XRMoCap

Release 0.5.0

XRMoCap Authors

Sep 14, 2022

GETTING STARTED

CHAPTER

ONE

INSTALLATION

- Requirements
- A from-scratch setup script
- Prepare environment
- Run with docker image
- Test environment
- Frequently Asked Questions

1.1 Requirements

- Linux
- ffmpeg
- Python 3.7+
- PyTorch 1.6.0, 1.7.0, 1.7.1, 1.8.0, 1.8.1, 1.9.0 or 1.9.1.
- CUDA 9.2+
- GCC 5+
- XRPrimer
- MMHuman3D
- MMCV

Optional:

1.2 A from-scratch setup script

```
conda create -n xrmocap python=3.8
source activate xrmocap

# install ffmpeg for video and images
conda install -y ffmpeg

# install pytorch
conda install -y pytorch==1.8.1 torchvision==0.9.1 cudatoolkit=10.1 -c pytorch
```

(continues on next page)

(continued from previous page)

```
# install pytorch3d
conda install -y -c fvcore -c iopath -c conda-forge fvcore iopath
conda install -y -c bottler nvidiacub
conda install -y pytorch3d -c pytorch3d

# install mmcv-full
pip install mmcv-full==1.5.3 -f https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.
↪1/index.html

# install xrprimer
pip install xrprimer

# clone xrmocap
git clone https://github.com/openxrlab/xrmocap.git
cd xrmocap

# install requirements for build
pip install -r requirements/build.txt
# install requirements for runtime
pip install -r requirements/runtime.txt

# install xrmocap
rm -rf .eggs && pip install -e .
```

1.3 Prepare environment

Here are advanced instructions for environment setup. If you have run A from-scratch setup script successfully, please skip this.

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

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

1.3.2 b. Install MMHuman3D.

Here we take `torch_version=1.8.1` and `cu_version=10.2` as example. For other versions, please follow the official instructions

```
# install ffmpeg from main channel
conda install ffmpeg
# install pytorch
conda install -y pytorch==1.8.1 torchvision==0.9.1 cudatoolkit=10.2 -c pytorch
# install pytorch3d
conda install -c fvcore -c iopath -c conda-forge fvcore iopath -y
conda install -c bottler nvidiacub -y
```

(continues on next page)

(continued from previous page)

```
conda install pytorch3d -c pytorch3d
# install mmcv-full for human_perception
pip install mmcv-full==1.5.3 -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.
↪1/index.html
# install mmhuman3d
pip install git+https://github.com/open-mmlab/mmhuman3d.git
```

Note1: Make sure that your compilation CUDA version and runtime CUDA version match.

Note2: The package `mmcv-full(gpu)` is essential if you are going to use `human_perception` modules.

Note3: Do not install optional requirements of mmhuman3d in this step.

1.3.3 c. Install XRPrimer.

```
pip install xrprimer
```

If you want to edit xrprimer, please follow the official instructions to install it from source.

1.3.4 d. Install XRMoCap to virtual environment, in editable mode.

```
git clone https://github.com/openxrlab/xrmocap.git
cd xrmocap
pip install -r requirements/build.txt
pip install -r requirements/runtime.txt
pip install -e .
```

1.3.5 e. Run unittests or demos

If everything goes well, try to run unittest or go back to run demos

1.3.6 Run with Docker Image

We provide a Dockerfile to build an image. Ensure that you are using docker version >=19.03 and `"default-runtime": "nvidia"` in daemon.json.

```
# build an image with PyTorch 1.8.1, CUDA 10.2
docker build -t xrmocap .
```

Run it with

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/xrmocap/data xrmocap
```

Or pull a built image from docker hub.

```
docker pull openxrlab/xrmocap_runtime
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/xrmocap/data openxrlab/xrmocap_
↪runtime
```

1.3.7 Test environment

To test whether the environment is well installed, please refer to *test doc*.

1.3.8 Frequently Asked Questions

If your environment fails, check our *FAQ* first, it might be helpful to some typical questions.

CHAPTER

TWO

DATASET PREPARATION

- Overview
- Supported datasets
- Download converted meta-data
- Convert a dataset manually
- Validate converted meta-data by visualization

2.1 Overview

Our data pipeline converts original dataset to our unified meta-data, with data converters controlled by configs.

2.2 Supported datasets

2.3 Download converted meta-data

Considering that it takes long to run a converter if perception2d is checked, we have done it for you. Our perception 2D is generated by mmtrack and mmpose, defined in coco_wholebody by default. You can download compressed zip file for converted meta-data below.

For where to put the downloaded meta-data, check xrmocap dataset structure for details.

For CMU panoptic meta-data, frames extracted from videos have been removed before uploading. One has to convert panoptic data locally with `bbox_detector = None` and `kps2d_estimator = None` first, and then copy download data into the converted meta-data directory.

2.4 Convert a dataset manually

Use our prepare_dataset tool to convert a dataset. See the tool tutorial for details.

2.5 Validate converted meta-data by visualization

Use our visualize_dataset tool to visualize meta-data. See the tool tutorial for details.

For CMU panoptic meta-data, the multi-view all-in-one video is too large to load, please set `vis_aio_video = False` to avoid OOM.

CHAPTER

THREE

GETTING STARTED

- Installation
- Data Preparation
- Body Model Preparation (Optional)
- Inference
- Evaluation
- Training
- More tutorials

3.1 Installation

Please refer to *installation.md* for installation.

3.2 Data Preparation

Please refer to *data_preparation.md* for data preparation.

3.3 Body Model Preparation (Optional)

If you want to obtain keypoints3d, the body model is not necessary. If you want to infer SMPL as well, you can prepare the body_model as follows.

- SMPL v1.0 is used in our experiments.
  - Neutral model can be downloaded from SMPLify.
  - All body models have to be renamed in `SMPL_{GENDER}.pkl` format. For example, `mv basicModel_neutral_lbs_10_207_0_v1.0.0.pkl SMPL_NEUTRAL.pkl`
- smpl_mean_params.npz
- gmm_08.zip from smplify-x repo
- gmm_08.pkl from openxrlab backup

Download the above resources and arrange them in the following file structure:

```
xrmocap
├── xrmocap
├── docs
├── tests
├── tools
├── configs
└── xrmocap_data
    └── body_models
        ├── gmm_08.pkl
        ├── smpl_mean_params.npz
        └── smpl
            ├── SMPL_FEMALE.pkl
            ├── SMPL_MALE.pkl
            └── SMPL_NEUTRAL.pkl
```

3.4 Inference

We provide a demo script to estimate SMPL parameters for single-person or multi-person from multi-view synchronized input images or videos. With this demo script, you only need to choose a method, we currently support two types of methods, namely, optimization-based approaches and end-to-end learning algorithms, specify a few arguments, and then you can get the estimated results.

We assume that the cameras have been calibrated. If you want to know more about camera calibration, refer to XRPrimer for more details.

3.4.1 Perception Model

Prepare perception models, including detection, 2d pose estimation, tracking and CamStyle models.

```
sh scripts/download_weight.sh
```

You could find perception models in `weight` file.

3.4.2 Single Person

Currently, we only provide optimization-based method for single person estimation.

1. Download body model. Please refer to Body Model Preparation
2. Download a 7z file from humman dataset.
3. Extract the 7z file.

```
cd xrmocap_data/humman
7z x p000127_a000007.7z
```

3. Run *process_smc* tool.

```
python tools/process_smc.py \
        --estimator_config configs/humman_mocap/mview_sperson_smpl_estimator.py \
        --smc_path xrmocap_data/humman/p000127_a000007.smc \
```

(continues on next page)

(continued from previous page)

```
        --output_dir xrmocap_data/humman/p000127_a000007_output \
        --visualize
```

3.4.3 Multiple People

A small test dataset for quick demo can be downloaded here. It contains 50 frames from the Shelf sequence, with 5 camera views calibrated and synchronized.

Optimization-based methods

For optimization-based approaches, it utilizes the association between 2D keypoints and generates 3D keypoints by triangulation or other methods. Taking MVPose as an example, it can be run as

1. Download data and body model

- download data

```
mkdir xrmocap_data
wget https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrmocap/example_resources/
↪Shelf_50.zip -P xrmocap_data
cd xrmocap_data/ && unzip -q Shelf_50.zip && rm Shelf_50.zip && cd ..
```

- download body model

Please refer to Body Model Preparation

2. Run demo

```
python tools/mview_mperson_topdown_estimator.py \
      --estimator_config 'configs/mvpose_tracking/mview_mperson_topdown_estimator.py' \
      --image_and_camera_param 'xrmocap_data/Shelf_50/image_and_camera_param.txt' \
      --start_frame 300 \
      --end_frame 350 \
      --output_dir 'output/estimation' \
      --enable_log_file
```

If all the configuration is OK, you could see the results in `output_dir`.

Learning-based methods

For learning-based methods, it resorts to an end-to-end learning scheme so as to require training before inference. Taking MvP as an example, we can download pretrained MvP model and run it on Shelf_50 as:

1. Install `Deformable` package

Download the `./ops` folder, rename and place the folder as `xrmocap/model/deformable`. Install `Deformable` by running:

```
cd xrmocap/model/deformable/
sh make.sh
```

2. Download data and run demo

```
# download data
mkdir -p xrmocap_data
wget https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrmocap/example_resources/
↪Shelf_50.zip -P xrmocap_data
cd xrmocap_data/ && unzip -q Shelf_50.zip && rm Shelf_50.zip && cd ..

