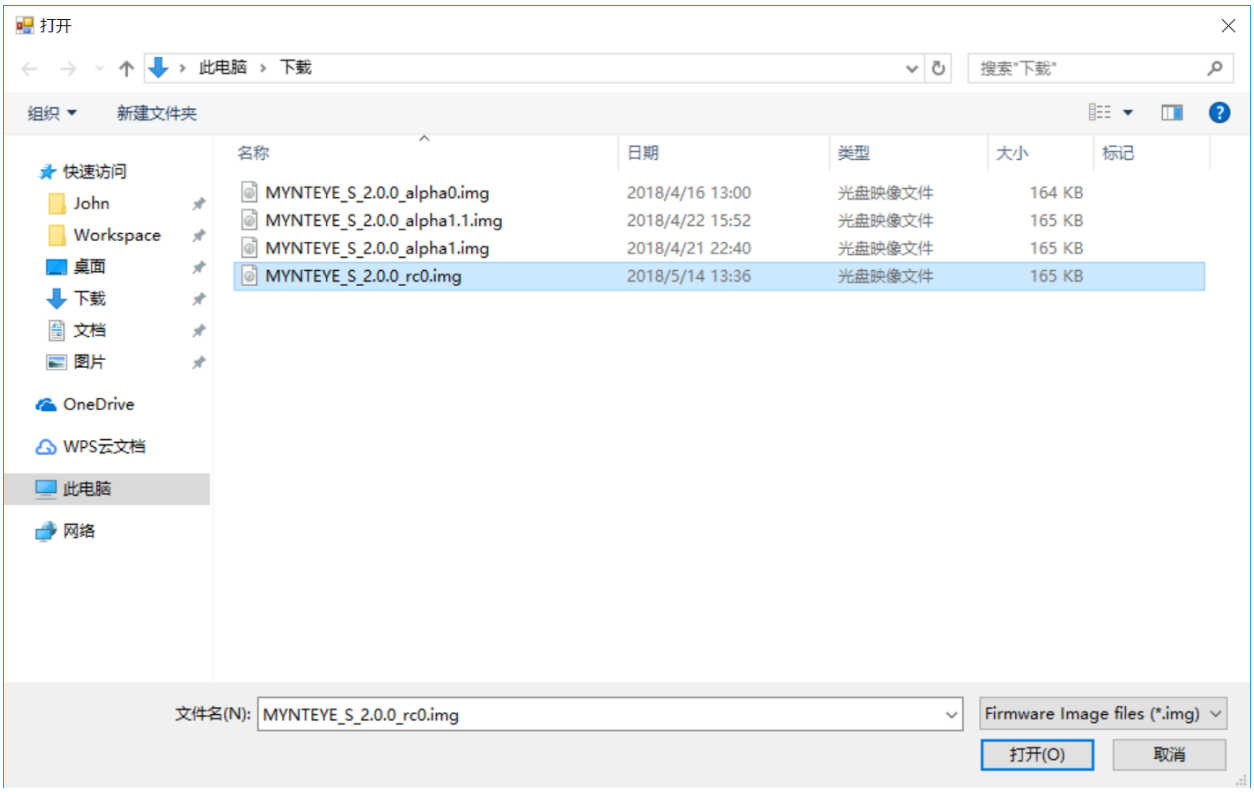


- 弹出警告对话框，直接 确定 即可。
  - 由于该操作会擦除固件，所以弹出警告。详情见 README 。
    * 通常在升级过程中，MYNT EYE TOOL会自动安装驱动。
    * 如果升级遇到问题，参考 README 解决。

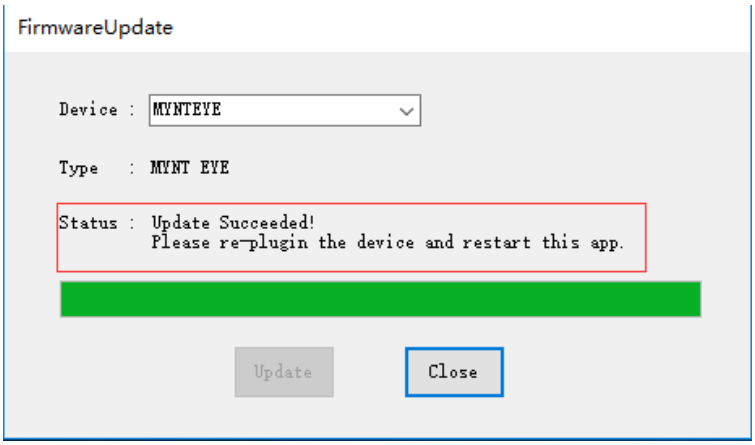




- 在打开的文件选择框里，选择要升级的固件，开始升级。



- 升级完成后，状态变为 `Succeeded` 。



- 关闭MYNT EYE TOOL，结束。

Warning: 固件升级后，初次打开 MYNT® EYE 设备时，请静置 3 秒，其会有一个零漂补偿过程。或者，请主动调用控制接口 `RunOptionAction(Option::ZERO_DRIFT_CALIBRATION)` 来进行零漂补偿。

3.3 从 1.x 换到 2.x

SDK 如果要从 1.x 换到 2.x 版本，你需要做的几件事：

1） 安装 SDK 2 ，见 *MYNT® EYE SDK* 。

2） 升级固件到 2.x 版本，见 *MYNT® EYE* 固件 。

3） SDK 1.x 运行后，图像标定参数会保存到 `<MYNTEYE_SDK_ROOT>/settings/SN*.conf` 。请按照 写入图像标定参数 将 `SN*.conf` 写入设备。

CHAPTER

FOUR

MYNT® EYE 数据

4.1 获取设备信息

通过 API 的 `GetInfo()` 函数，就可以获取当前打开设备的各类信息值。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

