OpenVSLAM

Shinya Sumikura

Feb 20, 2022
### GETTING STARTED

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Installation</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Tutorial</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Reference</td>
<td>4</td>
</tr>
<tr>
<td>Installation</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Source code</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Dependencies</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Prerequisites for Unix</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Build Instructions</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Server Setup for SocketViewer</td>
<td>11</td>
</tr>
<tr>
<td>Simple Tutorial</td>
<td>13</td>
</tr>
<tr>
<td>3.1 TL; DR</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Sample Datasets</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Tracking and Mapping</td>
<td>15</td>
</tr>
<tr>
<td>3.4 Localization</td>
<td>18</td>
</tr>
<tr>
<td>Running on Docker</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Instructions for PangolinViewer</td>
<td>21</td>
</tr>
<tr>
<td>4.2 Instructions for SocketViewer</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Bind of Directories</td>
<td>24</td>
</tr>
<tr>
<td>Running on ROS</td>
<td>25</td>
</tr>
<tr>
<td>5.1 ROS Package</td>
<td>25</td>
</tr>
<tr>
<td>5.2 ROS2 Package</td>
<td>27</td>
</tr>
<tr>
<td>Example</td>
<td>31</td>
</tr>
<tr>
<td>6.1 SLAM with Video Files</td>
<td>31</td>
</tr>
<tr>
<td>6.2 SLAM with Image Sequences</td>
<td>32</td>
</tr>
<tr>
<td>6.3 SLAM with Standard Datasets</td>
<td>33</td>
</tr>
<tr>
<td>6.4 SLAM with UVC camera</td>
<td>37</td>
</tr>
<tr>
<td>Parameters</td>
<td>39</td>
</tr>
<tr>
<td>7.1 Camera</td>
<td>39</td>
</tr>
<tr>
<td>7.2 Feature</td>
<td>39</td>
</tr>
<tr>
<td>7.3 Preprocessing</td>
<td>40</td>
</tr>
<tr>
<td>7.4 Tracking</td>
<td>40</td>
</tr>
<tr>
<td>7.5 Mapping</td>
<td>40</td>
</tr>
<tr>
<td>7.6 StereoRectifier</td>
<td>40</td>
</tr>
<tr>
<td>7.7 Initializer</td>
<td>41</td>
</tr>
</tbody>
</table>
This is the OpenVSLAM documentation.
OpenVSLAM is a monocular, stereo, and RGBD visual SLAM system. The notable features are:

- It is compatible with various type of camera models and can be easily customized for other camera models.
- Created maps can be stored and loaded, then OpenVSLAM can localize new images based on the prebuilt maps.
- The system is fully modular. It is designed by encapsulating several functions in separated components with easy-to-understand APIs.
- We provided some code snippets to understand the core functionalities of this system.

OpenVSLAM is based on an indirect SLAM algorithm with sparse features, such as ORB-SLAM, ProSLAM, and UcoSLAM. One of the noteworthy features of OpenVSLAM is that the system can deal with various type of camera models, such as perspective, fisheye, and equirectangular. If needed, users can implement extra camera models (e.g. dual fisheye, catadioptric) with ease. For example, visual SLAM algorithm using equirectangular camera models (e.g. RICOH THETA series, insta360 series, etc) is shown above.

Some code snippets to understand the core functionalities of the system are provided. You can employ these snippets for in your own programs. Please see the *.cc files in ./example directory or check Simple Tutorial and Example. Also, some examples to run OpenVSLAM on ROS framework are provided. Please check ROS Package.

Please contact us via GitHub issues if you have any questions or notice any bugs about the software.
1.1 Installation

Please see *Installation* chapter.

*The instructions for Docker users* are also provided.

1.2 Tutorial

Please see *Simple Tutorial*.

A sample ORB vocabulary file can be downloaded from here.
Sample datasets are also provided at here.

If you would like to run visual SLAM with standard benchmarking datasets (e.g. KITTI Odometry dataset), please see *SLAM with standard datasets*.

1.3 Reference


2.1 Source code

The source code can be viewed from this GitHub repository.

Cloning the repository:

```bash
git clone https://github.com/OpenVSLAM-Community/openvslam.git
cd openvslam
git submodule update -i --recursive
```

If you are a Windows 10 user, please install the dependencies and OpenVSLAM with SocketViewer support on Windows Subsystem for Linux (WSL). We have checked the correct operation of OpenVSLAM and SocketViewer on Ubuntu 16.04 running on WSL.

Docker systems can be used instead of preparing the dependencies manually.

2.2 Dependencies

OpenVSLAM requires a C++11-compliant compiler. It relies on several open-source libraries as shown below.

2.2.1 Requirements for OpenVSLAM

- **Eigen**: version 3.3.0 or later.
- **g2o**: Please use the latest release. Tested on commit ID 9b41a4e.
- **SuiteSparse**: Required by g2o.
- **FBoW**: Please use the custom version of FBoW released in https://github.com/OpenVSLAM-Community/FBoW.
- **yaml-cpp**: version 0.6.0 or later.
- **OpenCV**: version 3.3.1 or later.

**Note:** OpenCV with GUI support is necessary for using the built-in viewer (Pangolin Viewer).

**Note:** OpenCV with video support is necessary if you plan on using video files (e.g. .mp4) as inputs.
2.2.2 Requirements for PangolinViewer

We provided an OpenGL-based simple viewer. This viewer is implemented with Pangolin. Thus, we call it PangolinViewer. Please install the following dependencies if you plan on using PangolinViewer.

- Pangolin: Please use the latest release. Tested on commit ID ad8b5f8.
- GLEW: Required by Pangolin.

2.2.3 Requirements for SocketViewer

We provided a WebGL-based simple viewer running on web browsers. The SLAM systems publish the map and the frame to the server implemented with Node.js via WebSocket. Thus, we call it SocketViewer. Please install the following dependencies if you plan on using SocketViewer.

- Protobuf: version 3 or later.

The following libraries are the dependencies for the server.

- Node.js: version 6 or later.
- npm: Tested on version 3.5.2.

2.2.4 Recommended

- backward-cpp: Used for stack-trace logger.

2.3 Prerequisites for Unix

Note: In the following instruction, we assume that CMAKE_INSTALL_PREFIX is /usr/local. If you want to install the libraries to the different location, set CMAKE_INSTALL_PREFIX to your environment and set the environment variables accordingly.

Note: If your PC is frozen during the build, please reduce the number of parallel compile jobs when executing make (e.g. make -j2).
2.3.1 Installing for Linux

Tested for Ubuntu 18.04.

Install the dependencies via `apt`.

```
atp update -y
apt upgrade -y --no-install-recommends
# basic dependencies
apt install -y build-essential pkg-config cmake git wget curl unzip
# g2o dependencies
apt install -y libatlas-base-dev libsuitesparse-dev
# OpenCV dependencies
apt install -y libgdk-3-dev
apt install -y ffmpeg
apt install -y libavcodec-dev libavformat-dev libavutil-dev libswscale-dev libavresample-dev
# eigen dependencies
apt install -y gfortran
# backward-cpp dependencies (optional)
apt install -y binutils-dev
# other dependencies
apt install -y libyaml-cpp-dev libgflags-dev
# (if you plan on using PangolinViewer)
# Pangolin dependencies
apt install -y libgglew-dev
# (if you plan on using SocketViewer)
# Protobuf dependencies
apt install -y autogen autoconf libtool
# Node.js
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
apt install -y nodejs
```

Download and install Eigen from source.

```
cd /path/to/working/dir
wget -q https://gitlab.com/libeigen/eigen/-/archive/3.3.7/eigen-3.3.7.tar.bz2
tar xf eigen-3.3.7.tar.bz2
rm -rf eigen-3.3.7.tar.bz2
cd eigen-3.3.7
mkdir -p build & cd build
make -j4
make install
```

Download, build and install OpenCV from source.

```
cd /path/to/working/dir
wget -q https://github.com/opencv/opencv/archive/3.4.0.zip
```

(continues on next page)
unzip -q 3.4.0.zip
rm -rf 3.4.0.zip
cd opencv-3.4.0
mkdir -p build && cd build
cmake
    -DCMAKE_BUILD_TYPE=Release
    -DCMAKE_INSTALL_PREFIX=/usr/local
    -DENABLE_CXX11=ON
    -DBUILD_DOCS=OFF
    -DBUILD_EXAMPLES=OFF
    -DBUILD_JASPER=OFF
    -DBUILD_OPENEXR=OFF
    -DBUILD_PERF_TESTS=OFF
    -DBUILD_TESTS=OFF
    -DWITH_EIGEN=ON
    -DWITH_FFMPEG=ON
    -DWITH_OPENMP=ON
.. make -j4
make install

Jump to Common Installation Instructions for the next step.