# download pretrained model
mkdir -p weight/mvp
wget https://openxrlab-share.oss-cn-hongkong.aliyuncs.com/xrmocap/weight/mvp/xrmocap_mvp_
↪shelf-22d1b5ed_20220831.pth -P weight/mvp

sh ./scripts/eval_mvp.sh 1 configs/mvp/shelf_config/mvp_shelf_50.py weight/mvp/xrmocap_
↪mvp_shelf-22d1b5ed_20220831.pth
```

For detailed tutorials about dataset preparation, model weights and checkpoints download for learning-based methods, please refer to the *training tutorial* and *evaluation tutorial*.

3.5 Evaluation

3.5.1 Evaluate with a single GPU / multiple GPUs

Optimization-based methods

- Evaluate on the Shelf dataset and run the tool without tracking.

```
python tools/mview_mperson_evaluation.py \
      --enable_log_file \
      --evaluation_config configs/mvpose/shelf_config/eval_keypoints3d.py
```

- Evaluate on the Shelf dataset and run the tool with tracking.

```
python tools/mview_mperson_evaluation.py \
      --enable_log_file \
      --evaluation_config configs/mvpose_tracking/shelf_config/eval_keypoints3d.py
```

Learning-based methods

For learning-based methods, more details about dataset preparation, model weights and checkpoints download can be found at *evaluation tutorial*.

With the downloaded pretrained MvP models from *model_zoo*:

```
sh ./scripts/val_mvp.sh ${NUM_GPUS} ${CFG_FILE} ${MODEL_PATH}
```

Example:

```
sh ./scripts/val_mvp.sh 8 configs/mvp/shelf_config/mvp_shelf.py weight/xrmocap_mvp_shelf.
↪pth.tar
```

3.5.2 Evaluate with slurm

If you can run XRMoCap on a cluster managed with slurm, you can use the script `scripts/slurm_eval_mvp.sh`.

```
sh ./scripts/slurm_eval_mvp.sh ${PARTITION} ${NUM_GPUS} ${CFG_FILE} ${MODEL_PATH}
```

Example:

```
sh ./scripts/slurm_eval_mvp.sh ${PARTITION} 8 configs/mvp/shelf_config/mvp_shelf.py␣
↪weight/xrmocap_mvp_shelf.pth.tar
```

3.6 Training

Training is only applicable to learning-based methods.

3.6.1 Training with a single / multiple GPUs

To train the learning-based model, such as a MvP model, follow the *training tutorial* to prepare the datasets and pre-trained weights:

```
sh ./scripts/train_mvp.sh ${NUM_GPUS} ${CFG_FILE}
```

Example:

```
sh ./scripts/train_mvp.sh 8 configs/mvp/campus_config/mvp_campus.py
```

3.6.2 Training with Slurm

If you can run XRMoCap on a cluster managed with slurm, you can use the script `scripts/slurm_train_mvp.sh`.

```
sh ./scripts/slurm_train_mvp.sh ${PARTITION} ${NUM_GPUS} ${CFG_FILE}
```

Example:

```
sh ./scripts/slurm_train_mvp.sh ${PARTITION} 8 configs/mvp/shelf_config/mvp_shelf.py
```

3.7 More Tutorials

- *Introduction*
- *Config*
- *New dataset*
- *New module*

CHAPTER

FOUR

BENCHMARK AND MODEL ZOO

For optimization-based methods, we provide configuration files and log files. For learning-based methods, we provide configuration files, log files and pretrained models. Moreover, all supported methods are evaluated on three common benchmarks: Campus, Shelf, and CMU Panoptic.

4.1 Baselines

4.1.1 MVPose (Single frame)

Please refer to MVPose for details.

4.1.2 MVPose (Temporal tracking and filtering)

Please refer to MVPose with tracking for details.

4.1.3 Shape-aware 3D Pose Optimization

Please refer to Shape-aware 3D Pose Optimization for details.

4.1.4 MvP

Please refer to MvP benchmarks for details.

CHAPTER

FIVE

RUNNING TESTS

- Data Preparation
- Environment Preparation
- Running tests through pytest

5.1 Data Preparation

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

```
sh scripts/download_test_data.sh
```

Download weights from Internet, and extract files to `weight`.

```
sh scripts/download_weight.sh
```

5.2 Environment Preparation

Install packages for test.

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

5.3 Running tests through pytest

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

```
pytest tests/
```

Or generate a coverage when testing:

```
coverage run --source xrmocap -m pytest tests/
coverage xml
coverage report -m
```

Or starts a CPU-only test on a GPU machine:

```
export CUDA_VISIBLE_DEVICES=-1
pytest tests/
```

CHAPTER

SIX

KEYPOINTS

- Overview
- Key/Value definition
- Attribute definition
- Name convention
- Create an instance
- Auto-completion
- Convert between numpy and torch
- File IO
- Keypoints convention

6.1 Overview

Keypoints is a class for multi-frame, multi-person keypoints data, based on python dict class. It accepts either `numpy.ndarray` or `torch.Tensor`, keeps them in their original type, and offers type conversion methods.

6.2 Key/Value definition

- keypoints: A tensor or ndarray for keypoints, with confidence at the last dim.

  kps2d in shape [n_frame, n_person, n_kps, 3],

  kps3d in shape [n_frame, n_person, n_kps, 4].

- mask: A tensor or ndarray for keypoint mask,

  in shape [n_frame, n_person, n_kps], in dtype uint8.

- convention: Convention name of the keypoints, type str,

  can be found in KEYPOINTS_FACTORY.

We allow you to set other keys and values in a Keypoints instance, but they will be dropped when calling `to_tensor` or `to_numpy`.

6.3 Attribute definition

- logger: Logger for logging. If None, root logger will be selected.
- dtype: The data type of this Keypoints instance, it could be one among `numpy`, `torch` or `auto`. Values will be converted to the certain dtype when setting. If dtype==auto, it will be changed the first time `set_keypoints()` is called, and never changes.

6.4 Name convention

- kps: `kps` is the abbreviation for keypoints. We use `kps` for array-like keypoints data. More precisely, we could use `kps_arr` or `kps_np` for ndarray type keypoints data, and `kps_tensor` for Tensor type data.
- keypoints: `keypoints` denotes an instance of `class Keypoints`, including kps data, mask and convention.

6.5 Create an instance

a. Call `Keypoints()`, keypoints, mask and convention are necessary.

```
from xrmocap.data_structure.keypoints import Keypoints

# If we have kps and mask in numpy.
kps_arr = np.zeros(shape=(2, 3, 25, 3))
mask_arr = np.zeros(shape=(2, 3, 25))
convention = 'openpose_25'
keypoints = Keypoints(kps=kps_arr, mask=mask_arr, convention=convention)
# isinstance(keypoints.get_keypoints(), np.ndarray)

# Or if we have kps and mask in torch.
kps_tensor = torch.zeros(size=(2, 3, 25, 3))
mask_tensor = torch.zeros(size=(2, 3, 25))
convention = 'openpose_25'
keypoints = Keypoints(kps=kps_tensor, mask=mask_tensor, convention=convention)
# isinstance(keypoints.get_keypoints(), torch.Tensor)

# The default dtype is auto. We could set it to 'numpy',
# converting torch values into np.ndarray
kps_tensor = torch.zeros(size=(2, 3, 25, 3))
mask_tensor = torch.zeros(size=(2, 3, 25))
convention = 'openpose_25'
keypoints = Keypoints(dtype='numpy', kps=kps_tensor, mask=mask_tensor,␣
↪convention=convention)
# isinstance(keypoints.get_keypoints(), np.ndarray)
```

b. New an empty instance and set values manually.

```
keypoints = Keypoints()

kps_arr = np.zeros(shape=(2, 3, 25, 3))
mask_arr = np.zeros(shape=(2, 3, 25))
convention = 'openpose_25'
```

(continues on next page)

(continued from previous page)

```
# If you'd like to set them manually, it is recommended
# to obey the following turn: convention -> keypoints -> mask.
keypoints.set_convention(convention)
keypoints.set_keypoints(kps_arr)
keypoints.set_mask(mask_arr)
```

c. New an instance with a dict.

```
kps_arr = np.zeros(shape=(2, 3, 25, 3))
mask_arr = np.zeros(shape=(2, 3, 25))
convention = 'openpose_25'

keypoints_dict = {
  'keypoints': kps_arr,
  'mask': mask_arr,
  'convention': convention
}
keypoints = Keypoints(src_dict=kps_dict)
```

6.6 Auto-completion

We are aware that some users only have data for single frame, single person, and we can deal with that for convenience.

```
keypoints = Keypoints()

kps_arr = np.zeros(shape=(25, 3))
convention = 'openpose_25'

keypoints.set_convention(convention)
keypoints.set_keypoints(kps_arr)
print(keypoints.get_keypoints().shape)
# (1, 1, 25, 3), unsqueeze has been done inside Keypoints

kps_arr = np.zeros(shape=(2, 3, 25, 3))
mask_arr = np.zeros(shape=(25,))  # all the people share the same mask
keypoints.set_keypoints(kps_arr)
keypoints.set_mask(mask_arr)
print(keypoints.get_mask().shape)
# (2, 3, 25), unsqueeze and repeat have been done inside Keypoints
```

6.7 Convert between numpy and torch

a. Convert a Keypoints instance from torch to numpy.

```
# keypoints.dtype == 'torch'
keypoints = keypoints.to_numpy()
# keypoints.dtype == 'numpy' and isinstance(keypoints.get_keypoints(), np.ndarray)
```

b. Convert a Keypoints instance from numpy to torch.

```
# keypoints.dtype == 'numpy'
keypoints_torch = keypoints.to_tensor()
# keypoints_torch.dtype == 'torch' and isinstance(keypoints_torch.get_keypoints(), torch.
↪Tensor)

# we could also assign a device, default is cpu
keypoints_torch = keypoints.to_tensor(device='cuda:0')
```

6.8 File IO

a. Dump an instance to an npz file.

```
dump_path = './output/kps2d.npz'
keypoints.dump(dump_path)
# Even if keypoints.dtype == torch, the dumped arrays in npz are still numpy.ndarray.
```

b. Load an instance from file. The dtype of a loaded instance is always `numpy`.