LOG(INFO) << "Device name: " << api->GetInfo(Info::DEVICE_NAME);
LOG(INFO) << "Serial number: " << api->GetInfo(Info::SERIAL_NUMBER);
LOG(INFO) << "Firmware version: " << api->GetInfo(Info::FIRMWARE_VERSION);
LOG(INFO) << "Hardware version: " << api->GetInfo(Info::HARDWARE_VERSION);
LOG(INFO) << "Spec version: " << api->GetInfo(Info::SPEC_VERSION);
LOG(INFO) << "Lens type: " << api->GetInfo(Info::LENS_TYPE);
LOG(INFO) << "IMU type: " << api->GetInfo(Info::IMU_TYPE);
LOG(INFO) << "Nominal baseline: " << api->GetInfo(Info::NOMINAL_BASELINE);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/get_device_info
I0503 16:40:21.109391 32106 utils.cc:13] Detecting MYNT EYE devices
I0503 16:40:21.604116 32106 utils.cc:20] MYNT EYE devices:
I0503 16:40:21.604127 32106 utils.cc:24]   index: 0, name: MYNT-EYE-S1000
I0503 16:40:21.604142 32106 utils.cc:30] Only one MYNT EYE device, select index: 0
I0503 16:40:21.615054 32106 get_device_info.cc:10] Device name: MYNT-EYE-S1000
I0503 16:40:21.615113 32106 get_device_info.cc:11] Serial number: 0610243700090720
I0503 16:40:21.615129 32106 get_device_info.cc:12] Firmware version: 2.0
I0503 16:40:21.615139 32106 get_device_info.cc:13] Hardware version: 2.0
I0503 16:40:21.615146 32106 get_device_info.cc:14] Spec version: 1.0
I0503 16:40:21.615155 32106 get_device_info.cc:15] Lens type: 0000
I0503 16:40:21.615164 32106 get_device_info.cc:16] IMU type: 0000
I0503 16:40:21.615171 32106 get_device_info.cc:17] Nominal baseline: 120
```

完整代码样例，请见 get_device_info.cc 。

4.2 获取图像标定参数

通过 API 的 `GetIntrinsics()` `GetExtrinsics()` 函数，就可以获取当前打开设备的图像标定参数。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

LOG(INFO) << "Intrinsics left: {" << api->GetIntrinsics(Stream::LEFT) << "}";
LOG(INFO) << "Intrinsics right: {" << api->GetIntrinsics(Stream::RIGHT)
          << "}";
LOG(INFO) << "Extrinsics left to right: {"
          << api->GetExtrinsics(Stream::LEFT, Stream::RIGHT) << "}";
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/get_img_params
I0510 15:00:22.643263  6980 utils.cc:26] Detecting MYNT EYE devices
I0510 15:00:23.138811  6980 utils.cc:33] MYNT EYE devices:
I0510 15:00:23.138849  6980 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0510 15:00:23.138855  6980 utils.cc:43] Only one MYNT EYE device, select index: 0
I0510 15:00:23.210491  6980 get_img_params.cc:23] Intrinsics left: {width: 752,␣
↪height: 480, fx: 736.38305001095545776, fy: 723.50066150722432212, cx: 356.
↪91961817119693023, cy: 217.27271340923883258, model: 0, coeffs: [-0.
↪54898645145016478, 0.52837141203888638, 0.00000000000000000, 0.00000000000000000, 0.
↪00000000000000000]}
I0510 15:00:23.210551  6980 get_img_params.cc:24] Intrinsics right: {width: 752,␣
↪height: 480, fx: 736.38305001095545776, fy: 723.50066150722432212, cx: 456.
↪68367112303980093, cy: 250.70083335536796199, model: 0, coeffs: [-0.
↪51012886039889305, 0.38764476500996770, 0.00000000000000000, 0.00000000000000000, 0.
↪00000000000000000]}
I0510 15:00:23.210577  6980 get_img_params.cc:26] Extrinsics left to right:
↪{rotation: [0.99701893306553813, -0.00095378124886237, -0.07715139279485062, 0.
↪00144939967628305, 0.99997867219985104, 0.00636823256494144, 0.07714367342455503, -
↪0.00646107164115277, 0.99699905125522237], translation: [-118.88991734400046596, -0.
↪04560580387053091, -3.95313736911933855]}
```

完整代码样例，请见 get_img_params.cc 。

4.3 获取 IMU 标定参数

通过 API 的 `GetMotionIntrinsics()` `GetMotionExtrinsics()` 函数，就可以获取当前打开设备的 IMU 标定参数。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

LOG(INFO) << "Motion intrinsics: {" << api->GetMotionIntrinsics() << "}";
LOG(INFO) << "Motion extrinsics left to imu: {"
          << api->GetMotionExtrinsics(Stream::LEFT) << "}";
```

完整代码样例，请见 get_imu_params.cc 。

4.4 获取双目原始图像

API 提供了 `Start()` `Stop()` 函数，用于开始或停止捕获数据。如果只捕获图像数据的话，参数用 `Source::VIDEO_STREAMING` 即可。

开始捕获数据后，首先调用 `WaitForStreams()` 函数，等待捕获到数据。接着，通过 `GetStreamData()` 函数，就能获取想要的数据了。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

api->Start(Source::VIDEO_STREAMING);

cv::namedWindow("frame");

while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);
  cv::imshow("frame", img);

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::VIDEO_STREAMING);
```

上述代码，用了 OpenCV 来显示图像。选中显示窗口时，按 `ESC/Q` 就会结束程序。

完整代码样例，请见 get_stereo.cc 。

4.5 获取双目纠正图像

API 提供的 `GetStreamData()` 默认仅能获取到硬件的原始数据，例如双目原始图像。

而双目纠正图像，属于上层合成数据。此类数据，需要事先 `EnableStreamData()` 启用，然后 `GetStreamData()` 才能获取到。

另外，`WaitForStreams()` 等待的是关键原始数据。刚开始时，合成数据可能还在处理，取出的是空值，所以需要判断下不为空。

Tip: 如果想要合成数据一生成就立即获取到，请参阅 从回调接口获取数据 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

api->EnableStreamData(Stream::LEFT_RECTIFIED);
api->EnableStreamData(Stream::RIGHT_RECTIFIED);

api->Start(Source::VIDEO_STREAMING);

cv::namedWindow("frame");
```

(continues on next page)

(continued from previous page)

```
while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT_RECTIFIED);
  auto &&right_data = api->GetStreamData(Stream::RIGHT_RECTIFIED);

  if (!left_data.frame.empty() && !right_data.frame.empty()) {
    cv::Mat img;
    cv::hconcat(left_data.frame, right_data.frame, img);
    cv::imshow("frame", img);
  }

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::VIDEO_STREAMING);
```

上述代码，用了 OpenCV 来显示图像。选中显示窗口时，按 ESC/Q 就会结束程序。

完整代码样例，请见 get_stereo_rectified.cc 。

4.6 获取视差图像

视差图像，属于上层合成数据。需要事先 EnableStreamData() 启用，然后 GetStreamData() 获取。另外，判断不为空后再使用。

详细流程说明，请参阅 获取双目原始图像 获取双目纠正图像 。

另外，推荐使用插件计算深度：深度图效果会更好，并且运算速度更快。具体请参阅 使用插件获取数据 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// api->EnableStreamData(Stream::DISPARITY);
api->EnableStreamData(Stream::DISPARITY_NORMALIZED);

api->Start(Source::VIDEO_STREAMING);

cv::namedWindow("frame");
// cv::namedWindow("disparity");
cv::namedWindow("disparity_normalized");

while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);
  cv::imshow("frame", img);
```

(continues on next page)

(continued from previous page)

```
  // auto &&disp_data = api->GetStreamData(Stream::DISPARITY);
  // if (!disp_data.frame.empty()) {
  //   cv::imshow("disparity", disp_data.frame);
  // }

  auto &&disp_norm_data = api->GetStreamData(Stream::DISPARITY_NORMALIZED);
  if (!disp_norm_data.frame.empty()) {
    cv::imshow("disparity_normalized", disp_norm_data.frame);  // CV_8UC1
  }

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::VIDEO_STREAMING);
```

上述代码，用了 OpenCV 来显示图像。选中显示窗口时，按 ESC/Q 就会结束程序。

完整代码样例，请见 get_disparity.cc 。

4.7 获取深度图像

深度图像，属于上层合成数据。需要事先 EnableStreamData() 启用，然后 GetStreamData() 获取。另外，判断不为空后再使用。

详细流程说明，请参阅 获取双目原始图像 获取双目纠正图像 。

另外，推荐使用插件计算深度：深度图效果会更好，并且运算速度更快。具体请参阅 使用插件获取数据 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

api->EnableStreamData(Stream::DEPTH);

api->Start(Source::VIDEO_STREAMING);

cv::namedWindow("frame");
cv::namedWindow("depth");

while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);
  cv::imshow("frame", img);

  auto &&depth_data = api->GetStreamData(Stream::DEPTH);
  if (!depth_data.frame.empty()) {
    cv::imshow("depth", depth_data.frame);  // CV_16UC1
```

(continues on next page)

(continued from previous page)

