
XRPrimer

Release 0.6.0

XRPrimer Authors

Dec 15, 2022

INSTALLATION

1	Installation (CPP)	1
1.1	Requirements	1
1.2	Compilation	2
1.3	Test	2
1.4	How to link in C++ projects	3
2	Installation (Python)	5
2.1	Requirements	5
2.2	Prepare environment	5
2.3	Install XRPrimer (python)	5
3	Camera	7
3.1	Pinhole	7
3.2	Fisheye	9
4	Image	11
4.1	Attributes	11
4.2	Create an Image	11
4.3	From Image to OpenCV	11
4.4	From OpenCV to Image	12
5	Pose	13
5.1	Attributes	13
5.2	Construct a Pose	13
5.3	Set Pose to identity	13
5.4	Scale a Pose	14
5.5	Inverse a Pose	14
6	Triangulator	15
6.1	Prepare camera parameters	15
6.2	Build a triangulator	16
6.3	Set cameras of a triangulator	16
6.4	Triangulate points from 2D to 3D	16
6.5	Get reprojection error	17
6.6	Camera selection	17
6.7	Get a conjugated projector	17
7	Projector	19
7.1	Prepare camera parameters	19
7.2	Build a projector	19
7.3	Set cameras of a projector	20

7.4	Project points from 3D to 2D	20
7.5	Camera selection	20
8	Camera convention	21
8.1	Intrinsic convention	21
8.2	Extrinsic convention	22
9	Calibrate Multiple Cameras	23
10	Running Tests	25
10.1	Data Preparation	25
10.2	Environment Preparation	25
10.3	Running tests through pytest	25
11	Frequently Asked Questions	27
11.1	Installation	27
12	Changelog	29
12.1	v0.6.0 (01/09/2022/)	29
13	LICENSE	31
14	C++ documentation	37
15	xrprimer.data_structure	39
15.1	camera	39
16	xrprimer.calibration	43
17	xrprimer.ops	45
17.1	projection	45
17.2	triangulation	46
18	xrprimer.transform	49
18.1	camera	49
18.2	camera convention	51
18.3	image	51
19	xrprimer.utils	53
20	Indices and tables	59
	Python Module Index	61
	Index	63

INSTALLATION (CPP)

- Requirements
- Compilation
- Test
- How to link in C++ projects

1.1 Requirements

- C++14 or later compiler
- GCC 7.5+
- CMake 3.15+
- LAPACK & BLAS
 1. If using conda, `conda install -c conda-forge lapack`
 2. If sudo is available, `apt update & apt -y install libatlas-base-dev`

Optional:

- [Conan](#) (for using pre-built 3rd-party libraries)

```
# 0. install conan
pip install conan

# 1. first run
conan profile new --detect --force default
conan profile update settings.compiler.libcxx=libstdc++11 default

# 2. add conan artifactory
conan remote add openxrlab http://conan.openxrlab.org.cn/artifactory/api/conan/
↪ openxrlab

# 3. check
conan remote list
```

1.2 Compilation

```
git clone https://github.com/openxrlab/xrprimer.git
cd xrprimer/

cmake -S. -Bbuild [Compilation options]
cmake --build build --target install -j4
```

It is currently tested on Linux and iOS. Ideally it can be also compiled on macOS or Windows.

1.2.1 Compilation options

- `ENABLE_TEST` Enable unit test. default: OFF
- `PYTHON_BINDING` Enable Python binding. default: ON
- `BUILD_EXTERNAL` Enable build external. default: OFF, download deps libraries from conan.

```
# build external from source
cmake -S. -Bbuild -DBUILD_EXTERNAL=ON -DCMAKE_BUILD_TYPE=Release
cmake --build build --target install

# use conan for external
cmake -S. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --target install
```

1.2.2 Compilation on iOS

Refer to `build_ios.sh` for more details.

1.3 Test

CPP library

```
# compile (Skip the following two lines if it has been compiled)
cmake -S. -Bbuild -DCMAKE_BUILD_TYPE=Release -DENABLE_TEST=ON
cmake --build build -j4

# run test
cd build
wget -q https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrprimer/xrprimer.tar.gz && \
tar -xzf xrprimer.tar.gz && rm xrprimer.tar.gz
ln -sf xrprimer/test test
./bin/test_calibrator
```

Python library

```
# compile (Skip the following two lines if it has been compiled)
cmake -S. -Bbuild -DCMAKE_BUILD_TYPE=Release -DENABLE_TEST=ON
cmake --build build -j4
```

(continues on next page)

(continued from previous page)

```
# run test
cd build
wget -q https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrprimer/xrprimer.tar.gz &&
tar -xzf xrprimer.tar.gz && rm xrprimer.tar.gz
PYTHONPATH=./lib/ python ../cpp/tests/test_multi_camera_calibrator.py
```

1.4 How to link in C++ projects

see cpp sample

```
cmake_minimum_required(VERSION 3.16)

project(sample)

# set path for find XRPrimer package (config mode)
set(XRPrimer_DIR "<package_path>/lib/cmake")
find_package(XRPrimer REQUIRED)

add_executable(sample sample.cpp)

target_link_libraries(sample XRPrimer::xrprimer)
```


INSTALLATION (PYTHON)

- Requirements
- Prepare environment
- Install XRPrimer(python)

2.1 Requirements

- Linux
- Conda
- Python 3.6+

2.2 Prepare environment

- a. Create a conda virtual environment and activate it.

2.3 Install XRPrimer (python)

2.3.1 Install with pip

```
pip install xrprimer
```

2.3.2 Install by compiling from source

- a. Create a conda virtual environment and activate it.

```
conda create -n openxrlab python=3.8 -y  
conda activate openxrlab
```

- b. Clone the repo.

```
git clone https://github.com/openxrlab/xrprimer.git  
cd xrprimer/
```

c. (Optional) Install conan

```
# compiling with conan accelerates the compilation with pre-built libs, otherwise it_
↳ builds external libs from source
pip install conan
conan remote add openxrlab http://conan.openxrlab.org.cn/artifactory/api/conan/openxrlab
```

d. Install PyTorch and MMCV

Install PyTorch and torchvision following [official instructions](#).

E.g., install PyTorch 1.8.2 & CPU

```
pip install torch==1.8.2+cpu torchvision==0.9.2+cpu -f https://download.pytorch.org/whl/
↳ lts/1.8/torch_lts.html
```

Install mmcv without cuda operations

```
pip install mmcv
```

e. Install xrprimer in editable mode

```
pip install -e . # or "python setup.py develop"
python -c "import xrprimer; print(xrprimer.__version__)"
```

CAMERA

This file introduces the supported camera and distortion models. It is defined in C++ and bound to Python for extended use.

3.1 Pinhole

A pinhole camera is a simple camera without a lens but with a tiny aperture (from [Wikipedia](#)).

3.1.1 Attributes

Here are attributes of class `PinholeCameraParameter`:

For detailed convention, please refer to *camera convention doc*.

3.1.2 Create a Camera

Create a pinhole camera in C++

```
#include <data_structure/camera/pinhole_camera.h>

auto pinhole_param = PinholeCameraParameter();
std::cout << pinhole_param.ClassName() << std::endl;
```

Create a pinhole camera in Python

```
from xrprimer.data_structure.camera import PinholeCameraParameter

pinhole_param = PinholeCameraParameter()
print(type(pinhole_param).__name__)
```

3.1.3 File IO

A camera parameter defined in XRPrimer can dump its parameters to a json file or load a dumped json file easily.

```
# load method 1
pinhole_param = PinholeCameraParameter.fromfile('./pinhole_param.npz')
# load method 2
pinhole_param = PinholeCameraParameter()
pinhole_param.load('./pinhole_param.npz')
# dump method
pinhole_param.dump('./pinhole_param.npz')
```

3.1.4 Set intrinsic

There are 3 ways of setting intrinsic.

a. Set with a 4x4 K matrix.

```
pinhole_param.set_KRT(K=mat_4x4)
pinhole_param.set_resolution(h, w)
```

b. Set with a 3x3 K matrix.

```
# method 1, only for perspective camera
pinhole_param.set_KRT(K=mat_3x3)
pinhole_param.set_resolution(h, w)
# method 2
pinhole_param.set_intrinsic(
    mat3x3=mat_3x3,
    width=w, height=h,
    perspective=True)
```

c. Set with focal length and principal point.

```
pinhole_param.set_intrinsic(
    fx=focal[0], fy=focal[1],
    cx=principal[0], cy=principal[1],
    width=w, height=h,
    perspective=True)
```