```
load_path = './output/kps2d.npz'
keypoints = Keypoints.fromfile(load_path)
# We could also new an instance and load.
keypoints = Keypoints()
keypoints.load(load_path)
```

6.9 Keypoints convention

The definition of keypoints varies among dataset. Keypoints convention helps us convert keypoints from one to another.

```
from xrmocap.transform.convention.keypoints_convention import convert_keypoints

# assume we have a keypoint defined in coco_wholebody
# keypoints.get_convention() == 'coco_wholebody'
smplx_keypoints = convert_keypoints(keypoints=keypoints, dst='smplx')
# the output keypoints will have the same dtype as input
```

CHAPTER

SEVEN

LIMBS

- Overview
- Attribute definition
- Create an instance

7.1 Overview

Limbs is a class for person limbs data, recording connection vectors between keypoints. It accepts either `numpy.ndarray` or `torch.Tensor`, convert them into `numpy.ndarray`, `numpy.int32`.

7.2 Attribute definition

- connections: An ndarray for connections, in shape [n_conn, 2], `connections[:, 0]` are start point indice and `connections[:, 1]` are end point indice. `connections[n, :]` is `[start_index, end_index]` of the No.n connection.
- connection_names: A list of strings, could be None. If not None, length of `connection_names` equals to length of `connections`.
- parts: A nested list, could be None. If not None, `len(parts)` is number of parts, and `len(parts[0])` is number of connections in the first part. `parts[i][j]` is an index of connection.
- part_names: A list of strings, could be None. If not None, length of `part_names` equals to length of `parts`.
- points: An ndarray for points, could be None. If not None, it is in shape [n_point, point_dim]. We could use the index record in `connections` to fetch a point.
- logger: Logger for logging. If None, root logger will be selected.

7.3 Create an instance

a. Create instance with raw data and `__init__()`.

```
from xrmocap.data_structure.limbs import Limbs

# only connections arg is necessary for Limbs
connections = np.asarray(
  [[0, 1], [0, 2], [1, 3]]
```

(continues on next page)

(continued from previous page)

```
)
limbs = Limbs(connections=connections)

# split connections into parts
parts = [[0, ], [1, 2]]
part_names = ['head', 'right_arm']
limbs = Limbs(connections=connections, parts=parts, part_names=part_names)
```

b. Get limbs from a well-defined Keypoints instance. The connections will be searched from a sub-set of `human_data` limbs.

```
from xrmocap.transform.limbs import get_limbs_from_keypoints

# Get limbs according to keypoints' mask and convention.
limbs = get_limbs_from_keypoints(keypoints=keypoints2d)
# connections, parts and part_names have been set

# torch type is also accepted
keypoints2d_torch = keypoints2d.to_tensor()
limbs = get_limbs_from_keypoints(keypoints=keypoints2d_torch)

# If both frame_idx and person_idx have been set,
# limbs are searched from a certain frame
# limbs.points have also been set
limbs = get_limbs_from_keypoints(keypoints=keypoints2d, frame_idx=0, person_idx=0)
```

CHAPTER

EIGHT

SMPLDATA

- Overview
- Key/Value definition
- Attribute definition
- Create an instance
- Convert into body_model input
- File IO

8.1 Overview

SMPLData, SMPLXData and SMPLXDData are a classes for SMPL(X/XD) parameters, based on python dict class. It accepts either `numpy.ndarray` or `torch.Tensor`, convert them into `numpy.ndarray`.

8.2 Key/Value definition

- gender: A string marks gender of body_model, female, male or neutral.
- fullpose: An ndarray of full pose, including `global_orient`, `body_pose`, and other pose if exists.

  It's in shape [batch_size, fullpose_dim, 3], while `fullpose_dim` between among SMPLData and SMPLXData.
- transl: An ndarray of translation, in shape [batch_size, 3].
- betas: An ndarray of body shape parameters, in shape [batch_size, betas_dim], while `betas_dim` is defined by input, and it's 10 by default.

8.3 Attribute definition

- logger: Logger for logging. If None, root logger will be selected.

8.4 Create an instance

a. Store the output of SMPLify.

```
smpl_data = SMPLData()
smpl_data.from_param_dict(registrant_output)
```

b. New an instance with ndarray or Tensor.

```
smpl_data = SMPLData(
  gender='neutral',
        fullpose=fullpose,
        transl=transl,
        betas=betas)
```

c. New an instance with a dict.

```
smpl_dict = dict(smpl_data)
another_smpl_data = SMPLData(src_dict=smpl_dict)
```

8.5 Convert into body_model input

```
smpl_data.to_tensor_dict(device='cuda:0')
```

8.6 File IO

a. Dump an instance to an npz file.

```
dump_path = './output/smpl_data.npz'
smpl_data.dump(dump_path)
```

b. Load an instance from file.