```
  }

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::VIDEO_STREAMING);
```

上述代码，用了 OpenCV 来显示图像。选中显示窗口时，按 ESC/Q 就会结束程序。

完整代码样例，请见 get_depth.cc 。

Tip: 预览深度图某区域的值，请见 get_depth_with_region.cc 。

4.8 获取点云图像

点云图像，属于上层合成数据。需要事先 EnableStreamData() 启用，然后 GetStreamData() 获取。另外，判断不为空后再使用。

详细流程说明，请参阅 获取双目原始图像 获取双目纠正图像 。

另外，推荐使用插件计算深度：深度图效果会更好，并且运算速度更快。具体请参阅 使用插件获取数据 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

api->EnableStreamData(Stream::POINTS);

api->Start(Source::VIDEO_STREAMING);

cv::namedWindow("frame");
PCViewer pcviewer;

while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);
  cv::imshow("frame", img);

  auto &&points_data = api->GetStreamData(Stream::POINTS);
  if (!points_data.frame.empty()) {
    pcviewer.Update(points_data.frame);
  }

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
```

(continues on next page)

(continued from previous page)

```
  }
  if (pcviewer.WasStopped()) {
    break;
  }
}

api->Stop(Source::VIDEO_STREAMING);
```

上述代码，用了 PCL 来显示点云。关闭点云窗口时，也会结束程序。

完整代码样例，请见 get_points.cc 。

Attention: 准备好了 PCL 库，编译教程样例时才会有此例子。如果 PCL 库安装到了自定义目录，那么请打开 tutorials/CMakeLists.txt ，找到 `find_package(PCL)` ，把 `PCLConfig.cmake` 所在目录添加进 `CMAKE_PREFIX_PATH` 。

4.9 获取 IMU 数据

API 提供了 `Start() Stop()` 函数，用于开始或停止捕获数据。要捕获 IMU 数据的话，参数用 `Source::MOTION_TRACKING` 。或者 `Source::ALL` 同时捕获图像和 IMU 数据。

开始捕获数据后，需要 `EnableMotionDatas()` 启用缓存，才能通过 `GetMotionDatas()` 函数获取到 IMU 数据。否则，只能通过回调接口得到 IMU 数据，请参阅 从回调接口获取数据 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// Enable this will cache the motion datas until you get them.
api->EnableMotionDatas();

api->Start(Source::ALL);

CVPainter painter;

cv::namedWindow("frame");

while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);

  auto &&motion_datas = api->GetMotionDatas();
  /*
  for (auto &&data : motion_datas) {
    LOG(INFO) << "Imu frame_id: " << data.imu->frame_id
              << ", timestamp: " << data.imu->timestamp
              << ", accel_x: " << data.imu->accel[0]
              << ", accel_y: " << data.imu->accel[1]
```

(continues on next page)

(continued from previous page)

```
              << ", accel_z: " << data.imu->accel[2]
              << ", gyro_x: " << data.imu->gyro[0]
              << ", gyro_y: " << data.imu->gyro[1]
              << ", gyro_z: " << data.imu->gyro[2]
              << ", temperature: " << data.imu->temperature;
  }
  */

  painter.DrawImgData(img, *left_data.img);
  if (!motion_datas.empty()) {
    painter.DrawImuData(img, *motion_datas[0].imu);
  }

  cv::imshow("frame", img);

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::ALL);
```

上述代码，用了 OpenCV 来显示图像和数据。选中显示窗口时，按 ESC/Q 就会结束程序。

完整代码样例，请见 get_imu.cc 。

4.10 从回调接口获取数据

API 提供了 SetStreamCallback() SetMotionCallback() 函数，来设定各类数据的回调。

> **Attention:** 一定不要阻塞回调。如果需要长时间处理数据，请将回调作为数据生产者。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// Attention: must not block the callbacks.

// Get left image from callback
std::atomic_uint left_count(0);
api->SetStreamCallback(
    Stream::LEFT, [&left_count](const api::StreamData &data) {
      CHECK_NOTNULL(data.img);
      ++left_count;
    });

// Get depth image from callback
api->EnableStreamData(Stream::DEPTH);
std::atomic_uint depth_count(0);
cv::Mat depth;
std::mutex depth_mtx;
api->SetStreamCallback(
```

(continues on next page)

(continued from previous page)

```
    Stream::DEPTH,
    [&depth_count, &depth, &depth_mtx](const api::StreamData &data) {
      UNUSED(data)
      ++depth_count;
      {
        std::lock_guard<std::mutex> _(depth_mtx);
        depth = data.frame;
      }
    });

// Get motion data from callback
std::atomic_uint imu_count(0);
std::shared_ptr<mynteye::ImuData> imu;
std::mutex imu_mtx;
api->SetMotionCallback(
    [&imu_count, &imu, &imu_mtx](const api::MotionData &data) {
      CHECK_NOTNULL(data.imu);
      ++imu_count;
      {
        std::lock_guard<std::mutex> _(imu_mtx);
        imu = data.imu;
      }
    });

api->Start(Source::ALL);

CVPainter painter;

cv::namedWindow("frame");
cv::namedWindow("depth");

unsigned int depth_num = 0;
while (true) {
  api->WaitForStreams();

  auto &&left_data = api->GetStreamData(Stream::LEFT);
  auto &&right_data = api->GetStreamData(Stream::RIGHT);

  // Concat left and right as img
  cv::Mat img;
  cv::hconcat(left_data.frame, right_data.frame, img);

  // Draw img data and size
  painter.DrawImgData(img, *left_data.img);

  // Draw imu data
  if (imu) {
    std::lock_guard<std::mutex> _(imu_mtx);
    painter.DrawImuData(img, *imu);
  }

  // Draw counts
  std::ostringstream ss;
  ss << "left: " << left_count << ", depth: " << depth_count
     << ", imu: " << imu_count;
  painter.DrawText(img, ss.str(), CVPainter::BOTTOM_RIGHT);
```

(continues on next page)

(continued from previous page)

```
  // Show img
  cv::imshow("frame", img);

  // Show depth
  if (!depth.empty()) {
    // Is the depth a new one?
    if (depth_num != depth_count || depth_num == 0) {
      std::lock_guard<std::mutex> _(depth_mtx);
      depth_num = depth_count;
      // LOG(INFO) << "depth_num: " << depth_num;
      ss.str("");
      ss.clear();
      ss << "depth: " << depth_count;
      painter.DrawText(depth, ss.str());
      cv::imshow("depth", depth);  // CV_16UC1
    }
  }

  char key = static_cast<char>(cv::waitKey(1));
  if (key == 27 || key == 'q' || key == 'Q') {  // ESC/Q
    break;
  }
}

api->Stop(Source::ALL);
```

上述代码，用了 OpenCV 来显示图像和数据。选中显示窗口时，按 ESC/Q 就会结束程序。

完整代码样例，请见 get_from_callbacks.cc 。

4.11 使用插件获取数据

API 提供了 EnablePlugin() 函数，以启用某路径下的插件。

官方目前提供了些计算双目视差的插件，在 MYNTEYE_BOX 的 Plugins 目录内。

```
Plugins/
├─linux-x86_64/
│   ├─libplugin_b_ocl1.2_opencv3.4.0.so
│   ├─libplugin_g_cuda9.1_opencv2.4.13.5.so
│   ├─libplugin_g_cuda9.1_opencv3.3.1.so
│   └─libplugin_g_cuda9.1_opencv3.4.0.so
├─tegra-armv8/
└─win-x86_64/
```

- 目录 linux-x86_64 表明了系统和架构。
  - 可从系统信息或 uname -a 得知你的 CPU 架构。
- 库名 libplugin_* 表明了插件标识和第三方依赖。
  - b g 是插件标识，说明用了不同算法。
  - ocl1.2 表明依赖了 OpenCL 1.2 ，如果存在。
  - cuda9.1 表明依赖了 CUDA 9.1 ，如果存在。
  - opencv3.4.0 表明依赖了 OpenCV 3.4.0 ，如果存在。