3.1.5 Set extrinsics

To set extrinsic_r or extrinsic_t, call `set_KRT()`. Remember that `world2cam` argument is important, always check the direction before setting.

```
# set RT that transform points from camera space to world space
pinhole_param.set_KRT(R=mat_3x3, T=vec_3, world2cam=False)
# set RT but do not modify extrinsic direction stored in pinhole_param
pinhole_param.set_KRT(R=mat_3x3, T=vec_3)
```

3.1.6 Inverse extrinsics

Sometimes the extrinsic parameters are not what you desire. Call `inverse_extrinsic()` to inverse the direction, `world2cam` will be inversed synchronously.

```
assert pinhole_param.world2cam
world2cam_r = pinhole_param.get_extrinsic_r()
pinhole_param.inverse_extrinsic()
cam2world_r = pinhole_param.get_extrinsic_r()
```

3.1.7 Clone

In order to get a new camera parameter instance which can be modified arbitrarily, call `clone()`.

```
another_pinhole_param = pinhole_param.clone()
```

3.1.8 Get attributes

```
# intrinsic
intrinsic33 = pinhole_param.intrinsic33() # an ndarray in shape [3, 3]
intrinsic33 = pinhole_param.get_intrinsic() # a nested list in shape [3, 3]
intrinsic44 = pinhole_param.get_intrinsic(4) # a nested list in shape [4, 4]
# extrinsic
rotation_mat = pinhole_param.get_extrinsic_r() # a nested list in shape [3, 3]
translation_vec = pinhole_param.get_extrinsic_t() # a list whose length is 3
```

3.2 Fisheye

A fisheye lens is an ultra wide-angle lens that produces strong visual distortion intended to create a wide panoramic or hemispherical image (from [Wikipedia](#)). In XRPrimer, it's a sub-class of class `PinholeCameraParameter`.

3.2.1 Attributes

Here are additional attributes of a `FisheyeCameraParameter`. There are 6 parameters for radial distortion (k1-k6) and 2 parameters for tangential distortion (p1-p2).

3.2.2 Set distortion coefficients

a. Set all the coefficients.

```
fisheye_param.set_dist_coeff(dist_coeff_k=[k1, k2, k3, k4, k5, k6], dist_coeff_p=[p1, ↵
↵p2])
```

b. Set all the first four ks, k5 and k6 will keep their value.

```
fisheye_param.set_dist_coeff(dist_coeff_k=[k1, k2, k3, k4], dist_coeff_p=[p1, p2])
```

3.2.3 Get attributes

```
# distortion coefficients in opencv sequence
dist_coeff_list = fisheye_param.get_dist_coeff() # a list of float, k1, k2, p1, p2, k3, ↵
↵k4, k5, k6
```

IMAGE

This file introduces the supported image data structure in C++. It is an extension of OpenCV Mat, and also provides a way to convert between OpenCV Mat and Image.

4.1 Attributes

Here are attributes of class Image.

Besides the normal attributes for an image, it defines attributes like `timestamp` which is convenient for algorithms like SLAM.

4.2 Create an Image

Note that Image follows the order (width, height).

```
// create a color image with w=20 and h=10
Image img(20, 10, RGB24);
```

4.3 From Image to OpenCV

```
int width = 20;
int height = 10;
Image img(width, height, BGR24);

// Image to OpenCV
cv::Mat mat_warpper(img.height(), img.width(), CV_8UC3, img.mutable_data());
```

4.4 From OpenCV to Image

```
int width = 20;
int height = 10;
cv::Mat black = cv::Mat::zeros(height, width, CV_8UC3);
cv::imwrite("black.bmp", black);

// OpenCV to Image
Image i_black(black.cols, black.rows, black.step, BGR24, black.data);
```


This file introduces the supported pose data structure in C++. Generally, pose consists of a rotation and position.

5.1 Attributes

Here are attributes of class Pose.

5.2 Construct a Pose

Construct a pose with default value, where rotation is identity matrix and position is zero.

```
Pose pose;
```

Construct a pose with rotation and position. Rotation can be represented as quaternion, axis angle or rotation matrix.

```
Eigen::Vector3d vec3d;  
  
Eigen::Quaterniond quaternion;  
Pose pose1(quaternion.setIdentity(), vec3d.setZero());  
  
Eigen::AngleAxisd angleAxis(30, Eigen::Vector3d::UnitY());  
Pose pose2(angleAxis, vec3d.setZero());  
  
Eigen::Matrix3d mat3d;  
Pose pose3(mat3d.setIdentity(), vec3d.setZero());
```

5.3 Set Pose to identity

A identity pose denotes to identity rotation and zero position

```
pose.SetIdentity();
```

5.4 Scale a Pose

Scale a pose means multiplying scaling factor with the position.

```
auto p = pose3.Scale(1.2);  
pose3.ScaleMutable(1.4);
```

5.5 Inverse a Pose

Inverse a pose is defined as (1) applying rotation inversion and (2) multiplying position with inversed rotation.

```
pose.Inverse();  
pose.InverseMutable();
```

TRIANGULATOR

- Prepare camera parameters
- Build a triangulator
- Triangulate points from 2D to 3D
- Get reprojection error
- Camera selection

6.1 Prepare camera parameters

A triangulator requires a list of camera parameters to triangulate points. Each camera parameter should be an instance of `PinholeCameraParameter` or its sub-class. There are several ways to create the camera parameter list.

a. Assign camera parameters manually.

```
from xrprimer.data_structure.camera import PinholeCameraParameter

cam_param_list = []
for kinect_index in range(n_view):
    cam_param = PinholeCameraParameter(
        name=f'cam_{kinect_index:02d}',
        world2cam=True)
    cam_param.set_KRT(
        K=intrinsics[kinect_index],
        R=rotations[kinect_index],
        T=translations[kinect_index])
    cam_param_list.append(cam_param)
```

b. Load dumped camera parameter files.

```
from xrprimer.data_structure.camera import PinholeCameraParameter

cam_param_list = []
for kinect_index in range(n_view):
    cam_param_path = os.path.join(input_dir,
                                   f'cam_{kinect_index:02d}.json')
    cam_param = PinholeCameraParameter()
    cam_param.load(cam_param_path)
    cam_param_list.append(cam_param)
```

Note that `convention` and `world2cam` shall be set correctly. It is essential for the triangulator to know how to deal with input parameters.

6.2 Build a triangulator

In XRprimer, we use registry and builder to build a certain triangulator among multiple alternative classes.

```
import mmcv

from xrprimer.ops.triangulation.builder import build_triangulator

triangulator_config = dict(
    mmcv.Config.fromfile(
        'config/ops/triangulation/opencv_triangulator.py'))
triangulator_config['camera_parameters'] = cam_param_list
triangulator = build_triangulator(triangulator_config)
```

6.3 Set cameras of a triangulator

Camera parameters can also be set after building.

```
triangulator.set_cameras(cam_param_list)
```

6.4 Triangulate points from 2D to 3D

If there's only one point in 3D space, we could use `triangulate_single_point()`.

```
# points2d in shape [n_view, 2], in type numpy.ndarray, or nested list/tuple
point3d = triangulator.triangulate_single_point(points2d)
# points3d in shape [3, ], in type numpy.ndarray
```

For more than one point, `triangulate()` is recommended.

```
# points2d in shape [n_view, n_point, 2], in type numpy.ndarray, or nested list/tuple
point3d = triangulator.triangulate(points2d)
# points3d in shape [n_point, 3], in type numpy.ndarray
```

In multi-view scenario, not every view is helpful. To filter the good sources in 2D space, `points_mask` is introduced.

```
# points_mask in shape [n_view, n_point 1]
#           point0           point1           point2
# view0           0           nan           1
# view1           1           nan           1
# view2           1           nan           1
# result      combine 1,2      nan      combine 0,1,2
point3d = triangulator.triangulate_single_point(points2d, points_mask)
```

6.5 Get reprojection error

To evaluate the triangulation quality, we also provide a point-wise reprojection error, between input points2d and projected points2d. points_mask is also functional here.

```
point3d = triangulator.triangulate(points2d, points_mask)
error2d = triangulator.get_reprojection_error(points2d, points3d, points_mask)
# error2d has the same shape as points2d
```

6.6 Camera selection

To select a sub-set of all the cameras, we provide a selection method.

```
# select two cameras by index list
sub_triangulator = triangulator[[0, 1]]
# a tuple argument is same as list
sub_triangulator = triangulator[(0, 1)]
# select the first 3 cameras by slice
sub_triangulator = triangulator[:3]
# select cameras whose index is divisible by 2
sub_triangulator = triangulator[::2]
```

6.7 Get a conjugated projector

The returned multi-view projector will have the same cameras as triangulator.

```
projector = triangulator.get_projector()
```


PROJECTOR

- Prepare camera parameters
- Build a triangulator
- Triangulate points from 2D to 3D
- Get reprojection error
- Camera selection

7.1 Prepare camera parameters

A multi-view projector requires a list of camera parameters, just like the triangulator. Each camera parameter should be an instance of `PinholeCameraParameter` or its sub-class. For details, please refer to triangulator doc .