```
load_path = './output/smpl_data.npz'
smpl_data = SMPLData.fromfile(load_path)
# We could also new an instance and load.
smpl_data = SMPLData()
smpl_data.load(load_path)
```

CHAPTER

NINE

TRIANGULATION

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

9.1 Overview

Triangulators in XRMoCap are sub-classes of XRPrimer triangulator. For basic usage of triangulators, please refer to xrprimer doc.

9.2 Triangulate points from 2D to 3D

In XRMoCap, we allow triangulators defined in `xrmocap/ops/triangulation` to take input data in arbitrary shape. The first dim shall be view and the last dim shall be `2+n` while n >=0. Here are shapes of some useful examples below:

CHAPTER TEN

SMPLIFY

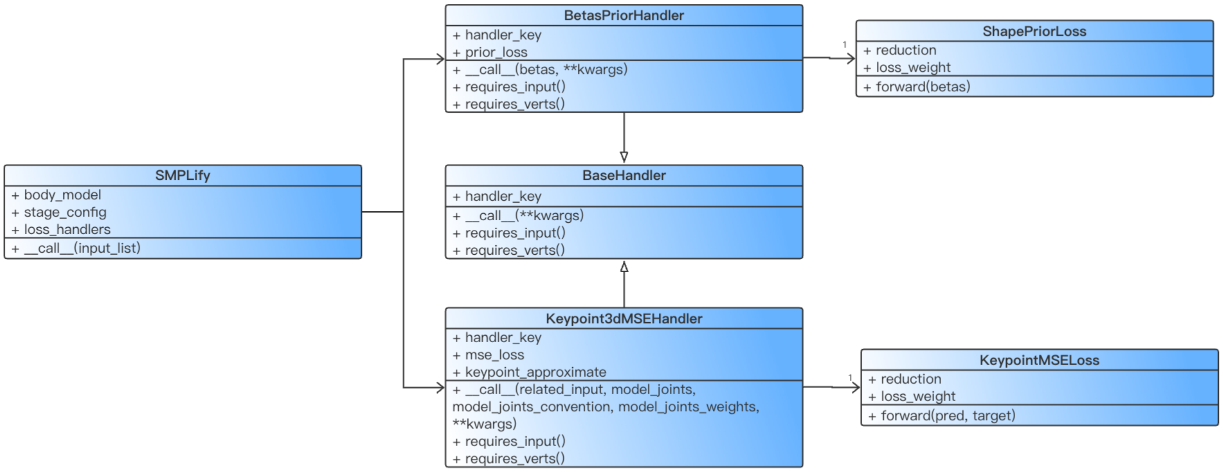
- Overview
- Relationships between classes
- Build a registrant
- Prepare the input and run
- Develop a new loss
- How to write a config file

10.1 Overview

`SMPLify` and `SMPLifyX` are two registrant classes for body model fitting.

10.2 Relationships between classes

- registrant: `SMPLify` and `SMPLifyX`, which holds loss_handlers and get losses of different stages by `input_list` .
- loss_handlers : Sub-classes of `BaseHandler`. It has a `handler_key` as ID for matching and verbose, a loss module for computation. A handler takes body_model parameters, and related input if necessary, prepare them for the loss module, and return loss value to registrant.
- loss module: Sub-classes of `torch.nn.Module`. It has reduction, loss_weight and a forward method.



10.3 Build a registrant

We need a config file to build a registrant, there's an example config at `config/model/registrant/smplify.py`.

```
from xrmocap.model.registrant.builder import build_registrant

smplify_config = dict(
        mmcv.Config.fromfile('configs/modules/model/registrant/smplify.py'))
smplify = build_registrant(smplify_config)
```

To create your own config file and smpl-fitting workflow, see guides.

10.4 Prepare the input and run

We could have keypoints, pointcloud and meshes as input for optimization targets. To organize the input data, we need a sub-class of `BaseInput`. The input class for `Keypoint3dMSEHandler` is `Keypoint3dMSEInput`, and the input class for `Keypoint3dLimbLenHandler` is `Keypoint3dLimbLenInput`. A handler whose handler_key is `keypoints3d_mse` takes an input instance having the same key.

```
from xrmocap.model.registrant.handler.builder import build_handler
from xrmocap.transform.convention.keypoints_convention import convert_keypoints

# keypoints3d is an instance of class Keypoints
keypoints_smpl = convert_keypoints(keypoints=keypoints3d, dst='smpl')
kps3d = torch.from_numpy(keypoints_smpl.get_keypoints()[:, 0, :, :3]).to(
        dtype=torch.float32, device=device)
kps3d_conf = torch.from_numpy(keypoints_smpl.get_mask()[:, 0, :]).to(
    dtype=torch.float32, device=device)

kp3d_mse_input = build_handler(dict(
    type=Keypoint3dMSEInput,
    keypoints3d=kps3d,
    keypoints3d_conf=kps3d_conf,
```

(continues on next page)

(continued from previous page)

```
    keypoints3d_convention='smpl',
    handler_key='keypoints3d_mse'))

kp3d_llen_input = build_handler(dict(
    type=Keypoint3dLimbLenInput,
    keypoints3d=kps3d,
    keypoints3d_conf=kps3d_conf,
    keypoints3d_convention='smpl',
    handler_key='keypoints3d_limb_len'))

smplify_output = smplify(input_list=[kp3d_mse_input, kp3d_llen_input])
```

10.5 Develop a new loss

To develop a new loss and add it to XRMoCap SMPLify, you need 1 or 3 new classes. Here's a tutorial.

10.5.1 a. `SmoothJointLoss`, a loss only requires body_model parameters.

For loss module, we need reduction and loss weight.