### 2.3.2 Installing for macOS

Tested for macOS High Sierra.

Install the dependencies via brew.

```
brew update
# basic dependencies
brew install pkg-config cmake git
# g2o dependencies
brew install suite-sparse
# OpenCV dependencies and OpenCV
brew install eigen
brew install ffmpeg
brew install opencv
# other dependencies
brew install yaml-cpp glog gflags

# (if you plan on using PangolinViewer)
# Pangolin dependencies
brew install glew

# (if you plan on using SocketViewer)
# Protobuf dependencies
brew install automake autoconf libtool
# Node.js
brew install node
```

Jump to Common Installation Instructions for the next step.
### 2.3.3 Common Installation Instructions

Download, build and install the custom FBoW from source.

```bash
cd /path/to/working/dir
git clone https://github.com/OpenVSLAM-Community/FBoW.git
cd FBoW
mkdir build && cd build
  cmake 
    -DCMAKE_BUILD_TYPE=Release 
    -DCMAKE_INSTALL_PREFIX=/usr/local 
    ...
  make -j4
make install
```

Download, build and install g2o.

```bash
cd /path/to/working/dir
git clone https://github.com/RainerKuemmerle/g2o.git
cd g2o
git checkout 9b41a4ea5ade8e1250b9c1b279f3a9c098811b5a
mkdir build && cd build
cmake 
  -DCMAKE_BUILD_TYPE=Release 
  -DCMAKE_INSTALL_PREFIX=/usr/local 
  -DCMAKE_CXX_FLAGS=-std=c++11 
  -DBUILD_SHARED_LIBS=ON 
  -DBUILD_UNITTESTS=OFF 
  -DG2O_USE_CHOLMOD=OFF 
  -DG2O_USE_CSPARSE=ON 
  -DG2O_USE_OPENGL=OFF 
  -DG2O_USE_OPENMP=ON 
  ...
  make -j4
make install
```

(if you plan on using PangolinViewer)

Download, build and install Pangolin from source.

```bash
cd /path/to/working/dir
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
git checkout ad8b5f83222291c51b4800d5a5873b0e90a0cf81
mkdir build && cd build
cmake 
  -DCMAKE_BUILD_TYPE=Release 
  -DCMAKE_INSTALL_PREFIX=/usr/local 
  ...
  make -j4
make install
```
(if you plan on using SocketViewer)

Download, build and install socket.io-client-cpp from source.

```
cd /path/to/working/dir
git clone https://github.com/shinsumicco/socket.io-client-cpp.git
cd socket.io-client-cpp
git submodule init
git submodule update
mkdir build && cd build

cmake \
   -DCMAKE_BUILD_TYPE=Release \n   -DCMAKE_INSTALL_PREFIX=/usr/local \n   -DBUILD_UNIT_TESTS=OFF \n   ...
make -j4
make install
```

(if you plan on using SocketViewer)

Install Protobuf.

If you use Ubuntu 18.04 or macOS, Protobuf 3.x can be installed via apt or brew.

```
# for Ubuntu 18.04 (or later)
apt install -y libprotobuf-dev protobuf-compiler
# for macOS
brew install protobuf
```

Otherwise, please download, build and install Protobuf from source.

```
wget -q https://github.com/google/protobuf/archive/v3.6.1.tar.gz
tar xf v3.6.1.tar.gz
cd protobuf-3.6.1
./autogen.sh
./configure \
   --prefix=/usr/local \
   --enable-static=no
make -j4
make install
```

### 2.4 Build Instructions

When building with support for PangolinViewer, please specify the following cmake options: 
-DUSE_PANGOLIN_VIEWER=ON and -DUSE_SOCKET PUBLISHER=OFF.

```
cd /path/to/openvslam
mkdir build && cd build

cmake \
(continues on next page)
```
When building with support for SocketViewer, please specify the following cmake options:
- `DUSE_PANGOLIN_VIEWER=OFF` and `DUSE_SOCKET_PUBLISHER=ON`.

```bash
cd /path/to/openvslam
mkdir build && cd build
make \
-DUSE_PANGOLIN_VIEWER=OFF \
-DUSE_SOCKET_PUBLISHER=ON \
-DINSTALL_SOCKET_PUBLISHER=ON \
-DBUILD_TESTS=ON \
-DBUILD_EXAMPLES=ON \
.. 
made -j4
make install
```

After building, check to see if it was successfully built by executing `./run_kitti_slam -h`.

```bash
$ ./run_kitti_slam -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-d, --data-dir arg directory path which contains dataset
-c, --config arg config file path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
--eval-log store trajectory and tracking times for evaluation
-p, --map-db arg store a map database at this path after SLAM
```

### 2.5 Server Setup for SocketViewer

If you plan on using SocketViewer, please setup the environment for the server with npm.

```bash
$ cd /path/to/openvslam/viewer
$ ls
Dockerfile app.js package.json public views
$ npm install
added 88 packages from 60 contributors and audited 204 packages in 2.105s
found 0 vulnerabilities
$ ls
Dockerfile app.js node_modules package-lock.json package.json public views
```

---

**2.5. Server Setup for SocketViewer**
Then, launch the server with `node app.js`.

```bash
$ cd /path/to/openvslam/viewer
$ ls
Dockerfile app.js node_modules package-lock.json package.json public views
$ node app.js
WebSocket: listening on *:3000
HTTP server: listening on *:3001
```

After launching, please access to `http://localhost:3001/` to check whether the server is correctly launched.

---

**Note:** When you try *the tutorial* and *the examples* with SocketViewer, please launch the server in the other terminal and access to it with the web browser **in advance.**
CHAPTER
THREE

SIMPLE TUTORIAL

3.1 TL; DR

Note: If you use SocketViewer, please launch the server in the other terminal and access to it with the web browser in advance.

Running the following commands will give a feel for what OpenVSLAM can do. The later parts of this chapter explains what each of the commands do in more detail.

```
# at the build directory of openvslam ...
$ pwd
/path/to/openvslam/build/
$ ls
run_video_slam  run_video_localization  lib/  ...

# download an ORB vocabulary from GitHub
curl -sL "https://github.com/OpenVSLAM-Community/FBoW_orb_vocab/raw/main/orb_vocab.fbow" -o orb_vocab.fbow

# download a sample dataset from Google Drive
FILE_ID="1d8kADKWbPtEqTF7jeVhKat8EdN7g0ikY"
FILE_ID="1TVf2D2QvMZPHsFoTb7HNxbXc1PoFMGLX"
FILE_ID="1d8kADKWbPtEqTF7jeVhKat8EdN7g0ikY"
curl -sc /tmp/cookie "https://drive.google.com/uc?export=download&id=${FILE_ID}" > /dev/null
CODE="$(awk '/_warning_/ {print $NF}' /tmp/cookie)"
curl -sLb /tmp/cookie "https://drive.google.com/uc?export=download&confirm=${CODE}&id=${FILE_ID}" -o aist_living_lab_1.zip
unzip aist_living_lab_1.zip
unzip aist_living_lab_2.zip

# run tracking and mapping
./run_video_slam -v ./orb_vocab.fbow -m ./aist_living_lab_1/video.mp4 -c ../example/aist/equirectangular.yaml --frame-skip 3 --no-sleep --map-db map.msg
```

(continues on next page)
# click the [Terminate] button to close the viewer
# you can find map.msg in the current directory

# run localization
./run_video_localization -v ./orb_vocab.fbow -m ./aist_living_lab_2/video.mp4 -c ../
˓→ example/aist/equirectangular.yaml --frame-skip 3 --no-sleep --map-db map.msg

3.2 Sample Datasets

You can use OpenVSLAM with various video datasets. If you want to run OpenVSLAM with standard benchmarking datasets, please see this section.