- mynteye2.0.0 表明依赖了 MYNT EYE SDK 2.0.0，如果存在。

首先，根据具体情况，选择你想测试使用的插件。如果依赖了第三方，那么请安装一致的版本。

然后，参考如下代码启用插件：

```
auto &&api = API::Create(argc, argv);

api->EnablePlugin("plugins/linux-x86_64/libplugin_g_cuda9.1_opencv3.4.0.so");
```

路径可以是绝对路径，也可以是相对路径（相对于当前工作目录）。

最终，和之前一样调用 API 获取数据就行了。

Tip: 如果没有启用插件的话，`api->Start(Source::VIDEO_STREAMING);` 时会自动在 `<sdk>/plugins/<platform>` 目录里找合适的插件去加载。

换句话说，可以把当前平台的插件目录整个搬进 `<sdk>/plugins` 目录内。安装好对应的 CUDA OpenCV 等插件依赖后重编译，此后运行 API 层接口程序，就会自动加载官方插件了。

运行前，请执行如下命令，以确保能搜索到插件的依赖库：

```
# Linux
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
# /usr/local/lib 指依赖库所在路径

# macOS
export DYLD_LIBRARY_PATH=/usr/local/lib:$DYLD_LIBRARY_PATH
# /usr/local/lib 指依赖库所在路径

# Windows
set PATH=C:\opencv\x64\vc14\bin;%PATH%
# 或者，添加进系统环境变量 Path 里。
```

此外，可执行如下命令，检查是否能搜索到插件的依赖库：

```
# Linux
ldd *.so
# *.so 指具体插件路径

# macOS
otool -L *.dylib
# *.dylib 指具体插件路径

# Windows
# 请下载如 Dependency Walker ，打开 DLL 。
```

如果找不到插件的依赖库，加载时将会报错 “Open plugin failed”。

完整代码样例，请见 get_with_plugin.cc 。

Tip: Linux 上也可以把依赖库路径加入系统环境，编译出的程序就可以直接运行了（不需要于终端里 `export LD_LIBRARY_PATH` 再运行）。

- 新建 `/etc/ld.so.conf.d/libmynteye.conf` 文件，写入依赖库路径。
- 终端里执行 `sudo /sbin/ldconfig` 命令，刷新缓存。

Listing 1: e.g. libmynteye.conf

```
# libmynteye configuration
#
# 1) Copy this file to: /etc/ld.so.conf.d/libmynteye.conf
# 2) Run this cmd in Terminal: sudo /sbin/ldconfig

/usr/local/cuda/lib64
$HOME/opencv-3.4.1/lib
```

4.12 保存设备信息和参数

SDK 提供了保存信息和参数的工具 `save_all_infos` 。工具详情可见 tools/README.md 。

参考运行命令：

```
./tools/_output/bin/writer/save_all_infos

# Windows
.\tools\_output\bin\writer\save_all_infos.bat
```

参考运行结果，于 Linux 上：

```
$ ./tools/_output/bin/writer/save_all_infos
I0512 21:40:08.687088  4092 utils.cc:26] Detecting MYNT EYE devices
I0512 21:40:09.366693  4092 utils.cc:33] MYNT EYE devices:
I0512 21:40:09.366734  4092 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0512 21:40:09.366757  4092 utils.cc:43] Only one MYNT EYE device, select index: 0
I0512 21:40:09.367609  4092 save_all_infos.cc:38] Save all infos to "config/
↪SN0610243700090720"
```

默认会保存进 `<workdir>/config` 目录。你也可以加参数，指定保存到其他目录。

保存内容如下：

```
<workdir>/
└─config/
   └─SN0610243700090720/
      ├─device.info
      ├─img.params
      └─imu.params
```

4.13 写入图像标定参数

SDK 提供了写入图像标定参数的工具 `img_params_writer` 。工具详情可见 tools/README.md 。

有关如何获取，请阅读 获取图像标定参数 。此参数会用于计算纠正、视差等。

参考运行命令：

```
./tools/_output/bin/writer/img_params_writer tools/writer/config/img.params

# Windows
.\tools\_output\bin\writer\img_params_writer.bat tools\writer\config\img.params
```

其中， tools/writer/config/img.params 是参数文件路径。如果你自己标定了参数，可以编辑此文件，然后执行上述命令写入设备。

Tip: 旧 SDK 提供的标定参数文件 `SN*.conf` 也可用此工具写入设备。

Warning: 请不要随意覆写参数。另外 `save_all_infos` 工具可帮你备份参数。

4.14 写入 IMU 标定参数

SDK 提供了写入 IMU 标定参数的工具 `imu_params_writer` 。工具详情可见 tools/README.md 。

有关如何获取，请阅读 获取 *IMU* 标定参数 。

参考运行命令：

```
./tools/_output/bin/writer/imu_params_writer tools/writer/config/imu.params

# Windows
.\tools\_output\bin\writer\imu_params_writer.bat tools\writer\config\imu.params
```

其中， tools/writer/config/imu.params 是参数文件路径。如果你自己标定了参数，可以编辑此文件，然后执行上述命令写入设备。

Warning: 请不要随意覆写参数。另外 `save_all_infos` 工具可帮你备份参数。

CHAPTER

FIVE

MYNT® EYE 控制

5.1 设定图像帧率和 IMU 频率

通过 API 的 `SetOptionValue()` 函数，就可以设定当前打开设备的各类控制值。

设定图像帧率和 IMU 频率，就是设定 `Option::FRAME_RATE` 和 `Option::IMU_FREQUENCY` 。

Attention:

- 图像帧率和 IMU 频率必须同时设定才能生效。
- 图像帧率有效值： 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 。
- IMU 频率有效值： 100, 200, 250, 333, 500 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// Attention: must set FRAME_RATE and IMU_FREQUENCY together, otherwise won't
// succeed.

// FRAME_RATE values: 10, 15, 20, 25, 30, 35, 40, 45, 50, 55
api->SetOptionValue(Option::FRAME_RATE, 25);
// IMU_FREQUENCY values: 100, 200, 250, 333, 500
api->SetOptionValue(Option::IMU_FREQUENCY, 500);

LOG(INFO) << "Set FRAME_RATE to " << api->GetOptionValue(Option::FRAME_RATE);
LOG(INFO) << "Set IMU_FREQUENCY to "
          << api->GetOptionValue(Option::IMU_FREQUENCY);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/ctrl_framerate
I0513 14:05:57.218222 31813 utils.cc:26] Detecting MYNT EYE devices
I0513 14:05:57.899404 31813 utils.cc:33] MYNT EYE devices:
I0513 14:05:57.899430 31813 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0513 14:05:57.899435 31813 utils.cc:43] Only one MYNT EYE device, select index: 0
I0513 14:05:58.076257 31813 framerate.cc:36] Set FRAME_RATE to 25
I0513 14:05:58.076836 31813 framerate.cc:37] Set IMU_FREQUENCY to 500
I0513 14:06:21.702361 31813 framerate.cc:82] Time beg: 2018-05-13 14:05:58.384967,␣
↪end: 2018-05-13 14:06:21.666115, cost: 23281.1ms
I0513 14:06:21.702388 31813 framerate.cc:85] Img count: 573, fps: 24.6122
I0513 14:06:21.702404 31813 framerate.cc:87] Imu count: 11509, hz: 494.348
```

样例程序按 ESC/Q 结束运行后，会输出计算得的图像帧率和 IMU 频率。

完整代码样例，请见 framerate.cc 。

5.2 设定加速度计及陀螺仪的量程

通过 API 的 SetOptionValue() 函数，就可以设定当前打开设备的各类控制值。

设定加速度计及陀螺仪的量程，就是设定 Option::ACCELEROMETER_RANGE 和 Option::GYROSCOPE_RANGE 。

Attention:

- 加速度计量程有效值（单位：g）： 4, 8, 16, 32 。
- 陀螺仪量程有效值（单位：deg/s）： 500, 1000, 2000, 4000 。

参考代码片段：