7.2 Build a projector

In XRprimer, we use registry and builder to build a certain projector among multiple alternative classes.

```
import mmcv

from xrprimer.ops.projection.builder import build_projector

projector_config = dict(
    mmcv.Config.fromfile(
        'config/ops/triangulation/opencv_projector.py'))
projector_config['camera_parameters'] = cam_param_list
projector_config = build_projector(projector_config)
```

7.3 Set cameras of a projector

Camera parameters can also be set after building.

```
projector.set_cameras(cam_param_list)
```

7.4 Project points from 3D to 2D

If there's only one point in 3D space, we could use `project_single_point()`.

```
# points3d in shape [3, ], in type numpy.ndarray, or list/tuple  
mview_point2d = projector.project_single_point(point3d)  
# mview_point2d in shape [n_view, 2], in type numpy.ndarray
```

For more than one point, `project()` is recommended.

```
# points3d in shape [n_point, 3], in type numpy.ndarray, or nested list/tuple  
points2d = triangulator.triangulate(points3d)  
# points2d in shape [n_view, n_point, 2], in type numpy.ndarray
```

In multi-view scenario, if we set the value at `points_mask[p_idx]` to zero, the point will not be projected.

```
points2d = triangulator.project(points3d, points_mask)
```

7.5 Camera selection

To select a sub-set of all the cameras, we provide a selection method. For details, please refer to `triangulator` doc .

CAMERA CONVENTION

8.1 Intrinsic convention

In OpenCV, shape of the intrinsic matrix is 3x3, while in some other system it's 4x4. In XRPrimer data structures, we store intrinsic in 4x4 manner, but you can get 3x3 intrinsic matrix by argument of get method. Here are the differences between intrinsic33 and intrinsic44.

Intrinsic33, only for perspective camera:

```
[[fx, 0, px],  
 [0, fy, py],  
 [0, 0, 1]]
```

Intrinsic44, perspective camera:

```
[[fx, 0, px, 0],  
 [0, fy, py, 0],  
 [0, 0, 0, 1],  
 [0, 0, 1, 0]]
```

Intrinsic44, orthographic camera:

```
[[fx, 0, 0, px],  
 [0, fy, 0, py],  
 [0, 0, 1, 0],  
 [0, 0, 0, 1]]
```

We can convert between intrinsic33 and intrinsic44 by `upgrade_k_3x3()`, `downgrade_k_4x4()`:

```
from xrprimer.transform.convention.camera import downgrade_k_4x4, upgrade_k_3x3  
  
intrinsic44 = upgrade_k_3x3(intrinsic33, is_perspective=True) # intrinsic33 in shape [3, 3] or [batch_size, 3, 3]  
intrinsic33 = downgrade_k_4x4(intrinsic44) # intrinsic44 in shape [4, 4] or [batch_size, 4, 4]
```

8.2 Extrinsic convention

In OpenCV camera space, a camera looks at Z+ of , screen right is X+ and screen up is Y-. However, not all the cameras are defined this way. We offer you a conversion method, converting a camera from one system to another.

For example, in order to convert an OpenCV camera into a Blender camera, call `convert_camera_parameter()`, and the direction of extrinsic (world2cam or cam2world) will not be changed.

```
from xrprimer.transform.convention.camera import convert_camera_parameter

blender_pinhole_param = convert_camera_parameter(pinhole_param, dst='blender')
```

Here is a sheet of supported camera conventions:

CALIBRATE MULTIPLE CAMERAS

TBA

RUNNING TESTS

- Data Preparation
- Environment Preparation
- Running tests through pytest

10.1 Data Preparation

Download data from the file server, and extract files to `python/tests/data`.

```
cd python/tests
wget -q https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrprimer/xrprimer.tar.gz
tar -xzf xrprimer.tar.gz && rm xrprimer.tar.gz
cp -r xrprimer/tests/data ./
rm -rf xrprimer && cd ../../
```

10.2 Environment Preparation

Install packages for test.

```
pip install -r requirements/test.txt
```

10.3 Running tests through pytest

Running all the tests below `python/tests`. It is a good way to validate whether `XRPrimer` has been correctly installed:

```
cd python
pytest tests/
cd ..
```

Generate a coverage for the test:

```
cd python
coverage run --source xrprimer -m pytest tests/
coverage xml
coverage report -m
cd ..
```


FREQUENTLY ASKED QUESTIONS

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them. If the contents here do not cover your issue, do not hesitate to create an issue!

11.1 Installation

- ‘ImportError: libpng16.so.16: cannot open shared object file: No such file or directory’
 1. If using conda, `conda install -c anaconda libpng`
 2. If sudo is available, `apt update & apt -y install libpng16-16`
- ‘ImportError: liblapack.so.3: cannot open shared object file: No such file or directory’
 1. If using conda, `conda install -c conda-forge lapack`
 2. If sudo is available, `apt update & apt -y install libatlas-base-dev`

CHANGELOG

12.1 v0.6.0 (01/09/2022/)

Highlights

- Support iOS and Linux compilation
- Support installation via pypi, ranging from python 3.6 to 3.10
- Support various camera models (Pinhole, Fisheye, Omni etc.)
- Support basic 3D operations (Triangulator, Projector etc.)
- Support Multi-camera extrinsic calibration tools

New Features

- Add pybind to create Python bindings of C++ data structures and switch python backend to C++ code
- Add camera convention convert method
- Add camera calibrator in python to support 3 types of calibration
- Add image class and support the conversion with OpenCV
- Add external deps and use conan manager to accelerate the compilation
- Provide samples to demonstrate linking XRPrimer in other C++ projects

LICENSE

The license of our codebase is Apache-2.0. Note that this license only applies to code in our library, the dependencies of which are separate and individually licensed. We would like to pay tribute to open-source implementations to which we rely on. Please be aware that using the content of dependencies may affect the license of our codebase. The license of our codebase and all external licenses are attached below.

XRMoCap is licensed for use as follows:

Copyright 2022 XRPrimer Authors. All rights reserved.

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation,

(continues on next page)

(continued from previous page)

and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct

(continues on next page)

(continued from previous page)

or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

(continues on next page)

(continued from previous page)

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

(continues on next page)

(continued from previous page)

Copyright 2022 XRPrimer Authors.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and

The XRPrimer license applies to all parts of XRPrimer that are not externally maintained libraries.

C++ DOCUMENTATION

C++ Doxygen documentation

XRPRIMER.DATA_STRUCTURE

15.1 camera

class `xrprimer.data_structure.camera.BaseCameraParameter`(*args: Any, **kwargs: Any)

LoadFile(filename: str) → bool

Load camera name and parameters from a dumped json file.

Parameters **filename** (str) – Path to the dumped json file.

Returns **bool** – True if load succeed.

SaveFile(filename: str) → int

Dump camera name and parameters to a json file.

Parameters **filename** (str) – Path to the dumped json file.

Returns **int** – returns 0.

clone() → *xrprimer.data_structure.camera.camera.BaseCameraParameter*

Clone a new CameraPrameter instance like self.

Returns **BaseCameraParameter**

dump(filename: str) → None

Dump camera name and parameters to a json file.

Parameters **filename** (str) – Path to the dumped json file.

Raises **RuntimeError** – Fail to dump a json file.

classmethod fromfile(filename: str) → *xrprimer.data_structure.camera.camera.BaseCameraParameter*

Construct a camera parameter data structure from a json file.

Parameters **filename** (str) – Path to the dumped json file.

Returns **CameraParameter** – An instance of CameraParameter class.

get_extrinsic_r() → list

Get extrinsic rotation matrix.

Returns **list** – Nested list of float32, 3x3 R mat.

get_extrinsic_t() → list

Get extrinsic translation vector.

Returns **list** – Nested list of float32, T vec of length 3.

get_intrinsic(*k_dim: int = 3*) → list

Get intrinsic K matrix.

Parameters **k_dim** (*int, optional*) – If 3, returns a 3x3 mat. Else if 4, returns a 4x4 mat. Defaults to 3.

Raises **ValueError** – k_dim is neither 3 nor 4.

Returns **list** – Nested list of float32, 4x4 or 3x3 K mat.

intrinsic33() → numpy.ndarray

Get an intrinsic matrix in shape (3, 3).