Start by downloading some datasets you like.

3.2.1 Equirectangular Datasets

<table>
<thead>
<tr>
<th>name</th>
<th>camera model</th>
<th>length</th>
<th>Google Drive</th>
<th>Baidu Wangpan</th>
</tr>
</thead>
<tbody>
<tr>
<td>aist_entrance_hall_1</td>
<td>equirectangular (mono)</td>
<td>0:54</td>
<td>link</td>
<td>link (Pass: r7r4)</td>
</tr>
<tr>
<td>aist_entrance_hall_2</td>
<td>equirectangular (mono)</td>
<td>0:54</td>
<td>link</td>
<td>link (Pass: 4qna)</td>
</tr>
<tr>
<td>aist_factory_A_1</td>
<td>equirectangular (mono)</td>
<td>1:55</td>
<td>link</td>
<td>link (Pass: yy2u)</td>
</tr>
<tr>
<td>aist_factory_A_2</td>
<td>equirectangular (mono)</td>
<td>1:54</td>
<td>link</td>
<td>link (Pass: 9vey)</td>
</tr>
<tr>
<td>aist_factory_B_1</td>
<td>equirectangular (mono)</td>
<td>1:04</td>
<td>link</td>
<td>link (Pass: gpec)</td>
</tr>
<tr>
<td>aist_factory_B_2</td>
<td>equirectangular (mono)</td>
<td>1:34</td>
<td>link</td>
<td>link (Pass: ugrx)</td>
</tr>
<tr>
<td>aist_living_lab_1</td>
<td>equirectangular (mono)</td>
<td>2:16</td>
<td>link</td>
<td>link (Pass: 434m)</td>
</tr>
<tr>
<td>aist_living_lab_2</td>
<td>equirectangular (mono)</td>
<td>1:47</td>
<td>link</td>
<td>link (Pass: 549f)</td>
</tr>
<tr>
<td>aist_living_lab_3</td>
<td>equirectangular (mono)</td>
<td>2:06</td>
<td>link</td>
<td>link (Pass: cc2p)</td>
</tr>
<tr>
<td>aist_stairs_A_1</td>
<td>equirectangular (mono)</td>
<td>2:27</td>
<td>link</td>
<td>link (Pass: ncdr)</td>
</tr>
<tr>
<td>aist_stairs_B_1</td>
<td>equirectangular (mono)</td>
<td>2:55</td>
<td>link</td>
<td>link (Pass: x5r)</td>
</tr>
<tr>
<td>aist_store_1</td>
<td>equirectangular (mono)</td>
<td>1:12</td>
<td>link</td>
<td>link (Pass: 47vq)</td>
</tr>
<tr>
<td>aist_store_2</td>
<td>equirectangular (mono)</td>
<td>1:44</td>
<td>link</td>
<td>link (Pass: xt8u)</td>
</tr>
<tr>
<td>aist_store_3</td>
<td>equirectangular (mono)</td>
<td>1:18</td>
<td>link</td>
<td>link (Pass: kghe)</td>
</tr>
<tr>
<td>ALL</td>
<td>equirectangular (mono)</td>
<td></td>
<td>link</td>
<td>link (Pass: ssv7)</td>
</tr>
</tbody>
</table>

3.2.2 Fisheye Datasets

<table>
<thead>
<tr>
<th>name</th>
<th>camera model</th>
<th>length</th>
<th>Google Drive</th>
<th>Baidu Wangpan</th>
</tr>
</thead>
<tbody>
<tr>
<td>aist_entrance_hall_1</td>
<td>fisheye (mono)</td>
<td>1:05</td>
<td>link</td>
<td>link (Pass: u86e)</td>
</tr>
<tr>
<td>aist_entrance_hall_2</td>
<td>fisheye (mono)</td>
<td>1:06</td>
<td>link</td>
<td>link (Pass: 9iyc)</td>
</tr>
<tr>
<td>aist_entrance_hall_3</td>
<td>fisheye (mono)</td>
<td>1:23</td>
<td>link</td>
<td>link (Pass: qaqc)</td>
</tr>
<tr>
<td>aist_entrance_hall_4</td>
<td>fisheye (mono)</td>
<td>1:27</td>
<td>link</td>
<td>link (Pass: em43)</td>
</tr>
<tr>
<td>aist_living_lab_1</td>
<td>fisheye (mono)</td>
<td>1:20</td>
<td>link</td>
<td>link (Pass: wcw4)</td>
</tr>
<tr>
<td>aist_living_lab_2</td>
<td>fisheye (mono)</td>
<td>2:26</td>
<td>link</td>
<td>link (Pass: dxns)</td>
</tr>
<tr>
<td>aist_living_lab_3</td>
<td>fisheye (mono)</td>
<td>3:43</td>
<td>link</td>
<td>link (Pass: 7nq4)</td>
</tr>
<tr>
<td>nu_eng2_corridor_1</td>
<td>fisheye (mono)</td>
<td>2:56</td>
<td>link</td>
<td>link (Pass: 71ws)</td>
</tr>
<tr>
<td>nu_eng2_corridor_2</td>
<td>fisheye (mono)</td>
<td>2:45</td>
<td>link</td>
<td>link (Pass: yrtj)</td>
</tr>
<tr>
<td>nu_eng2_corridor_3</td>
<td>fisheye (mono)</td>
<td>2:04</td>
<td>link</td>
<td>link (Pass: btpj)</td>
</tr>
<tr>
<td>ALL</td>
<td>fisheye (mono)</td>
<td></td>
<td>link</td>
<td>link (Pass: gumj)</td>
</tr>
</tbody>
</table>
After downloading and uncompressing a zip file, you will find a video file and a config file (old format) under the uncompressed directory.

```bash
$ ls dataset_name_X/
config.yaml  video.mp4
```

You can put the dataset in any directory where you have access to.

Additionally, please download a vocabulary file for FBoW from [here](#).

For the rest of this chapter, we will use `aist_living_lab_1` and `aist_living_lab_2` datasets for our example.