```
class SmoothJointLoss(torch.nn.Module):

    def __init__(self,
                 reduction: Literal['mean', 'sum', 'none'] = 'mean',
                 loss_weight: float = 1.0,
                 degree: bool = False,
                 loss_func: Literal['L1', 'L2'] = 'L1'):...
    def forward(
        self,
        body_pose: torch.Tensor,
        loss_weight_override: float = None,
        reduction_override: Literal['mean', 'sum',
                                    'none'] = None) -> torch.Tensor:...
```

For loss handler, we find that existing `BodyPosePriorHandler` meets our requirement. We do not have to develop a new handler class. In config file, add `SmoothJointLoss` like below, it will be deployed when running.

```
handlers = [
    dict(
        handler_key='smooth_joint',
        type='BodyPosePriorHandler',
        prior_loss=dict(
            type='SmoothJointLoss',
            loss_weight=1.0,
            reduction='mean',
            loss_func='L2'),
        logger=logger),
    ...
]
```

10.5.2 b. LimbLengthLoss, a loss requires both body_model parameters and target input.

For loss module, it computes between prediction and target.

```
class LimbLengthLoss(torch.nn.Module):

    def __init__(self,
                 convention: str,
                 reduction: Literal['mean', 'sum', 'none'] = 'mean',
                 loss_weight: float = 1.0,
                 eps: float = 1e-4):...
    def forward(
        self,
        pred: torch.Tensor,
        target: torch.Tensor,
        pred_conf: torch.Tensor = None,
        target_conf: torch.Tensor = None,
        loss_weight_override: float = None,
        reduction_override: Literal['mean', 'sum',
                                    'none'] = None) -> torch.Tensor:
```

For loss handler, we need an input-handler pair. Users pass the input class to registrant, and the handler inside registrant takes the input and compute loss.

```
class Keypoint3dLimbLenInput(BaseInput):

    def __init__(
        self,
        keypoints3d: torch.Tensor,
        keypoints3d_convention: str = 'human_data',
        keypoints3d_conf: torch.Tensor = None,
        handler_key='keypoints3d_limb_len',
    ) -> None:...
    def get_batch_size(self) -> int:...

class Keypoint3dLimbLenHandler(BaseHandler):

    def __init__(self,
                 loss: Union[_LimbLengthLoss, dict],
                 handler_key='keypoints3d_limb_len',
                 device: Union[torch.device, str] = 'cuda',
                 logger: Union[None, str, logging.Logger] = None) -> None:...
    def requires_input(self) -> bool:...
    def requires_verts(self) -> bool:...
    def get_loss_weight(self) -> float:...
    def __call__(self,
                 related_input: Keypoint3dLimbLenInput,
                 model_joints: torch.Tensor,
                 model_joints_convention: str,
                 loss_weight_override: float = None,
                 reduction_override: Literal['mean', 'sum', 'none'] = None,
                 **kwargs: dict) -> torch.Tensor:...
```

10.6 How to write a config file

In the config file, there are some simple values for a registrant.

```
# value of type is the key in registry of build_registrant
# normally it is a class name
type = 'SMPLify'

verbose = True
info_level = 'step'
logger = None
n_epochs = 1
use_one_betas_per_video = True
ignore_keypoints = [
    'neck_openpose', 'right_hip_openpose', 'left_hip_openpose',
    'right_hip_extra', 'left_hip_extra'
]
```

Instance attributes like `body_model` and `optimizer` are given as dictionaies.

```
body_model = dict(
    type='SMPL',
    gender='neutral',
    num_betas=10,
    keypoint_convention='smpl_45',
    model_path='xrmocap_data/body_models/smpl',
    batch_size=1,
    logger=logger)

optimizer = dict(
    type='LBFGS', max_iter=20, lr=1.0, line_search_fn='strong_wolfe')
```

Handlers are given in a list of dict, and the loss module is a sub-dict of the handler dict. It is safe to build some handlers which won't be used. Although it takes time, no error will be caused by the handlers not in use.