```
auto &&api = API::Create(argc, argv);
if (!api)
  return 1;

// ACCELEROMETER_RANGE values: 4, 8, 16, 32
api->SetOptionValue(Option::ACCELEROMETER_RANGE, 8);
// GYROSCOPE_RANGE values: 500, 1000, 2000, 4000
api->SetOptionValue(Option::GYROSCOPE_RANGE, 1000);

LOG(INFO) << "Set ACCELEROMETER_RANGE to "
          << api->GetOptionValue(Option::ACCELEROMETER_RANGE);
LOG(INFO) << "Set GYROSCOPE_RANGE to "
          << api->GetOptionValue(Option::GYROSCOPE_RANGE);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/ctrl_imu_range
I/utils.cc:28 Detecting MYNT EYE devices
I/utils.cc:38 MYNT EYE devices:
I/utils.cc:41   index: 0, name: MYNT-EYE-S1030, sn: 4B4C1F1100090712
I/utils.cc:49 Only one MYNT EYE device, select index: 0
I/imu_range.cc:34 Set ACCELEROMETER_RANGE to 8
I/imu_range.cc:36 Set GYROSCOPE_RANGE to 1000
I/imu_range.cc:81 Time beg: 2018-11-21 15:34:57.726428, end: 2018-11-21 15:35:12.
↪190478, cost: 14464ms
I/imu_range.cc:84 Img count: 363, fps: 25.0967
I/imu_range.cc:86 Imu count: 2825, hz: 195.312
```

样例程序按 ESC/Q 结束运行后，imu量程设置完成。该结果将固化在硬件内部，不受掉电影响。

完整代码样例，请见 imu_range.cc 。

5.3 启用自动曝光及其调节

通过 API 的 SetOptionValue() 函数，就可以设定当前打开设备的各类控制值。

启用自动曝光，就是设定 Option::EXPOSURE_MODE 为 0 。自动曝光时，可调节的设定有：

- Option::MAX_GAIN 最大增益。
- Option::MAX_EXPOSURE_TIME 最大曝光时间。
- Option::DESIRED_BRIGHTNESS 期望亮度。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// auto-exposure: 0
api->SetOptionValue(Option::EXPOSURE_MODE, 0);

// max_gain: range [0,48], default 48
api->SetOptionValue(Option::MAX_GAIN, 48);
// max_exposure_time: range [0,240], default 240
api->SetOptionValue(Option::MAX_EXPOSURE_TIME, 240);
// desired_brightness: range [0,255], default 192
api->SetOptionValue(Option::DESIRED_BRIGHTNESS, 192);

LOG(INFO) << "Enable auto-exposure";
LOG(INFO) << "Set MAX_GAIN to " << api->GetOptionValue(Option::MAX_GAIN);
LOG(INFO) << "Set MAX_EXPOSURE_TIME to "
          << api->GetOptionValue(Option::MAX_EXPOSURE_TIME);
LOG(INFO) << "Set DESIRED_BRIGHTNESS to "
          << api->GetOptionValue(Option::DESIRED_BRIGHTNESS);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/ctrl_auto_exposure
I0513 14:07:57.963943 31845 utils.cc:26] Detecting MYNT EYE devices
I0513 14:07:58.457536 31845 utils.cc:33] MYNT EYE devices:
I0513 14:07:58.457563 31845 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0513 14:07:58.457567 31845 utils.cc:43] Only one MYNT EYE device, select index: 0
I0513 14:07:58.474916 31845 auto_exposure.cc:37] Enable auto-exposure
I0513 14:07:58.491058 31845 auto_exposure.cc:38] Set MAX_GAIN to 48
I0513 14:07:58.505131 31845 auto_exposure.cc:39] Set MAX_EXPOSURE_TIME to 240
I0513 14:07:58.521375 31845 auto_exposure.cc:41] Set DESIRED_BRIGHTNESS to 192
```

样例程序会显示图像，左上角有真实曝光时间，单位毫秒。

完整代码样例，请见 auto_exposure.cc 。

5.4 启用手动曝光及其调节

通过 API 的 SetOptionValue() 函数，就可以设定当前打开设备的各类控制值。

启用手动曝光，就是设定 Option::EXPOSURE_MODE 为 1 。手动曝光时，可调节的设定有：

- Option::GAIN 增益。
- Option::BRIGHTNESS 亮度，或者说曝光时间。
- Option::CONTRAST 对比度，或者说黑电平校准。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// manual-exposure: 1
api->SetOptionValue(Option::EXPOSURE_MODE, 1);

// gain: range [0,48], default 24
api->SetOptionValue(Option::GAIN, 24);
// brightness/exposure_time: range [0,240], default 120
api->SetOptionValue(Option::BRIGHTNESS, 120);
// contrast/black_level_calibration: range [0,255], default 127
api->SetOptionValue(Option::CONTRAST, 127);

LOG(INFO) << "Enable manual-exposure";
LOG(INFO) << "Set GAIN to " << api->GetOptionValue(Option::GAIN);
LOG(INFO) << "Set BRIGHTNESS to " << api->GetOptionValue(Option::BRIGHTNESS);
LOG(INFO) << "Set CONTRAST to " << api->GetOptionValue(Option::CONTRAST);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/ctrl_manual_exposure
I0513 14:09:17.104431 31908 utils.cc:26] Detecting MYNT EYE devices
I0513 14:09:17.501519 31908 utils.cc:33] MYNT EYE devices:
I0513 14:09:17.501551 31908 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0513 14:09:17.501562 31908 utils.cc:43] Only one MYNT EYE device, select index: 0
I0513 14:09:17.552918 31908 manual_exposure.cc:37] Enable manual-exposure
I0513 14:09:17.552953 31908 manual_exposure.cc:38] Set GAIN to 24
I0513 14:09:17.552958 31908 manual_exposure.cc:39] Set BRIGHTNESS to 120
I0513 14:09:17.552963 31908 manual_exposure.cc:40] Set CONTRAST to 127
```

样例程序会显示图像，左上角有真实曝光时间，单位毫秒。

完整代码样例，请见 manual_exposure.cc 。

5.5 启用 IR 及其调节

通过 API 的 `SetOptionValue()` 函数，就可以设定当前打开设备的各类控制值。

启用 IR ，就是设定 `Option::IR_CONTROL` 大于 0 的值。值越大，强度越高。

参考代码片段：

```
auto &&api = API::Create(argc, argv);

// Detect infrared add-ons
LOG(INFO) << "Support infrared: " << std::boolalpha
          << api->Supports(AddOns::INFRARED);
LOG(INFO) << "Support infrared2: " << std::boolalpha
          << api->Supports(AddOns::INFRARED2);

// Get infrared intensity range
auto &&info = api->GetOptionInfo(Option::IR_CONTROL);
LOG(INFO) << Option::IR_CONTROL << ": {" << info << "}";

// Set infrared intensity value
api->SetOptionValue(Option::IR_CONTROL, 80);
```

参考运行结果，于 Linux 上：