### 3.3 Tracking and Mapping

Here we should know how to run SLAM and create a map database file with `aist_living_lab_1` dataset. You can use `./run_video_slam` to run SLAM with the video file.

```bash
# at the build directory of OpenVSLAM
$ ls
...
run_video_slam
...
$ ./run_video_slam -h
```

Allowed options:

- `h`, `--help` produce help message
- `-v`, `--vocab` arg vocabulary file path
- `-m`, `--video` arg video file path
- `-c`, `--config` arg config file path
- `--mask` arg mask image path
- `--frame-skip` arg (=1) interval of frame skip
- `--no-sleep` not wait for next frame in real time
- `--auto-term` automatically terminate the viewer
- `--debug` debug mode
- `--eval-log` store trajectory and tracking times for evaluation
- `--map-db` arg store a map database at this path after SLAM

Execute the following command to run SLAM. The paths should be changed accordingly.

```bash
$ ./run_video_slam \
-v /path/to/orb_vocab/orb_vocab.fbow \
-c /path/to/openvslam/example/aist/equirectangular.yaml \
-m /path/to/aist_living_lab_1/video.mp4 \
--frame-skip 3 \
--map-db aist_living_lab_1_map.msg
```

The frame viewer and map viewer should launch as well. If the two viewers are not launching correctly, check if you launched the command with the appropriate paths.
Chapter 3. Simple Tutorial
Camera Configuration:
- name: RICOH THETA S 960
- setup: Monocular
- fps: 30
- cols: 1920
- rows: 960
- color: RGB
- model: Equirectangular

ORB Configuration:
- number of keypoints: 2000
- scale factor: 1.2
- number of levels: 8
- initial fast threshold: 20
- minimum fast threshold: 7
- edge threshold: 19
- patch size: 31
- half patch size: 15
- mask rectangles:
  - [0, 1, 0, 0.1]
  - [0, 1, 0.84, 1]
  - [0, 0.2, 0.7, 1]
  - [0.8, 1, 0.7, 1]

Tracking Configuration:

(continues on next page)
Please click the **Terminate** button to close the viewer.

After terminating, you will find a map database file `aist_living_lab_1_map.msg`.

```bash
$ ls
...  
aist_living_lab_1_map.msg  
...  
```

The format of map database files is **MessagePack**, so you can reuse created maps for any third-party applications other than OpenVSLAM.

### 3.4 Localization

In this section, we will localize the frames in `aist_living_lab_2` dataset using the created map file `aist_living_lab_1_map.msg`. You can use `./run_video_localization` to run localization.

```bash
$ ./run_video_localization -h
Allowed options:
-h, --help           produce help message
-v, --vocab arg      vocabulary file path
-m, --video arg      video file path
-c, --config arg     config file path
-p, --map-db arg     path to a prebuilt map database
--mapping            perform mapping as well as localization
--mask arg           mask image path
--frame-skip arg (=1) interval of frame skip
--no-sleep           not wait for next frame in real time
--auto-term          automatically terminate the viewer
--debug              debug mode
```

Execute the following command to start localization. The paths should be changed accordingly.

```bash
$ ./run_video_localization \
  -v /path/to/orb_vocab/orb_vocab.fbow \
  -c /path/to/openvslam/example/aist/equirectangular.yaml \
  -m /path/to/aist_living_lab_2/video.mp4 \
  --frame-skip 3 \
  --map-db aist_living_lab_1_map.msg
```
The frame viewer and map viewer should launch as well. If the two viewers are not launching correctly, check if you launched the command with the appropriate paths.

You can see if the current frame is being localized, based on the prebuild map.
- name: RICOH THETA S 960
- setup: Monocular
- fps: 30
- cols: 1920
- rows: 960
- color: RGB
- model: Equirectangular

ORB Configuration:
- number of keypoints: 2000
- scale factor: 1.2
- number of levels: 8
- initial fast threshold: 20
- minimum fast threshold: 7
- edge threshold: 19
- patch size: 31
- half patch size: 15
- mask rectangles:
  - [0, 1, 0, 0.1]
  - [0, 1, 0.84, 1]
  - [0, 0.2, 0.7, 1]
  - [0.8, 1, 0.7, 1]

Tracking Configuration:

```
[2019-05-20 17:58:54.729] [I] loading ORB vocabulary: /path/to/orb_vocab/orb_vocab.fbow
[2019-05-20 17:58:55.083] [I] clear map database
[2019-05-20 17:58:55.083] [I] clear BoW database
[2019-05-20 17:58:55.083] [I] load the MessagePack file of database from aist_living_lab...
[2019-05-20 17:58:57.832] [I] decoding 1 camera(s) to load
[2019-05-20 17:58:57.832] [I] load the tracking camera "RICOH THETA S 960" from JSON
[2019-05-20 17:58:58.204] [I] decoding 301 keyframes to load
[2019-05-20 17:59:02.013] [I] decoding 19900 landmarks to load
[2019-05-20 17:59:02.564] [I] registering essential graph
[2019-05-20 17:59:03.161] [I] updating covisibility graph
[2019-05-20 17:59:03.341] [I] updating landmark geometry
[2019-05-20 17:59:04.189] [I] startup SLAM system
[2019-05-20 17:59:04.190] [I] start local mapper
[2019-05-20 17:59:04.191] [I] start loop closer
[2019-05-20 17:59:04.195] [I] pause local mapper
[2019-05-20 17:59:04.424] [I] relocalization succeeded
[2019-05-20 18:01:12.387] [I] shutdown SLAM system
median tracking time: 0.0370831[s]
mean tracking time: 0.0384683[s]
[2019-05-20 18:01:12.390] [I] clear BoW database
[2019-05-20 18:01:12.574] [I] clear map database
```

If you set the `--mapping` option, the mapping module is enabled to extend the prebuild map.
4.1 Instructions for PangolinViewer

Dockerfile.desktop can be used for easy installation. This chapter provides instructions on building and running examples with PangolinViewer support using Docker.

The instructions are tested on Ubuntu 16.04 and 18.04 and 20.04. Docker for Mac are NOT supported due to OpenGL forwarding.

Note that docker host machines with NVIDIA graphics cards are NOT officially supported yet.

Note: If you’re using Ubuntu, there are easy setup scripts in scripts/ubuntu. see scripts/ubuntu/README.md

If you plan on using a machine with NVIDIA graphics card(s), please use nvidia-docker2 and the version 390 or later of NVIDIA driver. These examples depend on X11 forwarding with OpenGL for visualization. Note that our Dockerfile.desktop is NOT compatible with nvidia-docker1.

If the viewer cannot be launched at all or you are using macOS, please install the dependencies manually or use the docker images for SocketViewer.

4.1.1 Building Docker Image

Execute the following commands:

```
git clone https://github.com/OpenVSLAM-Community/openvslam.git
cd openvslam
git submodule update -i --recursive
docker build -t openvslam-desktop -f Dockerfile.desktop .
```

You can accelerate the build of the docker image with --build-arg NUM_THREADS=<number of parallel builds> option. For example:

```
# building the docker image with four threads
docker build -t openvslam-desktop -f Dockerfile.desktop . --build-arg NUM_THREADS=`expr \$\(nproc\) - 1`'
```
4.1.2 Starting Docker Container

In order to enable X11 forwarding, supplemental options (-e DISPLAY=$DISPLAY and -v /tmp/.X11-unix:/tmp/.X11-unix:ro) are needed for docker run.

```
# before launching the container, allow display access from local users
xhost +local:

# launch the container
docker run -it --rm -e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix:ro openvslam-
→ desktop
```

**Note:** Additional option --runtime=nvidia is needed if you use NVIDIA graphics card(s). If you’re using Docker with Native GPU Support then the options are --gpus all. Please see here for more details.

After launching the container, the shell interface will be launched in the docker container.

```
root@ddad048b5fff:/openvslam/build# ls
lib run_image_slam run_video_slam
run_euroc_slam run_kitti_slam run_tum_slam
run_image_localization run_video_localization run_tum_rgbd_localization
```

See Tutorial to run SLAM examples in the container.

**Note:** If the viewer does not work, please install the dependencies manually on your host machine or use the docker images for SocketViewer instead.

If you need to access to any files and directories on a host machine from the container, bind directories between the host and the container.

4.2 Instructions for SocketViewer

Dockerfile.socket and viewer/Dockerfile can be used for easy installation. This chapter provides instructions on building and running examples with SocketViewer support using Docker.