Returns **ndarray** – An ndarray of intrinsic matrix.

inverse_extrinsic() → None

Inverse the direction of extrinsics, between world to camera and camera to world.

load(*filename: str*) → None

Load camera name and parameters from a dumped json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Raises

- **FileNotFoundError** – File not found at filename.
- **ValueError** – Content in filename is not correct.

set_KRT(*K: Optional[Union[list, numpy.ndarray]] = None, R: Optional[Union[list, numpy.ndarray]] = None, T: Optional[Union[list, numpy.ndarray]] = None, world2cam: Optional[bool] = None*) → None

Set K, R to matrix and T to vector.

Parameters

- **K** (*Union[list, np.ndarray, None]*) – Nested list of float32, 4x4 or 3x3 K mat. Defaults to None, intrinsic will not be changed.
- **R** (*Union[list, np.ndarray, None]*) – Nested list of float32, 3x3 R mat. Defaults to None, extrinsic_r will not be changed.
- **T** (*Union[list, np.ndarray, None]*) – List of float32, T vector. Defaults to None, extrinsic_t will not be changed.
- **world2cam** (*Union[bool, None], optional*) – Whether the R, T transform points from world space to camera space. Defaults to None, self.world2cam will not be changed.

set_intrinsic(*mat3x3: Optional[Union[list, numpy.ndarray]] = None, width: Optional[int] = None, height: Optional[int] = None, fx: Optional[float] = None, fy: Optional[float] = None, cx: Optional[float] = None, cy: Optional[float] = None, perspective: bool = True*) → None

Set the intrinsic of a camera. Note that mat3x3 has a higher priority than fx, fy, cx, cy.

Parameters

- **mat3x3** (*list, optional*) – A nested list of intrinsic matrix, in shape (3, 3). If mat is given, fx, fy, cx, cy will be ignored. Defaults to None.
- **width** (*int*) – Width of the screen.
- **height** (*int*) – Height of the screen.
- **fx** (*float, optional*) – Focal length. Defaults to None.
- **fy** (*float, optional*) – Focal length. Defaults to None.
- **cx** (*float, optional*) – Camera principal point. Defaults to None.

- **cy** (*float, optional*) – Camera principal point. Defaults to None.
- **perspective** (*bool, optional*) – Whether it is a perspective camera, if not, it's orthographics. Defaults to True.

set_resolution(*height: int, width: int*) → None
Set resolution of the camera.

Parameters

- **height** (*int*) – Height of the screen.
- **width** (*int*) – Width of the screen.

class xrprimer.data_structure.camera.**FisheyeCameraParameter**(*args: Any, **kwargs: Any)

LoadFile(*filename: str*) → bool
Load camera name and parameters from a dumped json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if load succeed.

SaveFile(*filename: str*) → bool
Dump camera name and parameters to a json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if save succeed.

clone() → *xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter*
Clone a new CameraPrameter instance like self.

Returns **FisheyeCameraParameter**

get_dist_coeff() → list
Get distortion coefficients in self.convention.

Raises **NotImplementedError** – convention not supported.

Returns

- **list** – A list of distortion coefficients, in a
- **turn defined by self.convention.**

set_dist_coeff(*dist_coeff_k: list, dist_coeff_p: list*) → None
Set distortion coefficients from list.

Parameters

- **dist_coeff_k** (*list*) – List of float. [k1, k2, k3, k4, k5, k6]. When length of list is n and n<6, only the first n coefficients will be set.
- **dist_coeff_p** (*list*) – List of float. [p1, p2]. To set only p1, pass [p1].

class xrprimer.data_structure.camera.**OmniCameraParameter**(*args: Any, **kwargs: Any)

LoadFile(*filename: str*) → bool
Load camera name and parameters from a dumped json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if load succeed.

SaveFile(*filename: str*) → bool

Dump camera name and parameters to a json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if save succeed.

clone() → *xrprimer.data_structure.camera.omni_camera.OmniCameraParameter*

Clone a new CameraPrameter instance like self.

Returns **PinholeCameraParameter**

set_dist_coeff(*dist_coeff_k: list, dist_coeff_p: list*) → None

Set distortion coefficients from list.

Parameters

- **dist_coeff_k** (*list*) – List of float. [k1, k2, k3, k4, k5, k6]. When length of list is n and n<6, only the first n coefficients will be set.
- **dist_coeff_p** (*list*) – List of float. [p1, p2]. To set only p1, pass [p1].

set_omni_param(*xi: Optional[float] = None, D: Optional[list] = None*) → None

Set omni parameters.

Parameters

- **xi** (*Union[float, None], optional*) – Omni parameter xi. Defaults to None, xi will not be modified.
- **D** (*Union[list, None], optional*) – List of float. [D0, D1, D2, D3]. When length of list is n and n<4, only the first n parameters will be set. Defaults to None, D will not be modified.

class *xrprimer.data_structure.camera.PinholeCameraParameter*(*args: Any, **kwargs: Any)

LoadFile(*filename: str*) → bool

Load camera name and parameters from a dumped json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if load succeed.

SaveFile(*filename: str*) → bool

Dump camera name and parameters to a json file.

Parameters **filename** (*str*) – Path to the dumped json file.

Returns **bool** – True if save succeed.

clone() → *xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter*

Clone a new CameraParameter instance like self.

Returns **PinholeCameraParameter**

XRPRIMER.CALIBRATION

```
class xrprimer.calibration.BaseCalibrator(work_dir: str = '/temp', logger: Union[None, str,  
                                         logging.Logger] = None)
```

Base class of camera calibrator.

calibrate() → *xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter*

Calibrate a camera or several cameras. Input args shall not be modified and the calibrated camera will be returned.

Returns *PinholeCameraParameter* – The calibrated camera.

```
class xrprimer.calibration.MviewFisheyeCalibrator(chessboard_width: int, chessboard_height: int,  
                                                  chessboard_square_size: int, work_dir: str,  
                                                  calibrate_intrinsic: bool = False,  
                                                  calibrate_distortion: bool = False,  
                                                  calibrate_extrinsic: bool = True, logger:  
                                                  Union[None, str, logging.Logger] = None)
```

Multi-view extrinsic calibrator for distorted fisheye cameras.

calibrate(frames: List[List[str]], fisheye_param_list:
 List[xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter]) →
 List[xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter]

Calibrate multi-FisheyeCameraParameters with a chessboard. It takes intrinsics and distortion coefficients from fisheye_param_list, calibrates only extrinsics on undistorted frames.

Parameters

- **frames** (*List[List[str]]*) – A nested list of distorted image paths. The shape is [n_frame, n_view], and each element is the path to an image file. ‘’ stands for an empty image.
- **fisheye_param_list** (*List[FisheyeCameraParameter]*) – A list of FisheyeCameraParameters. Intrinsic matrix and distortion coefficients are necessary for calibration.

Returns *List[FisheyeCameraParameter]* – A list of calibrated fisheye cameras, name, logger, resolution will be kept.

```
class xrprimer.calibration.MviewPinholeCalibrator(chessboard_width: int, chessboard_height: int,  
                                                  chessboard_square_size: int, calibrate_intrinsic:  
                                                  bool = False, calibrate_extrinsic: bool = True,  
                                                  logger: Union[None, str, logging.Logger] = None)
```

Multi-view extrinsic calibrator for pinhole cameras.

calibrate(frames: List[List[str]], pinhole_param_list:
 List[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter]) →
 List[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter]

Calibrate multi-PinholeCameraParameters with a chessboard.

Parameters

- **frames** (*List[List[str]*) – A nested list of image paths. The shape is [n_frame, n_view], and each element is the path to an image file. “” stands for an empty image.
- **pinhole_param_list** (*List[PinholeCameraParameter]*) – A list of PinholeCameraParameters. Intrinsic matrix is necessary for calibration.

Returns *List[PinholeCameraParameter]* – A list of calibrated pinhole cameras, name, logger, resolution will be kept.

```
class xrprimer.calibration.SviewFisheyeDistortionCalibrator(chessboard_width: int,  
                                                            chessboard_height: int, logger:  
                                                            Union[None, str, logging.Logger] =  
                                                            None)
```

Single-view distortion calibrator for distorted fisheye camera.

It takes an init intrinsic, fix it and calibrate distortion coefficients.

```
calibrate(frames: List[str], fisheye_param:  
          xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter) →  
          xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter
```

Calibrate FisheyeCameraParameter with a chessboard. It takes intrinsics from fisheye_param, calibrates only distortion coefficients on undistorted frames.

Parameters

- **frames** (*List[str]*) – A list of distorted image paths.
- **fisheye_param** (*FisheyeCameraParameter*) – An instance of FisheyeCameraParameter. Intrinsic matrix is necessary for calibration, and the input instance will not be modified.

Returns *FisheyeCameraParameter* – An instance of FisheyeCameraParameter. Distortion coefficients are the only difference from input.