```
$ ./samples/_output/bin/tutorials/ctrl_infrared
I0504 16:16:28.016624 25848 utils.cc:13] Detecting MYNT EYE devices
I0504 16:16:28.512462 25848 utils.cc:20] MYNT EYE devices:
I0504 16:16:28.512473 25848 utils.cc:24]   index: 0, name: MYNT-EYE-S1000
I0504 16:16:28.512477 25848 utils.cc:30] Only one MYNT EYE device, select index: 0
I0504 16:16:28.520848 25848 infrared.cc:13] Support infrared: true
I0504 16:16:28.520869 25848 infrared.cc:15] Support infrared2: true
I0504 16:16:28.520889 25848 infrared.cc:20] Option::IR_CONTROL: {min: 0, max: 160,␣
↪def: 0}
```

此时，如果显示了图像，就能够看到图像上会有 IR 光斑。

Attention: 硬件不会记忆 IR 值，断电会忘掉。如果需要保持启用 IR 的话，程序在打开设备后，一定要设定下 IR 值。

完整代码样例，请见 infrared.cc 。

CHAPTER

SIX

运行日志

6.1 启用日志文件

Tip: 如果引入 glog 库编译。

日志的通用配置，在头文件 logger.h 里 。

取消注释的 `FLAGS_log_dir = ".";` 重新编译，即可保存日志到当前工作目录。例如运行 `camera_a` 后，日志文件如下：

```
<workdir>/
├─camera_a.ERROR
├─camera_a.FATAL
├─camera_a.INFO
├─camera_a.WARNING
├─camera_a.john-ubuntu.john.log.ERROR.20180513-141833.519
├─camera_a.john-ubuntu.john.log.FATAL.20180513-141833.519
├─camera_a.john-ubuntu.john.log.INFO.20180513-141832.519
└─camera_a.john-ubuntu.john.log.WARNING.20180513-141833.519
```

`camera_a.INFO` 表明了哪个程序、什么日志级别。但它只是一个链接，指向真实的日志文件，如 `camera_a.john-ubuntu.john.log.INFO.20180513-141832.519` 。运行多次后， `camera_a.INFO` 仍只会有一个，指向最新的那个日志文件，便于你查看。

执行 `make cleanlog` 可以清理所有日志文件。

6.2 启用详细级别

Tip: 如果引入 glog 库编译。

日志的通用配置，在头文件 logger.h 里 。

取消注释的 `FLAGS_v = 2;` 重新编译，即可启用详细级别，指 `VLOG(n)` 打印的日志。

关于如何使用日志库，即如何配置、打印等，请如下打开其文档进行了解：

```
$ ./scripts/open.sh third_party/glog/doc/glog.html
```

CHAPTER

SEVEN

封装接口

7.1 ROS 如何使用

按照 *ROS 安装*，编译再运行节点。

`rostopic list` 可以列出发布的节点：

```
$ rostopic list
/mynteye/depth/image_raw
/mynteye/disparity/image_norm
/mynteye/disparity/image_raw
/mynteye/imu/data_raw
/mynteye/left/camera_info
/mynteye/left/image_raw
/mynteye/left/image_rect
/mynteye/points/data_raw
/mynteye/right/camera_info
/mynteye/right/image_raw
/mynteye/right/image_rect
/mynteye/temp/data_raw
...
```

`rostopic hz <topic>` 可以检查是否有数据：

```
$ rostopic hz /mynteye/imu/data_raw
subscribed to [/mynteye/imu/data_raw]
average rate: 505.953
  min: 0.000s max: 0.018s std dev: 0.00324s window: 478
average rate: 500.901
  min: 0.000s max: 0.018s std dev: 0.00327s window: 975
average rate: 500.375
  min: 0.000s max: 0.019s std dev: 0.00329s window: 1468
...
```

`rostopic echo <topic>` 可以打印发布数据等。了解更多，请阅读 rostopic 。

ROS 封装的文件结构，如下所示：

```
<sdk>/wrappers/ros/
├─src/
│   └─mynt_eye_ros_wrapper/
│       ├─launch/
│       │   ├─display.launch
│       │   └─mynteye.launch
```

(continues on next page)

(continued from previous page)

```
│       ├─msg/
│       ├─rviz/
│       ├─src/
│       │  ├─wrapper_node.cc
│       │  └─wrapper_nodelet.cc
│       ├─CMakeLists.txt
│       ├─nodelet_plugins.xml
│       └─package.xml
└─README.md
```

其中 `mynteye.launch` 里，可以配置发布的 topics 与 frame_ids 、决定启用哪些数据、以及设定控制选项。其中，`gravity` 请配置成当地重力加速度。

```
<arg name="gravity" default="9.8" />
```

如果想要打印调试信息，请编辑 `wrapper_node.cc` ，修改 `Info` 为 `Debug` 即可：

```
ros::console::set_logger_level(
    ROSCONSOLE_DEFAULT_NAME, ros::console::levels::Info);
```

CHAPTER

EIGHT

数据分析

8.1 录制数据集

SDK 提供了录制数据集的工具 `record`。工具详情可见 tools/README.md。

参考运行命令：

```
./tools/_output/bin/dataset/record

# Windows
.\tools\_output\bin\dataset\record.bat
```

参考运行结果，于 Linux 上：

```
$ ./tools/_output/bin/dataset/record
I0513 21:28:57.128947 11487 utils.cc:26] Detecting MYNT EYE devices
I0513 21:28:57.807116 11487 utils.cc:33] MYNT EYE devices:
I0513 21:28:57.807155 11487 utils.cc:37]   index: 0, name: MYNT-EYE-S1000
I0513 21:28:57.807163 11487 utils.cc:43] Only one MYNT EYE device, select index: 0
I0513 21:28:57.808437 11487 channels.cc:114] Option::GAIN: min=0, max=48, def=24,␣
↪cur=24
I0513 21:28:57.809999 11487 channels.cc:114] Option::BRIGHTNESS: min=0, max=240,␣
↪def=120, cur=120
I0513 21:28:57.818678 11487 channels.cc:114] Option::CONTRAST: min=0, max=255,␣
↪def=127, cur=127
I0513 21:28:57.831529 11487 channels.cc:114] Option::FRAME_RATE: min=10, max=60,␣
↪def=25, cur=25
I0513 21:28:57.848914 11487 channels.cc:114] Option::IMU_FREQUENCY: min=100, max=500,␣
↪def=200, cur=500
I0513 21:28:57.865185 11487 channels.cc:114] Option::EXPOSURE_MODE: min=0, max=1,␣
↪def=0, cur=0
I0513 21:28:57.881434 11487 channels.cc:114] Option::MAX_GAIN: min=0, max=48, def=48,␣
↪cur=48
I0513 21:28:57.897598 11487 channels.cc:114] Option::MAX_EXPOSURE_TIME: min=0,␣
↪max=240, def=240, cur=240
I0513 21:28:57.913918 11487 channels.cc:114] Option::DESIRED_BRIGHTNESS: min=0,␣
↪max=255, def=192, cur=192
I0513 21:28:57.930177 11487 channels.cc:114] Option::IR_CONTROL: min=0, max=160,␣
↪def=0, cur=0
I0513 21:28:57.946341 11487 channels.cc:114] Option::HDR_MODE: min=0, max=1, def=0,␣
↪cur=0
Saved 1007 imgs, 20040 imus to ./dataset
I0513 21:29:38.608772 11487 record.cc:118] Time beg: 2018-05-13 21:28:58.255395, end:␣
↪2018-05-13 21:29:38.578696, cost: 40323.3ms
```

(continues on next page)

(continued from previous page)

```
I0513 21:29:38.608853 11487 record.cc:121] Img count: 1007, fps: 24.9732
I0513 21:29:38.608873 11487 record.cc:123] Imu count: 20040, hz: 496.983
```

默认录制进 <workdir>/dataset 目录。你也可以加参数，指定录制到其他目录。

录制内容如下：