4.2.1 Building Docker Images

**Docker Image of OpenVSLAM**

Execute the following commands:

```
cd /path/to/openvslam
docker build -t openvslam-socket -f Dockerfile.socket .
```

You can accelerate the build of the docker image with --build-arg NUM_THREADS=<number of parallel builds> option. For example:

```
# building the docker image with four threads
docker build -t openvslam-socket -f Dockerfile.socket . --build-arg NUM_THREADS=\'expr \$\(nproc\)-1\'
```
Docker Image of Server

Execute the following commands:

```
cd /path/to/openvslam
cd viewer
docker build -t openvslam-server .
```

4.2.2 Starting Docker Containers

On Linux

Launch the server container and access to it with the web browser in advance. Please specify \(--net=host\) in order to share the network with the host machine.

```
$ docker run --rm -it --name openvslam-server --net=host openvslam-server
WebSocket: listening on *:3000
HTTP server: listening on *:3001
```

After launching, access to http://localhost:3001/ with the web browser.

Next, launch the container of OpenVSLAM. The shell interface will be launched in the docker container.

```
$ docker run --rm -it --name openvslam-socket --net=host openvslam-socket
root@hostname:/openvslam/build#
```

See Tutorial to run SLAM examples in the container.

If you need to access to any files and directories on a host machine from the container, \textit{bind directories} between the host and the container.

On macOS

Launch the server container and access to it with the web browser in advance. Please specify \(-p 3001:3001\) for port-forwarding.

```
$ docker run --rm -it --name openvslam-server -p 3001:3001 openvslam-server
WebSocket: listening on *:3000
HTTP server: listening on *:3001
```

After launching, access to http://localhost:3001/ with the web browser.

Then, inspect the container’s IP address and append the SocketPublisher.server_uri entry to the YAML config file of OpenVSLAM.

```
# inspect the server's IP address
$ docker inspect openvslam-server | grep -m 1 "IPAddress"
| sed 's/ //g' | sed 's/.///g' "IPAddress": "172.17.0.2"
```

```
# config file of OpenVSLAM
...
```

(continues on next page)
Next, launch the container of OpenVSLAM. The shell interface will be launched in the docker container.

```bash
$ docker run --rm -it --name openvslam-socket openvslam-socket
root@hostname:/openvslam/build#
```

See Tutorial to run SLAM examples in the container. Please don’t forget to append `SocketPublisher.server_uri` entry to the `config.yaml` if you use the downloaded datasets in the tutorial.

If you need to access to any files and directories on a host machine from the container, *bind directories* between the host and the container.

### 4.3 Bind of Directories

If you need to access to any files and directories on a host machine from the container, bind directories between the host and the container using `--volume` or `--mount` option. (See the docker documentation.)

For example:

```bash
# launch a container of openvslam-desktop with --volume option
$ docker run -it --rm --runtime=nvidia -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix:ro \
  --volume /path/to/dataset/dir:/dataset:ro \
  --volume /path/to/vocab/dir:/vocab:ro \
  openvslam-desktop
# dataset/ and vocab/ are found at the root directory in the container
root@0c0c9f115d74:/# ls /
... dataset/ vocab/ ...
```

```bash
# launch a container of openvslam-socket with --volume option
$ docker run --rm -it --name openvslam-socket --net=host \
  --volume /path/to/dataset/dir:/dataset:ro \
  --volume /path/to/vocab/dir:/vocab:ro \
  openvslam-socket
# dataset/ and vocab/ are found at the root directory in the container
root@0c0c9f115d74:/# ls /
... dataset/ vocab/ ...
```
We provide ROS and ROS2 package examples to help you run OpenVSLAM on ROS framework.

5.1 ROS Package

5.1.1 Installation

Requirements

- **ROS**: noetic is recommended. (If you have built OpenCV (3.3.1 or later) manually, you can use melodic or later.)
- **OpenVSLAM**
- **image_transport**: Required by this ROS package examples.
- **cv_bridge**: Please build it with the same version of OpenCV used in OpenVSLAM.

Prerequisites

Tested for Ubuntu 18.04.

Please install the following dependencies.

- **ROS**: Please follow Installation of ROS.
- **OpenVSLAM**: Please follow Installation of OpenVSLAM.

**Note**: Please build OpenVSLAM with PangolinViewer or SocketViewer if you plan on using it for the examples.

Install the dependencies via `apt`.

```
apt update -y
apt install ros-$ROS_DISTRO-image-transport
```

Download the source of cv_bridge.

```
mkdir -p ~/catkin_ws/src
git clone --branch $ROS_DISTRO --depth 1 https://github.com/ros-perception/visionopencv.git
```

(continues on next page)
OpenVSLAM

(continued from previous page)

```
cp -r vision_opencv/cv_bridge ~/catkin_ws/src
rm -rf vision_opencv
```

### Build Instructions

When building with support for PangolinViewer, please specify the following cmake options: `-DUSE_PANGOLIN_VIEWER=ON` and `-DUSE_SOCKET_PUBLISHER=OFF` as described in *build of OpenVSLAM*. openvslam and openvslam_ros need to be built with the same options.

```
cd ~/catkin_ws/src
git clone --branch ros --depth 1 https://github.com/OpenVSLAM-Community/openvslam_ros.git
cd ~/catkin_ws
catkin_make -DUSE_PANGOLIN_VIEWER=ON -DUSE_SOCKET_PUBLISHER=OFF
```

### 5.1.2 Examples

Run the core program required for ROS-based system in advance.

```
roscore
```

**Note:** Please leave the `roscore` run.

**Publisher**

If you want to input image sequences or videos into openvslam_ros, please use ROS2.

**Publish Images of a USB Camera**

For using a standard USB camera for visual SLAM or localization.

```
apt install ros-${ROS_DISTRO}-usb-cam
rosparam set usb_cam/pixel_format yuyv
rosrun usb_cam usb_cam_node
```

Republish the ROS topic to `/camera/image_raw`.

```
rosrun image_transport republish \
    --in:=/usb_cam/image_raw \n    --out:=/camera/image_raw
```
Subscriber

Subscribers continually receive images. Please execute one of the following command snippets in the new terminal.

Note: Option arguments are the same as the examples of OpenVSLAM.

Tracking and Mapping

We provide an example snippet for visual SLAM. The source code is placed at openvslam_ros/src/run_slam.cc.

```
source ~/catkin_ws/devel/setup.bash
rosrun openvslam_ros run_slam \
   -v /path/to/orb_vocab.fbow \
   -c /path/to/config.yaml
```

Localization

We provide an example snippet for localization based on a prebuilt map. The source code is placed at openvslam_ros/src/run_localization.cc.

```
source ~/catkin_ws/devel/setup.bash
rosrun openvslam_ros run_localization \
   -v /path/to/orb_vocab.fbow \
   -c /path/to/config.yaml \
   --map-db /path/to/map.msg
```

5.2 ROS2 Package

5.2.1 Installation

Requirements

- ROS2: foxy or later.
- OpenVSLAM
- image_common: Required by this ROS package examples.
- vision_opencv: Please build it with the same version of OpenCV used in OpenVSLAM.
- image_tools: An optional requirement to use USB cameras.
**Prerequisites**

Tested for **Ubuntu 18.04**.

Please install the following dependencies.

- ROS2: Please follow [Installation of ROS2](#).
- OpenVSLAM: Please follow [Installation of OpenVSLAM](#).

**Note:** Please build OpenVSLAM with PangolinViewer or SocketViewer if you plan on using it for the examples.

Download repositories of `image_common` and `vision_opencv`.

```bash
midir -p ~/ros2_ws/src
cd ~/ros2_ws/src
git clone -b ${{ROS_DISTRO}} --single-branch https://github.com/ros-perception/image_common.git
git clone -b ros2 --single-branch https://github.com/ros-perception/vision_opencv.git
```

For using USB cam as a image source, download a repository of `demos` and pick `image_tools` module.

```bash
cd ~/ros2_ws
git clone https://github.com/ros2/demos.git
cp -r demos/image_tools src/
rm -rf demos
```

**Build Instructions**

When building with support for PangolinViewer, please specify the following cmake options: `-DUSE_PANGOLIN_VIEWER=ON` and `-DUSE_SOCKET_PUBLISHER=OFF` as described in *build of OpenVSLAM*.