XRPRIMER.OPS

17.1 projection

```
class xrprimer.ops.projection.BaseProjector(camera_parameters:
    List[Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
    str]], logger: Union[None, str, logging.Logger] = None)

    BaseProjector for points projection.

    project(points: Union[numpy.ndarray, list, tuple], points_mask: Optional[Union[numpy.ndarray, list,
    tuple]] = None) → numpy.ndarray
        Project points with self.camera_parameters.

        Parameters
        • points(Union[np.ndarray, list, tuple]) – An ndarray or a nested list of points3d,
          in shape [n_point, 3].
        • points_mask(Union[np.ndarray, list, tuple], optional) – An ndarray or a
          nested list of mask, in shape [n_point, 1]. If points_mask[index] == 1, points[index] is
          valid for projection, else it is ignored. Defaults to None.

        Returns np.ndarray – An ndarray of points2d, in shape [n_view, n_point, 2].

    project_single_point(points: Union[numpy.ndarray, list, tuple]) → numpy.ndarray
        Project a single point with self.camera_parameters.

        Parameters points(Union[np.ndarray, list, tuple]) – An ndarray or a list of points3d,
          in shape [3].

        Returns np.ndarray – An ndarray of points2d, in shape [n_view, 2].

    set_cameras(camera_parameters:
        List[Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
        xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter]]) → None
        Set cameras for this projector.

        Parameters camera_parameters (List[Union[PinholeCameraParameter, str]]) – A
          list of PinholeCameraParameter or FisheyeCameraParameter.

class xrprimer.ops.projection.OpencvProjector(camera_parameters:
    List[xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter],
    logger: Union[None, str, logging.Logger] = None)

    Projector for points projection, powered by OpenCV.

    project(points: Union[numpy.ndarray, list, tuple], points_mask: Optional[Union[numpy.ndarray, list,
    tuple]] = None) → numpy.ndarray
        Project points with self.camera_parameters.
```

Parameters

- **points** (*Union*[*np.ndarray*, *list*, *tuple*]) – An ndarray or a nested list of points3d, in shape [n_point, 3].
- **points_mask** (*Union*[*np.ndarray*, *list*, *tuple*], *optional*) – An ndarray or a nested list of mask, in shape [n_point, 1]. If points_mask[index] == 1, points[index] is valid for projection, else it is ignored. Defaults to None.

Returns *np.ndarray* – An ndarray of points2d, in shape [n_view, n_point, 2].

project_single_point(*points: Union*[*numpy.ndarray*, *list*, *tuple*]) → *numpy.ndarray*

Project a single point with self.camera_parameters.

Parameters **points** (*Union*[*np.ndarray*, *list*, *tuple*]) – An ndarray or a list of points3d, in shape [3].

Returns *np.ndarray* – An ndarray of points2d, in shape [n_view, 2].

17.2 triangulation

class *xrprimer.ops.triangulation.BaseTriangulator*(*camera_parameters:*

List[*Union*[*xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter*, *str*]], *logger: Union*[*None*, *str*, *logging.Logger*] = *None*)

BaseTriangulator for points triangulation.

get_reprojection_error(*points2d: Union*[*numpy.ndarray*, *list*, *tuple*], *points3d: Union*[*numpy.ndarray*, *list*, *tuple*], *points_mask: Optional*[*Union*[*numpy.ndarray*, *list*, *tuple*]] = *None*) → *numpy.ndarray*

Get reprojection error between reprojected points2d and input points2d.

Parameters

- **points2d** (*Union*[*np.ndarray*, *list*, *tuple*]) – An ndarray or a nested list of points2d, in shape [n_view, n_point, 2].
- **points3d** (*Union*[*np.ndarray*, *list*, *tuple*]) – An ndarray or a nested list of points3d, in shape [n_point, 3].
- **points_mask** (*Union*[*np.ndarray*, *list*, *tuple*], *optional*) – An ndarray or a nested list of mask, in shape [n_view, n_point, 1]. If points_mask[index] == 1, points[index] is valid for triangulation, else it is ignored. If points_mask[index] == np.nan, the whole pair will be ignored and not counted by any method. Defaults to None.

Returns *np.ndarray* – An ndarray in shape [n_view, n_point, 2], record offset alone x, y axis of each point2d.

set_cameras(*camera_parameters:*

List[*Union*[*xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter*, *xrprimer.data_structure.camera.fisheye_camera.FisheyeCameraParameter*]]) → *None*

Set cameras for this triangulator.

Parameters **camera_parameters** (*List*[*Union*[*PinholeCameraParameter*, *str*]]) – A list of *PinholeCameraParameter* or *FisheyeCameraParameter*.

Raises **NotImplementedError** – Some camera_parameter from camera_parameters has a different camera convention from class requirement.

triangulate(*points*: Union[numpy.ndarray, list, tuple], *points_mask*: Optional[Union[numpy.ndarray, list, tuple]] = None) → numpy.ndarray

Triangulate points with self.camera_parameters.

Parameters

- **points** (Union[numpy.ndarray, list, tuple]) – An ndarray or a nested list of points2d, in shape [n_view, n_point 2].
- **points_mask** (Union[numpy.ndarray, list, tuple], optional) – An ndarray or a nested list of mask, in shape [n_view, n_point 1]. If points_mask[index] == 1, points[index] is valid for triangulation, else it is ignored. If points_mask[index] == np.nan, the whole pair will be ignored and not counted by any method. Defaults to None.

Returns numpy.ndarray – An ndarray of points3d, in shape [n_point, 3].

triangulate_single_point(*points*: Union[numpy.ndarray, list, tuple], *points_mask*: Optional[Union[numpy.ndarray, list, tuple]] = None) → numpy.ndarray

Triangulate a single point with self.camera_parameters.

Parameters

- **points** (Union[numpy.ndarray, list, tuple]) – An ndarray or a nested list of points2d, in shape [n_view, 2].
- **points_mask** (Union[numpy.ndarray, list, tuple], optional) – An ndarray or a nested list of mask, in shape [n_view, 1]. If points_mask[index] == 1, points[index] is valid for triangulation, else it is ignored. Defaults to None.

Returns numpy.ndarray – An ndarray of points3d, in shape [3,].

```
class xrprimer.ops.triangulation.OpencvTriangulator(camera_parameters:
                                                    List[xrprimer.data_structure.camera.fisheye_camera.FisheyeCamera],
                                                    multiview_reduction:
                                                    typing_extensions.Literal[mean, median] =
                                                    'mean', logger: Union[None, str,
                                                    logging.Logger] = None)
```

Triangulator for points triangulation, powered by OpenCV.

get_reprojection_error(*points2d*: Union[numpy.ndarray, list, tuple], *points3d*: Union[numpy.ndarray, list, tuple], *points_mask*: Optional[Union[numpy.ndarray, list, tuple]] = None) → numpy.ndarray

Get reprojection error between reprojected points2d and input points2d.

Parameters

- **points2d** (Union[numpy.ndarray, list, tuple]) – An ndarray or a nested list of points2d, in shape [n_view, n_point, 2].
- **points3d** (Union[numpy.ndarray, list, tuple]) – An ndarray or a nested list of points3d, in shape [n_point, 3].
- **points_mask** (Union[numpy.ndarray, list, tuple], optional) – An ndarray or a nested list of mask, in shape [n_view, n_point, 1]. If points_mask[index] == 1, points[index] is valid for triangulation, else it is ignored. If points_mask[index] == np.nan, the whole pair will be ignored and not counted by any method. Defaults to None.

Returns numpy.ndarray – An ndarray in shape [n_view, n_point, 2], record offset alone x, y axis of each point2d.

classmethod `prepare_triangulation_mat`(*camera_parameters:*
 List[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter]
 → numpy.ndarray

Prepare projection matrix for triangulation. According to opencv,

`ProjectionMatrix = [intrinsic33] * [extrinsic_r|extrinsic_t]`

Parameters **camera_parameters** (*List[PinholeCameraParameter]*) – A list of pinhole camera parameters.

Returns **np.ndarray** – The projection matrix in shape `[n_camera, 3, 4]`.

triangulate(*points: Union[numpy.ndarray, list, tuple], points_mask: Optional[Union[numpy.ndarray, list, tuple]] = None*) *→ numpy.ndarray*

Triangulate points with self.camera_parameters.

Parameters

- **points** (*Union[np.ndarray, list, tuple]*) – An ndarray or a nested list of points2d, in shape `[n_view, n_point 2]`.
- **points_mask** (*Union[np.ndarray, list, tuple], optional*) – An ndarray or a nested list of mask, in shape `[n_view, n_point 1]`. If `points_mask[index] == 1`, `points[index]` is valid for triangulation, else it is ignored. If `points_mask[index] == np.nan`, the whole pair will be ignored and not counted by any method. Defaults to None.