```
<workdir>/
└─dataset/
    ├─left/
    │   ├─stream.txt  # Image infomation
    │   ├─000000.png  # Image, index 0
    │   └─...
    ├─right/
    │   ├─stream.txt  # Image information
    │   ├─000000.png  # Image, index 0
    │   └─...
    └─motion.txt  # IMU information
```

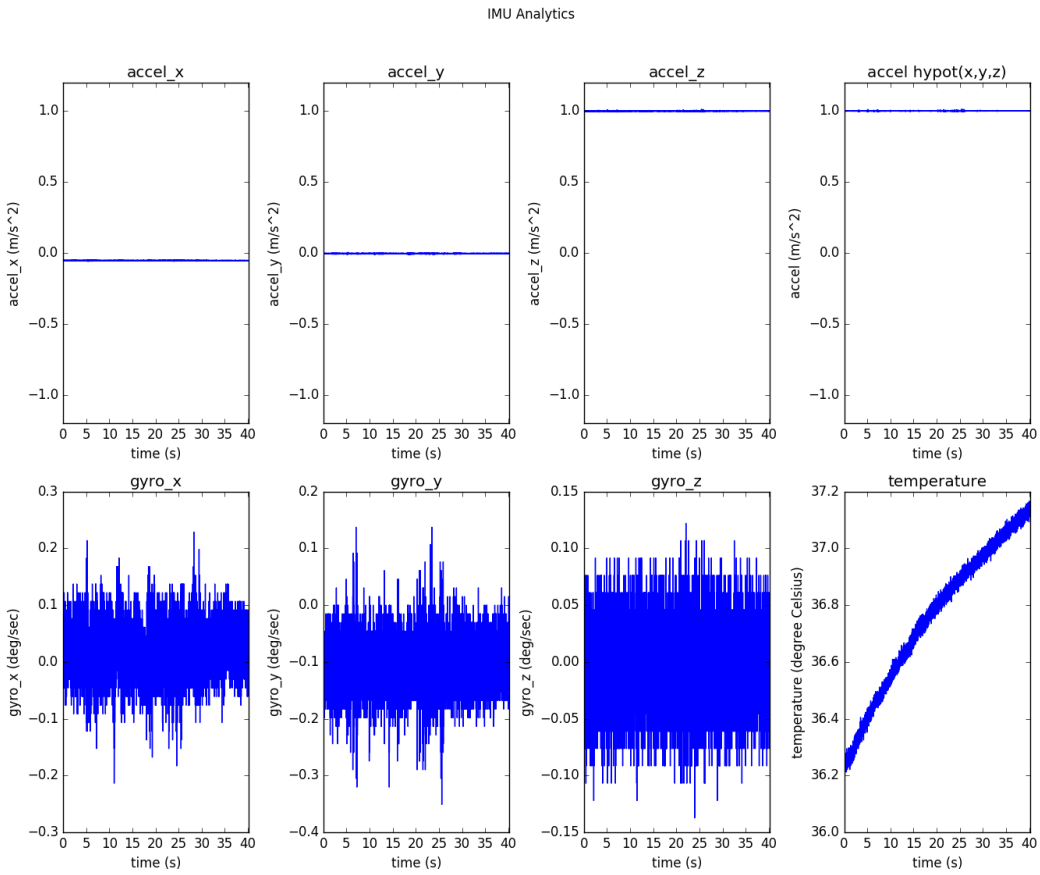
8.2 分析 IMU

SDK 提供了 IMU 分析的脚本 imu_analytics.py 。工具详情可见 tools/README.md 。

参考运行命令及结果，于 Linux 上：

```
$ python tools/analytics/imu_analytics.py -i dataset -c tools/config/mynteye/mynteye_
↪config.yaml -al=-1.2,1.2 -gl= -gdu=d -gsu=d -kl=
imu analytics ...
  input: dataset
  outdir: dataset
  gyro_limits: None
  accel_limits: [(-1.2, 1.2), (-1.2, 1.2), (-1.2, 1.2), (-1.2, 1.2)]
  time_unit: None
  time_limits: None
  auto: False
  gyro_show_unit: d
  gyro_data_unit: d
  temp_limits: None
open dataset ...
  imu: 20040, temp: 20040
  timebeg: 4.384450, timeend: 44.615550, duration: 40.231100
save figure to:
  dataset/imu_analytics.png
imu analytics done
```

分析结果图会保存进数据集目录，参考如下：



另外，脚本具体选项可执行 `-h` 了解:

```
$ python tools/analytics/imu_analytics.py -h
```

8.3 分析时间戳

SDK 提供了时间戳分析的脚本 `stamp_analytics.py` 。工具详情可见 tools/README.md 。

参考运行命令及结果，于 Linux 上:

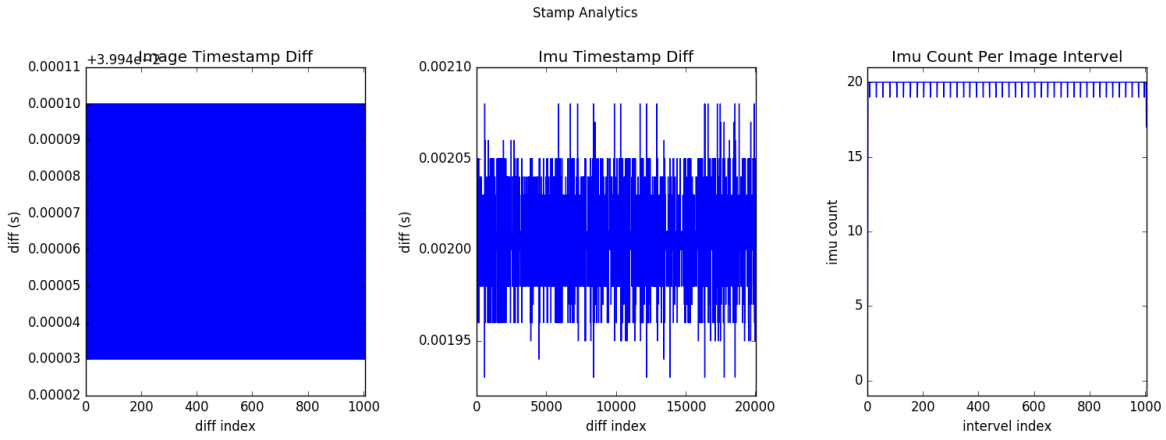
```
$ python tools/analytics/stamp_analytics.py -i dataset -c tools/config/mynteye/
↪mynteye_config.yaml
stamp analytics ...
  input: dataset
  outdir: dataset
open dataset ...
save to binary files ...
  binimg: dataset/stamp_analytics_img.bin
  binimu: dataset/stamp_analytics_imu.bin
  img: 1007, imu: 20040

rate (Hz)
  img: 25, imu: 500
```

(continues on next page)

(continued from previous page)

```
sample period (s)
  img: 0.04, imu: 0.002

diff count
  imgs: 1007, imus: 20040
  imgs_t_diff: 1006, imus_t_diff: 20039

diff where (factor=0.1)
  imgs where diff > 0.04*1.1 (0)
  imgs where diff < 0.04*0.9 (0)
  imus where diff > 0.002*1.1 (0)
  imus where diff < 0.002*0.9 (0)

image timestamp duplicates: 0

save figure to:
  dataset/stamp_analytics.png
stamp analytics done
```

分析结果图会保存进数据集目录，参考如下：



另外，脚本具体选项可执行 `-h` 了解：

```
$ python tools/analytics/stamp_analytics.py -h
```

Tip: 录制数据集时建议 `record.cc` 里注释显示图像 `cv::imshow()`， `dataset.cc` 里注释存储图像 `cv::imwrite()` 。因为此些操作都比较耗时，可能会导致丢弃图像。换句话说就是消费赶不上生产，所以丢弃了部分图像。 `record.cc` 里用的 `GetStreamDatas()` 仅缓存最新的 4 张图像。

CHAPTER

NINE

SLAM

9.1 VINS-Mono 如何整合

9.1.1 在 MYNT® EYE 上运行 VINS-Mono，请依照这些步骤:

1. 下载 MYNT-EYE-S-SDK 及安装 mynt_eye_ros_wrapper。
2. 按照一般步骤安装 VINS-Mono 。
3. 在 这里 更新 `distortion_parameters` 和 `projection_parameters` 参数。
4. 运行 mynt_eye_ros_wrapper 和 VINS-Mono 。

9.1.2 快捷安装 ROS Kinetic (若已安装，请忽略)