`openvslam` and `openvslam_ros` need to be built with the same options.

```bash
cd ~/catkin_ws/src
git clone --branch ros2 --depth 1 https://github.com/OpenVSLAM-Community/openvslam_ros.git
cd ~/ros2_ws
colcon build --symlink-install --cmake-args -DUSE_PANGOLIN_VIEWER=ON -DUSE_SOCKET_PUBLISHER=OFF
```

### 5.2.2 Examples

**Publisher**

If you want to input image sequences or videos into `openvslam_ros`, please refer to `dataset_publisher_ros2`. 
Publish Images Captured by a USB Camera

For using a standard USB camera for visual SLAM or localization.

```bash
ros2 run image_tools cam2image
```

Republish the ROS topic to /camera/image_raw.

```bash
ros2 run image_transport republish \ raw in:=image raw out:=/camera/image_raw
```

Subscriber

Subscribers continually receive images. Please execute one of the following command snippets in the new terminal.

**Note:** Option arguments are the same as the examples of OpenVSLAM.

Tracking and Mapping

We provide an example snippet for visual SLAM. The source code is placed at openvslam_ros/src/run_slam.cc.

```bash
source ~/ros2_ws/install/setup.bash
ros2 run openvslam_ros run_slam \ -v /path/to/orb_vocab.fbow \ -c /path/to/config.yaml
```

Localization

We provide an example snippet for localization based on a prebuilt map. The source code is placed at openvslam_ros/src/run_localization.cc.

```bash
source ~/ros2_ws/install/setup.bash
ros2 run openvslam_ros run_localization \ -v /path/to/orb_vocab.fbow \ -c /path/to/config.yaml \ --map-db /path/to/map.msg
```
We provided example code snippets for running OpenVSLAM with variety of datasets.

### 6.1 SLAM with Video Files

#### 6.1.1 Tracking and Mapping

We provide an example snippet for using video files (e.g. .mp4) for visual SLAM. The source code is placed at ./example/run_video_slam.cc. The following options are allowed:

```
$ ./run_video_slam -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-m, --video arg video file path
-c, --config arg config file path
--mask arg mask image path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
--eval-log store trajectory and tracking times for evaluation
-p, --map-db arg store a map database at this path after SLAM
```

The camera that captures the video file must be calibrated. Create a config file (.yaml) according to the camera parameters.

We provided a vocabulary file for FBoW at [here](#).
## 6.1.2 Localization

We provide an example snippet for using video files (e.g., .mp4) for localization based on a prebuilt map. The source code is placed at ./example/run_video_localization.cc. The following options are allowed:

```
$ ./run_video_localization -h
Allowed options:
-h, --help          produce help message
-v, --vocab arg    vocabulary file path
-m, --video arg    video file path
-c, --config arg   config file path
-p, --map-db arg   path to a prebuilt map database
--mapping          perform mapping as well as localization
--mask arg         mask image path
--frame-skip arg   (=1) interval of frame skip
--no-sleep          not wait for next frame in real time
--auto-term         automatically terminate the viewer
--debug             debug mode
```

The camera that captures the video file must be calibrated. Create a config file (.yaml) according to the camera parameters.

We provided a vocabulary file for FBoW at here.

You can create a map database file by running one of the run_***_slam executables with --map-db map_file_name.msg option.

## 6.2 SLAM with Image Sequences

### 6.2.1 Tracking and Mapping

We provided an example snippet for using image sequences for visual SLAM. The source code is placed at ./example/run_image_slam.cc. The following options are allowed:

```
$ ./run_image_slam -h
Allowed options:
-h, --help          produce help message
-v, --vocab arg    vocabulary file path
-i, --img-dir arg  directory path which contains images
-c, --config arg   config file path
--mask arg         mask image path
--frame-skip arg   (=1) interval of frame skip
--no-sleep          not wait for next frame in real time
--auto-term         automatically terminate the viewer
--debug             debug mode
--eval-log          store trajectory and tracking times for evaluation
-p, --map-db arg   store a map database at this path after SLAM
```

The camera that captures the video file must be calibrated. Create a config file (.yaml) according to the camera parameters.
parameters.
We provided a vocabulary file for FBoW at here.

### 6.2.2 Localization

We provided an example snippet for using image sequences for localization based on a prebuilt map. The source code is placed at ./example/run_image_localization.cc. The following options are allowed:

```
$ ./run_image_localization -h
Allowed options:
- h, --help produce help message
- v, --vocab arg vocabulary file path
- i, --img-dir arg directory path which contains images
- c, --config arg config file path
- p, --map-db arg path to a prebuilt map database
--mapping perform mapping as well as localization
--mask arg mask image path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
```

The camera that captures the video file must be calibrated. Create a config file (.yaml) according to the camera parameters.

We provided a vocabulary file for FBoW at here.

You can create a map database file by running one of the run_****_slam executables with --map-db map_file_name.msg option.

### 6.3 SLAM with Standard Datasets

#### 6.3.1 KITTI Odometry dataset

KITTI Odometry dataset is a benchmarking dataset for monocular and stereo visual odometry and lidar odometry that is captured from car-mounted devices. We provided an example source code for running monocular and stereo visual SLAM with this dataset. The source code is placed at ./example/run_kitti_slam.cc.

Start by downloading the dataset from here. Download the grayscale set (data_odometry_gray.zip).

After downloading and uncompressing it, you will find several sequences under the sequences/ directory.

```
$ ls sequences/
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
```

In addition, download a vocabulary file for FBoW from here.

A configuration file for each sequence is contained under ./example/kitti/.

If you built examples with Pangolin Viewer support, a map viewer and frame viewer will be launched right after executing the following command.
# at the build directory of OpenVSLAM

```
$ ls
...
run_kitti_slam
...
```

# monocular SLAM with sequence 00

```
$ ./run_kitti_slam
   -v /path/to/orb_vocab/orb_vocab.fbow
   -d /path/to/KITTI/Odometry/sequences/00/
   -c ../example/kitti/KITTI_mono_00-02.yaml
```

# stereo SLAM with sequence 05

```
$ ./run_kitti_slam
   -v /path/to/orb_vocab/orb_vocab.fbow
   -d /path/to/KITTI/Odometry/sequences/05/
   -c ../example/kitti/KITTI_stereo_04-12.yaml
```

The following options are allowed:

```
$ ./run_kitti_slam -h
Allowed options:
- h, --help produce help message
- v, --vocab arg vocabulary file path
- d, --data-dir arg directory path which contains dataset
- c, --config arg config file path
- frame-skip arg (=1) interval of frame skip
- no-sleep not wait for next frame in real time
- auto-term automatically terminate the viewer
- debug debug mode
- eval-log store trajectory and tracking times for evaluation
- p, --map-db arg store a map database at this path after SLAM
```

### 6.3.2 EuRoC MAV dataset

EuRoC MAV dataset is a benchmarking dataset for monocular and stereo visual odometry that is captured from drone-mounted devices. We provide an example source code for running monocular and stereo visual SLAM with this dataset. The source code is placed at ./example/run_euroc_slam.cc.

Start by downloading the dataset from here. Download the .zip file of a dataset you plan on using.

After downloading and uncompressing it, you will find several directories under the mav0/ directory.

```
$ ls mav0/
body.yaml cam0 cam1 imu0 leica0 state_groundtruth_estimate0
```

In addition, download a vocabulary file for FBoW from here.