Returns **np.ndarray** – An ndarray of points3d, in shape `[n_point, 3]`.

triangulate_single_point(*points: Union[numpy.ndarray, list, tuple], points_mask:*
 Optional[Union[numpy.ndarray, list, tuple]] = None) *→ numpy.ndarray*

Triangulate a single point with self.camera_parameters.

Parameters

- **points** (*Union[np.ndarray, list, tuple]*) – An ndarray or a nested list of points2d, in shape `[n_view, 2]`.
- **points_mask** (*Union[np.ndarray, list, tuple], optional*) – An ndarray or a nested list of mask, in shape `[n_view, 1]`. If `points_mask[index] == 1`, `points[index]` is valid for triangulation, else it is ignored. Defaults to None.

Returns **np.ndarray** – An ndarray of points3d, in shape `[3,]`.

XRPRIMER.TRANSFORM

18.1 camera

`xrprimer.transform.camera.rotate_camera`(*cam_param*:
Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
xr-
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter],
rotation_mat: numpy.ndarray) →
Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
xr-
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter]

Apply rotation to a camera parameter.

Parameters

- **cam_param** (*Union[PinholeCameraParameter, FisheyeCameraParameter]*) – The camera to rotate.
- **rotation_mat** (*np.ndarray*) – Rotation matrix defined in world space, shape [3, 3].

Returns *Union[PinholeCameraParameter, FisheyeCameraParameter]* – Rotated camera in same type and extrinsic direction like the input camera.

`xrprimer.transform.camera.translate_camera`(*cam_param*:
Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
xr-
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter],
translation: numpy.ndarray) →
Union[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,
xr-
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter]

Apply the translation to a camera parameter.

Parameters

- **cam_param** (*Union[PinholeCameraParameter, FisheyeCameraParameter]*) – The camera to translate.
- **translation** (*np.ndarray*) – Translation vector defined in world space, shape [3,].

Returns *Union[PinholeCameraParameter, FisheyeCameraParameter]* – Translated camera in same type and extrinsic direction like the input camera.

```
xrprimer.transform.camera.undistort_camera(distorted_cam: xr-  
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter)  
→ xr-  
primer.data_structure.camera.pinhole_camera.PinholeCameraParameter
```

Undistort a `FisheyeCameraParameter` to `PinholeCameraParameter`.

Parameters `distorted_cam` (`FisheyeCameraParameter`) – An instance of `FisheyeCameraParameter`. Convention will be checked, resolution, intrinsic mat and distortion coefficients will be used.

Raises `NotImplementedError` – Camera convention not supported.

Returns `PinholeCameraParameter` – Undistorted camera parameter.

```
xrprimer.transform.camera.undistort_images(distorted_cam: xr-  
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter,  
image_array: numpy.ndarray) → Tu-  
ple[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,  
numpy.ndarray]
```

Undistort a `FisheyeCameraParameter` to `PinholeCameraParameter`, and undistort an array of images shot on a fisheye camera.

Parameters

- **`distorted_cam`** (`FisheyeCameraParameter`) – An instance of `FisheyeCameraParameter`. Convention will be checked, resolution, intrinsic mat and distortion coefficients will be used.
- **`image_array`** (*`np.ndarray`*) – An array of images, in shape `[n_frame, height, width, n_channel]`.

Raises `NotImplementedError` – Camera convention not supported.

Returns `Tuple[PinholeCameraParameter, np.ndarray]` –

`PinholeCameraParameter`: Undistorted camera parameter.

`np.ndarray`: Corrected images in the same shape as input.

```
xrprimer.transform.camera.undistort_points(distorted_cam: xr-  
primer.data_structure.camera.fisheye_camera.FisheyeCameraParameter,  
points: numpy.ndarray) → Tu-  
ple[xrprimer.data_structure.camera.pinhole_camera.PinholeCameraParameter,  
numpy.ndarray]
```

Undistort a `FisheyeCameraParameter` to `PinholeCameraParameter`, and undistort an array of points in fisheye camera screen. Parameters and points will be casted to `np.float64` before operation.

Parameters

- **`distorted_cam`** (`FisheyeCameraParameter`) – An instance of `FisheyeCameraParameter`. Convention will be checked, resolution, intrinsic mat and distortion coefficients will be used.
- **`points`** (*`np.ndarray`*) – An array of points, in shape `[..., 2]`, int or float. ... could be `[n_point,], [n_frame, n_point,], [n_frame, n_object, n_point,], etc.`

Raises `NotImplementedError` – Camera convention not supported.

Returns `Tuple[PinholeCameraParameter, np.ndarray]` –

`PinholeCameraParameter`: Undistorted camera parameter.

`np.ndarray`: Corrected points location in the same shape as input, dtype is `np.float64`.

18.2 camera convention

`xrprimer.transform.convention.camera.convert_camera_parameter`(*cam_param*: `xrprimer.data_structure.camera.camera.BaseCameraParameter`, *dst*: *str*) → `xrprimer.data_structure.camera.camera.BaseCameraParameter`

Convert a camera parameter instance into opencv convention.

Parameters

- **cam_param** (`BaseCameraParameter`) – The input camera parameter, which is an instance of `BaseCameraParameter` subclass.
- **dst** (*str*) – The name of destination convention.

Returns `BaseCameraParameter` – A camera in the same type as input, whose direction is same as *cam_param*, and convention equals to *dst*.

`xrprimer.transform.convention.camera.downgrade_k_4x4`(*k*: `numpy.ndarray`) → `numpy.ndarray`
Convert opencv 4x4 intrinsic matrix to 3x3.

Parameters **K** (`np.ndarray`) – Input 4x4 intrinsic matrix, left mm defined.

Returns `np.ndarray` – Output 3x3 intrinsic matrix, left mm defined.

[[fx, 0, px], [0, fy, py], [0, 0, 1]]

`xrprimer.transform.convention.camera.upgrade_k_3x3`(*k*: `numpy.ndarray`, *is_perspective*: *bool* = *True*) → `numpy.ndarray`

Convert opencv 3x3 intrinsic matrix to 4x4.

Parameters

- **K** (`np.ndarray`) – Input 3x3 intrinsic matrix, left mm defined.
[[fx, 0, px], [0, fy, py], [0, 0, 1]]
- **is_perspective** (*bool*, *optional*) – whether is perspective projection. Defaults to *True*.

Returns `np.ndarray` – Output intrinsic matrix.

for perspective: [[fx, 0, px, 0], [0, fy, py, 0], [0, 0, 0, 1], [0, 0, 1, 0]]

for orthographics: [[fx, 0, 0, px], [0, fy, 0, py], [0, 0, 1, 0], [0, 0, 0, 1]]

18.3 image

`xrprimer.transform.image.bgr2rgb`(*input_array*: `numpy.ndarray`, *color_dim*: *int* = -1) → `numpy.ndarray`
Convert image array of any shape between BGR and RGB.

Parameters

- **input_array** (`np.ndarray`) – An array of images. The shape could be: [h, w, n_ch], [n_frame, h, w, n_ch], [n_view, n_frame, h, w, n_ch], etc.
- **color_dim** (*int*, *optional*) – Which dim is the color channel. Defaults to -1.

Returns `np.ndarray`

XRPRIMER.UTILS

class `xrprimer.utils.Existence`(*value*)

State of file existence.

`xrprimer.utils.array_to_images`(*image_array*: *numpy.ndarray*, *output_folder*: *str*, *img_format*: *str* = '%06d.png', *resolution*: *Optional[Union[Tuple[int, int], Tuple[float, float]]]* = *None*, *disable_log*: *bool* = *False*, *logger*: *Union[None, str, logging.Logger]* = *None*) → *None*

Convert an array to images directly.

Parameters

- **image_array** (*np.ndarray*) – shape should be (f * h * w * 3).
- **output_folder** (*str*) – output folder for the images.
- **img_format** (*str*, *optional*) – format of the images. Defaults to '%06d.png'.
- **resolution** (*Optional[Union[Tuple[int, int], Tuple[float, float]]]*, *optional*) – (height, width) of the output images. Defaults to *None*.
- **disable_log** (*bool*, *optional*) – whether close the ffmpeg command info. Defaults to *False*.

Raises

- **FileNotFoundError** – check output folder.
- **TypeError** – check input array.

Returns

None

`xrprimer.utils.array_to_video`(*image_array*: *numpy.ndarray*, *output_path*: *str*, *fps*: *Union[int, float]* = 30, *resolution*: *Optional[Union[Tuple[int, int], Tuple[float, float]]]* = *None*, *disable_log*: *bool* = *False*, *logger*: *Union[None, str, logging.Logger]* = *None*) → *None*

Convert an array to a video directly, gif not supported.