```
cd ~
wget https://raw.githubusercontent.com/oroca/oroca-ros-pkg/master/ros_install.sh && \
chmod 755 ./ros_install.sh && bash ./ros_install.sh catkin_ws kinetic
```

9.1.3 安装 MYNT-EYE-VINS-Sample

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
git clone -b mynteye-s https://github.com/slightech/MYNT-EYE-VINS-Sample.git
cd ..
catkin_make
source devel/setup.bash
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

9.1.4 获取图像校准参数

使用 MYNT® EYE 的左目摄像头和 IMU 。通过 MYNT-EYE-S-SDK API的 `GetIntrinsics()` 函数和 `GetExtrinsics()` 函数，可以获得当前工作设备的图像校准参数:

```
cd MYNT-EYE-S-SDK
./samples/_output/bin/tutorials/get_img_params
```

这时，可以获得针孔模型下的 distortion_parameters 和 projection_parameters 参数，然后在这里 更新。

9.1.5 在 MYNT® EYE 上运行 VINS-Mono

```
cd ~/catkin_ws
roslaunch mynt_eye_ros_wrapper mynteye.launch
roslaunch vins_estimator mynteye.launch
```

Note: 如果使用鱼眼相机模型，点击 这里 。

9.2 ORB_SLAM2 如何整合

9.2.1 在 MYNT® EYE 上运行 ORB_SLAM2 ，请依照这些步骤：

1. 下载 MYNT-EYE-S-SDK 及安装。
2. 按照一般步骤安装 ORB_SLAM2 。
3. 更新 distortion_parameters 和 projection_parameters 参数到 <ORB_SLAM2>/config/mynteye_*.yaml。
4. 在 MYNT® EYE 上运行例子。

9.2.2 双目样例

- 按照 ROS-StereoCalibration 或 OpenCV 校准双目摄像头，并在 <ORB_SLAM2>/config/mynteye_stereo.yaml 中更新参数。
- 运行脚本 build.sh：

```
chmod +x build.sh
./build.sh
```

- 依照下面样式运行双目样例：

```
./Examples/Stereo/stereo_mynt ./Vocabulary/ORBvoc.txt ./config/mynteye_stereo.yaml␣
↪true /mynteye/left/image_raw /mynteye/right/image_raw
```

9.2.3 ROS 下创建单目和双目节点

- 添加 Examples/ROS/ORB_SLAM2 路径到环境变量 ROS_PACKAGE_PATH 。打开 .bashrc 文件，在最后添加下面命令行。 PATH 为当前 ORB_SLAM2 存放路径:

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:PATH/ORB_SLAM2/Examples/ROS
```

- 运行脚本 *build_ros.sh* ：

```
chmod +x build_ros.sh
./build_ros.sh
```

Mono_ROS 例子

- 在 <ORBSLAM2>/config/mynteye_mono.yaml 里更新 distortion_parameters 和 projection_parameters 参数

```
cd MYNT-EYE-S-SDK

./samples/_output/bin/tutorials/get_img_params
```

这时获得针孔模型下 distortion_parameters 和 projection_parameters 参数，更新到 <ORBSLAM2>/config/mynteye_mono.yaml 中。

- 运行 ORB_SLAM2 Mono_ROS 例子

```
rosrun ORB_SLAM2 mynteye_mono ./Vocabulary/ORBvoc.txt ./config/mynteye_mono.yaml /
↪mynteye/left/image_raw
```

Stereo_ROS 例子

- 按照 ROS-StereoCalibration 或 OpenCV 校准双目摄像头，并在 <ORB_SLAM2>/config/mynteye_stereo.yaml 中更新参数。
- 运行 ORB_SLAM2 Stereo_ROS 例子

```
rosrun ORB_SLAM2 ros_mynteye_stereo ./Vocabulary/ORBvoc.txt ./config/mynteye_stereo.
↪yaml true /mynteye/left/image_raw /mynteye/right/image_raw
```

9.3 OKVIS 如何整合

9.3.1 在 MYNT® EYE 上运行 OKVIS，请依照这些步骤:

1. 下载 MYNT-EYE-S-SDK 并安装。
2. 安装依赖，按照原始 OKVIS 步骤安装 MYNT-EYE-OKVIS-Sample 。
3. 更新相机参数到 <OKVIS>/config/config_mynteye.yaml。
4. 在 MYNT® EYE 上运行 OKVIS 。

9.3.2 安装 MYNT® EYE OKVIS

首先安装原始 OKVIS 及依赖:

```
git clone -b mynteye-s https://github.com/slightech/MYNT-EYE-OKVIS-Sample.git
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j4
```

9.3.3 获取相机校准参数

通过 MYNT-EYE-S-SDK API 的 GetIntrinsics() 函数和 GetExtrinsics() 函数，可以获得当前工作设备的图像校准参数：

```
cd MYNT-EYE-S-SDK
./samples/_output/bin/tutorials/get_img_params
```

这时，可以获得针孔模型下的 distortion_parameters 和 projection_parameters 参数，然后在这里 更新。

9.3.4 运行 MYNT® EYE OKVIS

在 MYNT-EYE-OKVIS-Sample/build 中运行 okvis_app_mynteye_s：

```
cd MYNT-EYE-OKVIS-Sample/build
./okvis_app_mynteye_s ../config/config_mynteye.yaml
```

9.4 VIORB 如何整合

9.4.1 在 MYNT® EYE 上运行 VIORB，请依照这些步骤：

1. 下载 MYNT-EYE-S-SDK， 安装 mynt_eye_ros_wrapper。
2. 按照一般步骤安装 VIORB 。
3. 更新相机参数到 <VIO>/config/config_mynteye.yaml。
4. 运行 mynt_eye_ros_wrapper 和 VIORB 。

9.4.2 安装 MYNT-EYE-VIORB-Sample.

```
git clone -b mynteye-s https://github.com/slightech/MYNT-EYE-VIORB-Sample.git
cd MYNT-EYE-VIORB-Sample
```

添加 Examples/ROS/ORB_VIO 路径到环境变量 ROS_PACKAGE_PATH 。打开 .bashrc 文件，在最后添加下面命令行。 PATH 为当前 MYNT-EYE-VIORB-Sample. 存放路径：

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:PATH/Examples/ROS/ORB_VIO
```

执行：

```
cd MYNT-EYE-VIORB-Sample
./build.sh
```

9.4.3 获取相机校准参数

使用 MYNT® EYE 的左目摄像头和 IMU 。通过 MYNT-EYE-S-SDK API 的 GetIntrinsics() 函数和 GetExtrinsics() 函数，可以获得当前工作设备的图像校准参数：

```
cd MYNT-EYE-S-SDK
./samples/_output/bin/tutorials/get_img_params
```

这时，可以获得针孔模型下的 distortion_parameters 和 projection_parameters 参数，然后在 <MYNT-EYE-VIORB-Sample>/config/mynteye.yaml 中更新。

9.4.4 运行 VIORB 和 mynt_eye_ros_wrapper

```
roslaunch mynt_eye_ros_wrapper mynteye.launch
roslaunch ORB_VIO testmynteye.launch
```

最后，pyplotscripts 下的脚本会将结果可视化。

9.5 Maplab 如何整合 x