We provided the two config files for EuRoC, ./example/euroc/EuRoC_mono.yaml for monocular and ./example/euroc/EuRoC_stereo.yaml for stereo.

If you have built examples with Pangolin Viewer support, a map viewer and frame viewer will be launched right after executing the following command.

```
# at the build directory of OpenVSLAM

$ ls
```
... run_euroc_slam ...

# monocular SLAM with any EuRoC sequence
$ ./run_euroc_slam \
   -v /path/to/orb_vocab/orb_vocab.fbow \
   -d /path/to/EuRoC/MAV/mav0/ \
   -c ../example/euroc/EuRoC_mono.yaml

# stereo SLAM with any EuRoC sequence
$ ./run_euroc_slam \
   -v /path/to/orb_vocab/orb_vocab.fbow \
   -d /path/to/EuRoC/MAV/mav0/ \
   -c ../example/euroc/EuRoC_stereo.yaml

The following options are allowed:

```
$ ./run_euroc_slam -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-d, --data-dir arg directory path which contains dataset
-c, --config arg config file path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
--eval-log store trajectory and tracking times for evaluation
-p, --map-db arg store a map database at this path after SLAM
```

6.3.3 TUM RGBD dataset

TUM RGBD dataset is a benchmarking dataset containing RGB-D data and ground-truth data with the goal to establish a novel benchmark for the evaluation of visual odometry and visual SLAM systems. The source code is placed at ./example/run_tum_rgbd_localization.cc.

Start by downloading the various dataset from here. One of many example datasets can be found from here. Download the .tgz file of a dataset you plan on using.

After downloading and uncompressing it, you will find two directories and few text files under the rgbd_dataset_freiburg3_calibration_rgb_depth/ directory.

```
$ ls rgbd_dataset_freiburg3_calibration_rgb_depth
accelerometer.txt  depth depth.txt  groundtruth.txt  rgb  rgb.txt
```

If you would like to preprocess dataset then you can use tool from here.

In addition, download a vocabulary file for FBoW from here.

We provided the config files for RGBD dataset at, ./example/tum_rgbd.

For above specific example we shall use two config files, ./example/tum_rgbd/TUM_RGBD_mono_3.yaml for monocular and ./example/tum_rgbd/TUM_RGBD_rgbd_3.yaml for RGBD.
6.3.4 Tracking and Mapping

```bash
# at the build directory of OpenVSLAM
$ ls
...
run_tum_rgbd_slam
...

# monocular SLAM with rgbd_dataset_freiburg3_calibration_rgb_depth
$ ./run_tum_rgbd_slam
  -v /path/to/orb_vocab/orb_vocab.fbow
  -d /path/to/rgbd_dataset_freiburg3_calibration_rgb_depth/
  -c ../example/tum_rgbd/TUM_RGBD_mono_3.yaml
  --no-sleep
  --auto-term
  --map-db fr3_slam_mono.msg

# RGBD SLAM with rgbd_dataset_freiburg3_calibration_rgb_depth
$ ./run_tum_rgbd_slam
  -v /path/to/orb_vocab/orb_vocab.fbow
  -d /path/to/rgbd_dataset_freiburg3_calibration_rgb_depth/
  -c ../example/tum_rgbd/TUM_RGBD_rgbd_3.yaml
  --no-sleep
  --auto-term
  --map-db fr3_slam_rgbd.msg
```

The following options are allowed:

```bash
$ ./run_tum_rgbd_slam -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-d, --data-dir arg directory path which contains dataset
-c, --config arg config file path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
--eval-log store trajectory and tracking times for evaluation
-p, --map-db arg store a map database at this path after SLAM
```

6.3.5 Localization

```bash
# at the build directory of OpenVSLAM
$ ls
...
run_tum_rgbd_localization
...

# monocular localization with rgbd_dataset_freiburg3_calibration_rgb_depth
$ ./run_tum_rgbd_localization
  -v /path/to/orb_vocab/orb_vocab.fbow
  -d /path/to/rgbd_dataset_freiburg3_calibration_rgb_depth/
```
The following options are allowed:

```
$ ./run_tum_rgbd_localization -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-d, --data-dir arg directory path which contains dataset
-c, --config arg config file path
--frame-skip arg (=1) interval of frame skip
--no-sleep not wait for next frame in real time
--auto-term automatically terminate the viewer
--debug debug mode
--mapping perform mapping as well as localization
-p, --map-db arg store a map database at this path after SLAM
```

### 6.4 SLAM with UVC camera

#### 6.4.1 Tracking and Mapping

We provided an example snippet for using a UVC camera, which is often called a webcam, for visual SLAM. The source code is placed at `./example/run_camera_slam.cc`. The following options are allowed:

```
$ ./run_camera_slam -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-n, --number arg camera number
-c, --config arg config file path
--mask arg mask image path
--scale arg (=1) scaling ratio of images
-p, --map-db arg store a map database at this path after SLAM
--debug debug mode
```

Please specify the camera number you want to use by `-n` option.

The camera must be calibrated. Create a config file (`.yaml`) according to the camera parameters.
You can scale input images to the performance of your machine by \texttt{-s} option. Please modify the config accordingly. We provided a vocabulary file for FBoW at \url{here}.

### 6.4.2 Localization

We provided an example snippet for using a UVC camera for localization based on a prebuilt map. The source code is placed at \texttt{./example/run_camera_localization.cc}. The following options are allowed:

```
$ ./run_camera_localization -h
Allowed options:
-h, --help produce help message
-v, --vocab arg vocabulary file path
-n, --number arg camera number
-c, --config arg config file path
--mask arg mask image path
-s, --scale arg (=1) scaling ratio of images
-p, --map-db arg path to a prebuilt map database
--mapping perform mapping as well as localization
--debug debug mode
```

Please specify the camera number you want to use by \texttt{-n} option. The camera must be calibrated. Create a config file (\texttt{.yaml}) according to the camera parameters. You can scale input images to the performance of your machine by \texttt{-s} option. Please modify the config accordingly. We provided a vocabulary file for FBoW at \url{here}.  

## 7.1 Camera

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>It is used by the camera database to identify the camera.</td>
</tr>
<tr>
<td>setup</td>
<td>monocular, stereo, RGBD</td>
</tr>
<tr>
<td>model</td>
<td>perspective, fisheye, equirectangular, radial_division (note: If you want to use stereo_rectifier, you need to specify perspective.)</td>
</tr>
<tr>
<td>fx, fy</td>
<td>Focal length (pixel)</td>
</tr>
<tr>
<td>cx, cy</td>
<td>Principal point (pixel)</td>
</tr>
<tr>
<td>k1, k2, p1, p2, k3</td>
<td>Distortion parameters for perspective camera. When using StereoRectifier, there is no distortion after stereo rectification.</td>
</tr>
<tr>
<td>k1, k2, k3, k4</td>
<td>Distortion parameters for fisheye camera</td>
</tr>
<tr>
<td>distortion</td>
<td>Distortion parameters for radial_division camera</td>
</tr>
<tr>
<td>fps</td>
<td>Framerate of input images</td>
</tr>
<tr>
<td>cols, rows</td>
<td>Resolution (pixel)</td>
</tr>
<tr>
<td>color_order</td>
<td>Gray, RGB, RGBA, BGR, BGRA</td>
</tr>
<tr>
<td>focal_x_baseline</td>
<td>For stereo cameras, it is the value of the baseline between the left and right cameras multiplied by the focal length fx. For RGBD cameras, if the measurement method is stereo, set it based on its baseline. If the measurement method is other than that, set the appropriate value based on the relationship between depth accuracy and baseline.</td>
</tr>
<tr>
<td>depth_threshold</td>
<td>The ratio used to determine the depth threshold.</td>
</tr>
</tbody>
</table>