Parameters

- **image_array** (*np.ndarray*) – shape should be (f * h * w * 3).
- **output_path** (*str*) – output video file path.
- **fps** (*Union[int, float]*, *optional*) – fps. Defaults to 30.
- **resolution** (*Optional[Union[Tuple[int, int], Tuple[float, float]]]*, *optional*) – (height, width) of the output video. Defaults to *None*.
- **disable_log** (*bool*, *optional*) – whether close the ffmpeg command info. Defaults to *False*.

Raises

- **FileNotFoundError** – check output path.
- **TypeError** – check input array.

Returns None.

```
xrprimer.utils.check_path(input_path: str, allowed_suffix: List[str] = [], allowed_existence:
                           List[xrprimer.utils.path_utils.Existence] = [<Existence.FileExist: 0>], path_type:
                           typing_extensions.Literal[file, dir, auto] = 'auto', logger: Union[None, str,
                           logging.Logger] = None) → None
```

Check both existence and suffix, raise error if check fails.

Parameters

- **input_path** (*str*) – Path to a file or folder.
- **allowed_suffix** (*List[str]*, *optional*) – What extension names are allowed. Offer a list like ['.jpg', '.jpeg']. When it's [], all will be received. Use [''] then directory is allowed. Defaults to [].
- **allowed_existence** (*List[Existence]*, *optional*) – What existence types are allowed. Defaults to [Existence.FileExist,].
- **path_type** (*Literal['file', 'dir', 'auto']*, *optional*) – What kind of file do we expect at the path. Choose among *file*, *dir*, *auto*. Defaults to 'auto'. Defaults to 'auto'.
- **logger** (*Union[None, str, logging.Logger]*, *optional*) – Logger for logging. If None, root logger will be selected. Defaults to None.

Raises

- **ValueError** – Wrong file suffix.
- **FileNotFoundError** – Wrong file existence.

```
xrprimer.utils.check_path_existence(path_str: str, path_type: typing_extensions.Literal[file, dir, auto] =
                                     'auto') → xrprimer.utils.path_utils.Existence
```

Check whether a file or a directory exists at the expected path.

Parameters

- **path_str** (*str*) – Path to check.
- **path_type** (*Literal['file', 'dir', 'auto']*, *optional*) – What kind of file do we expect at the path. Choose among *file*, *dir*, *auto*. Defaults to 'auto'.

Raises **KeyError** – if *path_type* conflicts with *path_str*

Returns **Existence** –

0. FileExist: file at path_str exists.
1. DirectoryExistEmpty: folder at path exists and.
2. DirectoryExistNotEmpty: folder at path_str exists and not empty.
3. MissingParent: its parent doesn't exist.
4. DirectoryNotExist: expect a folder at path_str, but not found.
5. FileNotExist: expect a file at path_str, but not found.

```
xrprimer.utils.check_path_suffix(path_str: str, allowed_suffix: Union[str, List[str]] = "") → bool
```

Check whether the suffix of the path is allowed.

Parameters

- **path_str** (*str*) – Path to check.
- **allowed_suffix** (*List[str]*, *optional*) – What extension names are allowed. Offer a list like ['.jpg', '.jpeg']. When it's [], all will be received. Use [''] then directory is allowed. Defaults to ''.

Returns **bool** – True: suffix test passed False: suffix test failed

`xrprimer.utils.get_logger(logger: Union[None, str, logging.Logger] = None) → logging.Logger`
Get logger.

Parameters **logger** (*Union[None, str, logging.Logger]*) – None for root logger. Besides, pass name of the logger or the logger itself. Defaults to None.

Returns **logging.Logger**

`xrprimer.utils.images_to_array(input_folder: str, resolution: Optional[Union[Tuple[int, int], Tuple[float, float]]) = None, img_format: str = '%06d.png', start: int = 0, end: Optional[int] = None, remove_raw_files: bool = False, disable_log: bool = False, logger: Union[None, str, logging.Logger] = None) → numpy.ndarray`

Read a folder of images as an array of (f * h * w * 3).

Parameters

- **input_folder** (*str*) – folder of input images.
- **resolution** (*Union[Tuple[int, int], Tuple[float, float]]*) – resolution(height, width) of output. Defaults to None.
- **img_format** (*str*, *optional*) – format of images to be read. Defaults to '%06d.png'.
- **start** (*int*, *optional*) – start frame index. Inclusive. If < 0, will be converted to frame_index range in [0, n_frame]. Defaults to 0.
- **end** (*int*, *optional*) – end frame index. Exclusive. Could be positive int or negative int or None. If None, all frames from start till the last frame are included. Defaults to None.
- **remove_raw_files** (*bool*, *optional*) – whether remove raw images. Defaults to False.
- **disable_log** (*bool*, *optional*) – whether close the ffmpeg command info. Defaults to False.

Raises **FileNotFoundError** – check the input path.

Returns **np.ndarray** – shape will be (f * h * w * 3).

`xrprimer.utils.images_to_array_opencv(input_folder: str, resolution: Optional[Union[Tuple[int, int], Tuple[float, float]]) = None, img_format: Optional[str] = None, start: int = 0, end: Optional[int] = None, logger: Union[None, str, logging.Logger] = None) → numpy.ndarray`

Read a folder of images as an array of (f * h * w * 3).

Parameters

- **input_folder** (*str*) – folder of input images.
- **resolution** (*Union[Tuple[int, int], Tuple[float, float]]*) – resolution(height, width) of output. Defaults to None.
- **img_format** (*str*, *optional*) – Format of images to be read, 'jpg' or 'png'. Defaults to None.

- **start** (*int*, *optional*) – start frame index. Inclusive. If < 0, will be converted to *frame_index* range in [0, *n_frame*]. Defaults to 0.
- **end** (*int*, *optional*) – end frame index. Exclusive. Could be positive *int* or negative *int* or *None*. If *None*, all frames from start till the last frame are included. Defaults to *None*.
- **logger** (*Union[None, str, logging.Logger]*, *optional*) – Logger for logging. If *None*, root logger will be selected. Defaults to *None*.

Raises **FileNotFoundError** – check the input path.

Returns **np.ndarray** – shape will be (f * h * w * 3).

`xrprimer.utils.images_to_sorted_images(input_folder, output_folder, img_format='%06d')`

Copy and rename a folder of images into a new folder following the *img_format*.

Parameters

- **input_folder** (*str*) – input folder.
- **output_folder** (*str*) – output folder.
- **img_format** (*str*, *optional*) – image format name, do not need extension. Defaults to '%06d'.

Returns **str** – image format of the rename images.

`xrprimer.utils.pad_for_libx264(image_array: numpy.ndarray) → numpy.ndarray`

Pad zeros if width or height of *image_array* is not divisible by 2. Otherwise you will get.

“[libx264 @ 0x1b1d560] width not divisible by 2 “

Parameters **image_array** (*np.ndarray*) – Image or images load by `cv2.imread()`. Possible shapes:

1. [height, width]
2. [height, width, channels]
3. [images, height, width]
4. [images, height, width, channels]

Returns **np.ndarray** – A image with both edges divisible by 2.

`xrprimer.utils.prepare_output_path(output_path: str, tag: str = 'output file', allowed_suffix: List[str] = [], path_type: typing_extensions.Literal[file, dir, auto] = 'auto', overwrite: bool = True, logger: Union[None, str, logging.Logger] = None) → None`

Check output folder or file.

Parameters

- **output_path** (*str*) – could be folder or file.
- **allowed_suffix** (*List[str]*, *optional*) – Check the suffix of *output_path*. If folder, should be [] or ['.']. If could both be folder or file, should be [suffixs..., '.']. Defaults to [].
- **tag** (*str*, *optional*) – The *string* tag to specify the output type. Defaults to 'output file'.
- **path_type** (*Literal[, optional]*) – Choose *file* for file and *dir* for folder. Choose *auto* if allowed to be both. Defaults to 'auto'.
- **overwrite** (*bool*, *optional*) – Whether overwrite the existing file or folder. Defaults to *True*.

Raises

- **FileNotFoundError** – suffix does not match.
- **FileExistsError** – file or folder already exists and *overwrite* is *False*.

Returns *None*

```
xrprimer.utils.setup_logger(logger_name: str = 'root', logger_level: int = 20, logger_path: Optional[str] =
                             None, logger_format: Optional[str] = None) → logging.Logger
```

Set up a logger.

Parameters

- **logger_name** (*str*, *optional*) – Name of the logger. Defaults to 'root'.
- **logger_level** (*int*, *optional*) – Set the logging level of this logger. Defaults to logging.INFO.
- **logger_path** (*str*, *optional*) – Path to the log file. Defaults to None, no file will be written, StreamHandler will be used.
- **logger_format** (*str*, *optional*) – The formatter for logger handler. Defaults to None.

Returns `logging.Logger` – A logger with settings above.

```
xrprimer.utils.video_to_array(input_path: str, resolution: Optional[Union[Tuple[int, int], Tuple[float,
float]]] = None, start: int = 0, end: Optional[int] = None, disable_log: bool
                             = False, logger: Union[None, str, logging.Logger] = None) →
                             numpy.ndarray
```

Read a video/gif as an array of (f * h * w * 3).

Parameters

- **input_path** (*str*) – input path.
- **resolution** (*Union[Tuple[int, int], Tuple[float, float]]*, *optional*) – resolution(height, width) of output. Defaults to None.
- **start** (*int*, *optional*) – start frame index. Inclusive. If < 0, will be converted to frame_index range in [0, n_frame]. Defaults to 0.
- **end** (*int*, *optional*) – end frame index. Exclusive. Could be positive int or negative int or None. If None, all frames from start till the last frame are included. Defaults to None.
- **disable_log** (*bool*, *optional*) – whether close the ffmpeg command info. Defaults to False.
- **logger** (*Union[None, str, logging.Logger]*, *optional*) – Logger for logging. If None, root logger will be selected. Defaults to None.

Raises `FileNotFoundError` – check the input path.

Returns `np.ndarray` – shape will be (f * h * w * 3).

INDICES AND TABLES

- `genindex`
- `search`

PYTHON MODULE INDEX

X

- `xrprimer.calibration`, 43
- `xrprimer.data_structure.camera`, 39
- `xrprimer.ops.projection`, 45
- `xrprimer.ops.triangulation`, 46
- `xrprimer.transform.camera`, 49
- `xrprimer.transform.convention.camera`, 51
- `xrprimer.transform.image`, 51
- `xrprimer.utils`, 53

INDEX

A

`array_to_images()` (in module `xrprimer.utils`), 53
`array_to_video()` (in module `xrprimer.utils`), 53

B

`BaseCalibrator` (class in `xrprimer.calibration`), 43
`BaseCameraParameter` (class in `xrprimer.data_structure.camera`), 39
`BaseProjector` (class in `xrprimer.ops.projection`), 45
`BaseTriangulator` (class in `xrprimer.ops.triangulation`), 46
`bgr2rgb()` (in module `xrprimer.transform.image`), 51

C

`calibrate()` (`xrprimer.calibration.BaseCalibrator` method), 43
`calibrate()` (`xrprimer.calibration.MviewFisheyeCalibrator` method), 43
`calibrate()` (`xrprimer.calibration.MviewPinholeCalibrator` method), 43
`calibrate()` (`xrprimer.calibration.SviewFisheyeDistortionCalibrator` method), 44
`check_path()` (in module `xrprimer.utils`), 54
`check_path_existence()` (in module `xrprimer.utils`), 54
`check_path_suffix()` (in module `xrprimer.utils`), 54
`clone()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39
`clone()` (`xrprimer.data_structure.camera.FisheyeCameraParameter` method), 41
`clone()` (`xrprimer.data_structure.camera.OmniCameraParameter` method), 42
`clone()` (`xrprimer.data_structure.camera.PinholeCameraParameter` method), 42
`convert_camera_parameter()` (in module `xrprimer.transform.convention.camera`), 51

D

`downgrade_k_4x4()` (in module `xrprimer.transform.convention.camera`), 51
`dump()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39

E

`Existence` (class in `xrprimer.utils`), 53

F

`FisheyeCameraParameter` (class in `xrprimer.data_structure.camera`), 41
`fromfile()` (`xrprimer.data_structure.camera.BaseCameraParameter` class method), 39

G

`get_dist_coeff()` (`xrprimer.data_structure.camera.FisheyeCameraParameter` method), 41
`get_extrinsic_r()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39
`get_extrinsic_t()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39
`get_intrinsic()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39
`get_logger()` (in module `xrprimer.utils`), 55
`get_reprojection_error()` (`xrprimer.ops.triangulation.BaseTriangulator` method), 46
`get_reprojection_error()` (`xrprimer.ops.triangulation.OpenCvTriangulator` method), 47
`images_to_array()` (in module `xrprimer.utils`), 55
`images_to_array_opencv()` (in module `xrprimer.utils`), 55
`images_to_sorted_images()` (in module `xrprimer.utils`), 56
`intrinsic33()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 40
`inverse_extrinsic()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 40

L

`load()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 40

`LoadFile()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39

`LoadFile()` (`xrprimer.data_structure.camera.FisheyeCameraParameter` method), 41

`LoadFile()` (`xrprimer.data_structure.camera.OmniCameraParameter` method), 41

`LoadFile()` (`xrprimer.data_structure.camera.PinholeCameraParameter` method), 42

M

module

`xrprimer.calibration`, 43

`xrprimer.data_structure.camera`, 39

`xrprimer.ops.projection`, 45

`xrprimer.ops.triangulation`, 46

`xrprimer.transform.camera`, 49

`xrprimer.transform.convention.camera`, 51

`xrprimer.transform.image`, 51

`xrprimer.utils`, 53

`MviewFisheyeCalibrator` (class in `xrprimer.calibration`), 43

`MviewPinholeCalibrator` (class in `xrprimer.calibration`), 43

O

`OmniCameraParameter` (class in `xrprimer.data_structure.camera`), 41

`OpencvProjector` (class in `xrprimer.ops.projection`), 45

`OpencvTriangulator` (class in `xrprimer.ops.triangulation`), 47

P

`pad_for_libx264()` (in module `xrprimer.utils`), 56

`PinholeCameraParameter` (class in `xrprimer.data_structure.camera`), 42

`prepare_output_path()` (in module `xrprimer.utils`), 56

`prepare_triangulation_mat()` (`xrprimer.ops.triangulation.OpencvTriangulator` class method), 47

`project()` (`xrprimer.ops.projection.BaseProjector` method), 45

`project()` (`xrprimer.ops.projection.OpencvProjector` method), 45

`project_single_point()` (`xrprimer.ops.projection.BaseProjector` method), 45

`project_single_point()` (`xrprimer.ops.projection.OpencvProjector` method), 46

R

`rotate_camera()` (in module `xrprimer.transform.camera`), 49

S

`SaveFile()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 39

`SaveFile()` (`xrprimer.data_structure.camera.FisheyeCameraParameter` method), 41

`SaveFile()` (`xrprimer.data_structure.camera.OmniCameraParameter` method), 41

`SaveFile()` (`xrprimer.data_structure.camera.PinholeCameraParameter` method), 42

`set_cameras()` (`xrprimer.ops.projection.BaseProjector` method), 45

`set_cameras()` (`xrprimer.ops.triangulation.BaseTriangulator` method), 46

`set_dist_coeff()` (`xrprimer.data_structure.camera.FisheyeCameraParameter` method), 41

`set_dist_coeff()` (`xrprimer.data_structure.camera.OmniCameraParameter` method), 42

`set_intrinsic()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 40

`set_KRT()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 40

`set_omni_param()` (`xrprimer.data_structure.camera.OmniCameraParameter` method), 42

`set_resolution()` (`xrprimer.data_structure.camera.BaseCameraParameter` method), 41

`setup_logger()` (in module `xrprimer.utils`), 56

`SviewFisheyeDistortionCalibrator` (class in `xrprimer.calibration`), 44

T

`translate_camera()` (in module `xrprimer.transform.camera`), 49

`triangulate()` (`xrprimer.ops.triangulation.BaseTriangulator` method), 46

`triangulate()` (`xrprimer.ops.triangulation.OpencvTriangulator` method), 48

`triangulate_single_point()` (`xrprimer.ops.triangulation.BaseTriangulator` method), 47

`triangulate_single_point()` (`xrprimer.ops.triangulation.OpencvTriangulator` method), 48

U

`undistort_camera()` (in module `xr-`

`primer.transform.camera`), 49
`undistort_images()` (in module `xr-`
`primer.transform.camera`), 50
`undistort_points()` (in module `xr-`
`primer.transform.camera`), 50
`upgrade_k_3x3()` (in module `xr-`
`primer.transform.convention.camera`), 51

V

`video_to_array()` (in module `xrprimer.utils`), 57

X

`xrprimer.calibration`
module, 43
`xrprimer.data_structure.camera`
module, 39
`xrprimer.ops.projection`
module, 45
`xrprimer.ops.triangulation`
module, 46
`xrprimer.transform.camera`
module, 49
`xrprimer.transform.convention.camera`
module, 51
`xrprimer.transform.image`
module, 51
`xrprimer.utils`
module, 53