## 7.2 Feature

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of ORB feature extraction model (id for saving)</td>
</tr>
<tr>
<td>scale_factor</td>
<td>Scale of the image pyramid</td>
</tr>
<tr>
<td>num_levels</td>
<td>Number of levels of in the image pyramid</td>
</tr>
<tr>
<td>ini_fast_threshold</td>
<td>FAST threshold for try first</td>
</tr>
<tr>
<td>min_fast_threshold</td>
<td>FAST threshold for try second time</td>
</tr>
</tbody>
</table>
7.3 Preprocessing

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_num_keypoints</td>
<td>Maximum number of feature points per frame to be used for SLAM.</td>
</tr>
<tr>
<td>ini_max_num_keypoints</td>
<td>Maximum number of feature points per frame to be used for Initialization. It is only used for monocular camera models.</td>
</tr>
<tr>
<td>depthmap_factor</td>
<td>The ratio used to convert depth image pixel values to distance.</td>
</tr>
</tbody>
</table>

7.4 Tracking

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reloc_distance_threshold</td>
<td>Maximum distance threshold (in meters) where close keyframes could be found when doing a relocalization by pose.</td>
</tr>
<tr>
<td>reloc_angle_threshold</td>
<td>Maximum angle threshold (in radians) between given pose and close keyframes when doing a relocalization by pose.</td>
</tr>
<tr>
<td>enable_auto_relocalization</td>
<td>If true, automatically try to relocalize when lost.</td>
</tr>
<tr>
<td>use_robust_matcher_for_relocalization_request</td>
<td>Use robust_matcher for relocalization request.</td>
</tr>
</tbody>
</table>

7.5 Mapping

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline_dist_thr_ratio</td>
<td>For two frames of baseline below the threshold, no triangulation will be performed. In the monocular case, the scale is indefinite, so relative values are recommended. Either baseline_dist_thr or this one should be specified. If not specified, baseline_dist_thr_ratio will be used.</td>
</tr>
<tr>
<td>baseline_dist_thr</td>
<td>For two frames of baseline below the threshold, no triangulation will be performed.</td>
</tr>
<tr>
<td>redundant_obs_ratio_thr</td>
<td></td>
</tr>
</tbody>
</table>

7.6 StereoRectifier

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Camera model type before rectification. The option is perspective or fisheye. (note: If you want to use fisheye model for stereo_rectifier, you need to specify Camera::model to perspective.)</td>
</tr>
<tr>
<td>K_left, K_right</td>
<td>Intrinsic parameters. The 3x3 matrix are written in row-major order.</td>
</tr>
<tr>
<td>D_left, D_right</td>
<td>Distortion parameters. The 5 parameters are k1, k2, p1, p2, k3.</td>
</tr>
<tr>
<td>R_left, R_right</td>
<td>Stereo-recification parameters. The 3x3 matrix are written in row-major order.</td>
</tr>
</tbody>
</table>
7.7 Initializer

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_min_triangulated_pts</td>
<td>Minimum number of triangulated points</td>
</tr>
</tbody>
</table>

7.8 Relocalizer

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bow_match_lowe_ratio</td>
<td></td>
</tr>
<tr>
<td>proj_match_lowe_ratio</td>
<td></td>
</tr>
<tr>
<td>min_num_bow_matches</td>
<td></td>
</tr>
<tr>
<td>min_num_valid_obs</td>
<td></td>
</tr>
</tbody>
</table>

7.9 KeyframeInserter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_interval</td>
<td>max interval to insert keyframe</td>
</tr>
<tr>
<td>lms_ratio_thr_almost_all_lms_tracked</td>
<td>ratio-threshold of “the number of 3D points observed in the current frame” / “that of 3D points observed in the last keyframe”</td>
</tr>
<tr>
<td>lms_ratio_thr_view_changed</td>
<td>threshold of “the number of 3D points observed in the current frame” / “that of 3D points observed in the last keyframe”</td>
</tr>
</tbody>
</table>

7.10 PangolinViewer

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyframe_size</td>
<td></td>
</tr>
<tr>
<td>keyframe_line_width</td>
<td></td>
</tr>
<tr>
<td>graph_line_width</td>
<td></td>
</tr>
<tr>
<td>point_size</td>
<td></td>
</tr>
<tr>
<td>camera_size</td>
<td></td>
</tr>
<tr>
<td>camera_line_width</td>
<td></td>
</tr>
<tr>
<td>viewpoint_x, viewpoint_y, viewpoint_z, viewpoint_f</td>
<td></td>
</tr>
</tbody>
</table>
### 7.11 SocketPublisher

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emitting_interval</td>
<td>If true, pointcloud transfer is enabled. The default is true. Pointcloud transfer is slow, so disabling pointcloud transfer may be useful to improve performance of SocketViewer.</td>
</tr>
<tr>
<td>image_quality</td>
<td></td>
</tr>
<tr>
<td>server_uri</td>
<td></td>
</tr>
<tr>
<td>publish_points</td>
<td></td>
</tr>
</tbody>
</table>

### 7.12 LoopDetector

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>flag which indicates the loop detector is enabled or not</td>
</tr>
<tr>
<td>num_final_matches_threshold</td>
<td>the threshold of the number of mutual matches after the Sim3 estimation</td>
</tr>
<tr>
<td>min_continuity</td>
<td>the threshold of the continuity of continuously detected keyframe set</td>
</tr>
</tbody>
</table>

### 7.13 BowDatabase

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject_by_graph_distance</td>
<td></td>
</tr>
<tr>
<td>loop_min_distance_on_graph</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER EIGHT

RELOCALIZATION

8.1 What is Relocalization? Why it is needed?

In Visual SLAM, the robot/camera explores its environment while

1. estimates its location using the map and the last location as prior information (Tracking), and simultaneously
2. update the map (the database that records landmarks) of environment (Mapping).

Relocalization module can estimate the location without using any prior information other than the map (with the high cost of computation). This is useful when the previous location cannot be used as prior information, for example when tracking fails.

8.2 Steps in Relocalization

1. Acquire relocalization candidate keyframes (Used as a reference keyframe for relocalization)
2. Compute matching points for each candidate by using BoW tree matcher
3. Discard the candidate if the number of 2D-3D matches is less than the threshold
4. Setup an PnP solver with the current 2D-3D matches
5. Estimate the camera pose using EPnP (+ RANSAC)
6. Apply pose optimizer
7. Apply projection match to increase 2D-3D matches
8. Re-apply the pose optimizer
9. Apply projection match again if the number of the observations is less than the threshold
10. Apply projection match again, then set the 2D-3D matches
11. Discard if the number of the observations is less than the threshold and do the pose estimation again
12. If the number of observation is greater than threshold succeed in relocalization

See the details on how to run the relocalization at here.
9.1 For building

1. OpenVSLAM terminates abnormally soon after launching or optimization with g2o.

   Please configure and rebuild g2o and OpenVSLAM with -DBUILD_WITH_MARCH_NATIVE=OFF option for cmake.

9.2 For SLAM
Chapter 9. Trouble Shooting
If you use OpenVSLAM for a publication, please cite it as:

@inproceedings{openvslam2019,
  author = {Sumikura, Shinya and Shibuya, Mikiya and Sakurada, Ken},
  title = {OpenVSLAM: A Versatile Visual SLAM Framework},
  booktitle = {Proceedings of the 27th ACM International Conference on Multimedia},
  series = {MM '19},
  year = {2019},
  isbn = {978-1-4503-6889-6},
  location = {Nice, France},
  pages = {2292--2295},
  numpages = {4},
  url = {http://doi.acm.org/10.1145/3343031.3350539},
  doi = {10.1145/3343031.3350539},
  acmid = {3350539},
  publisher = {ACM},
  address = {New York, NY, USA}
}