```
handlers = [
    dict(
        handler_key='keypoints3d_mse',
        type='Keypoint3dMSEHandler',
        mse_loss=dict(
            type='KeypointMSELoss',
            loss_weight=10.0,
            reduction='sum',
            sigma=100),
        logger=logger),
    dict(
        handler_key='shape_prior',
        type='BetasPriorHandler',
        prior_loss=dict(
            type='ShapePriorLoss', loss_weight=5e-3, reduction='mean'),
        logger=logger),
    dict(
        handler_key='joint_prior',
```

(continues on next page)

(continued from previous page)

```
        type='BodyPosePriorHandler',
        prior_loss=dict(
            type='JointPriorLoss',
            loss_weight=20.0,
            reduction='sum',
            smooth_spine=True,
            smooth_spine_loss_weight=20,
            use_full_body=True),
        logger=logger),
    dict(
        handler_key='pose_prior',
        type='BodyPosePriorHandler',
        prior_loss=dict(
            type='MaxMixturePriorLoss',
            prior_folder='xrmocap_data/body_models',
            num_gaussians=8,
            loss_weight=4.78**2,
            reduction='sum'),
        logger=logger),
    dict(
        handler_key='keypoints3d_limb_len',
        type='Keypoint3dLimbLenHandler',
        loss=dict(
            type='LimbLengthLoss',
            convention='smpl',
            loss_weight=1.0,
            reduction='mean'),
        logger=logger),
]
```

Stages are also given in a list of dict. It controls what loss to be used and what parameter to be updated in each stage. Weight or reduction can be override if `{handler_key}_weight` or `{handler_key}_reduction` is given.

```
stages = [
    # stage 0
    dict(
        n_iter=10,
        ftol=1e-4,
        fit_global_orient=False,
        fit_transl=False,
        fit_body_pose=False,
        fit_betas=True,
        keypoints3d_mse_weight=0.0,  # not in use
        keypoints3d_limb_len_weight=1.0,
        shape_prior_weight=5e-3,
        joint_prior_weight=0.0,
        pose_prior_weight=0.0),
    # stage 1
    dict(
        n_iter=30,
        ftol=1e-4,
        fit_global_orient=True,
```

(continues on next page)

(continued from previous page)

```
        fit_transl=True,
        fit_body_pose=True,
        fit_betas=False,
        keypoints3d_mse_weight=10.0,
        keypoints3d_mse_reduction='sum',
        keypoints3d_limb_len_weight=0.0,
        shape_prior_weight=0.0,
        joint_prior_weight=1e-4,
        pose_prior_weight=1e-4,
        body_weight=1.0,
        use_shoulder_hip_only=False),
]
```

CHAPTER

ELEVEN

MULTI-VIEW SINGLE-PERSON SMPL ESTIMATOR

- Overview
- Arguments
- Run
  - Step0: estimate_keypoints2d
  - Step1: estimate_keypoints3d
  - Step2: estimate_smpl
- Example

11.1 Overview

This tool takes multi-view videos and multi-view calibrated camera parameters as input. By a simple call of `run()`, it outputs detected keypoints2d, triangulated keypoints3d and SMPL parameters for the single person in the multi-view scene.

11.2 Arguments

To construct an estimator instance of this class, you will need a config file like config/estimation/mview_sperson_smpl_estimator.py. `work_dir` makes no sense in this estimator, could be any value, while `bbox_detector`, `kps2d_estimator`, `triangulator` and `smplify` are necessary. `triangulator` shall be a config for triangulator defined in `xrmocap/ops/triangulation`, instead of xrprimer triangulator. `smplify` can be a config for either `class SMPLify` or `class SMPLifyX`. `cam_pre_selector` , `cam_selector` and every list element of `final_selectors` are configs for point selector defined in `xrmocap/ops/triangulation/point_selection`. `kps3d_optimizers` is a list of kps3d_optimizer, defined in `xrmocap/transform/keypoints3d/optim`. When inferring images stored on disk, set `load_batch_size` to a reasonable value will prevent your machine from out of memory.

For more details, see the docstring in code.

11.3 Run

Inside run(), there are three major steps of estimation, and details of each step are shown in the diagram below.

11.3.1 Step0: estimate keypoints2d

In this step, we perform a top-down keypoints2d estimation, detect bbox2d by bbox_detector, and detect keypoints2d in every bbox by kps2d_estimator. You can choose the model and weight you like by modifying the config file.

11.3.2 Step1: estimate keypoints3d

In this step, we split the estimation into four sub-steps: camera selection, point selection, triangulation and optimization. Every sub-step can be skipped by passing None in config except triangulation. First, we use cam_pre_selector to select good 2D points from all detected keypoints2d, and select well-calibrated cameras by cam_selector. Second, we use cascaded point selectors in final_selectors to select 2D points from well-calibrated views, for triangulation. After multi-view triangulation, in the forth sub-step, we use cascaded keypoints3d optimizers in kps3d_optimizers to optimize keypoints3d, and the result of optimization will be the return value of step estimate_keypoints3d.

11.3.3 Step2: estimate SMPL

In this step, we estimate SMPL or SMPLX parameters from keypoints3d of last step. For details of smpl fitting, see smplify doc.

11.4 Example

```
import numpy as np

from xrmocap.estimation.builder import build_estimator
from xrprimer.data_structure.camera import FisheyeCameraParameter


# multi-view camera parameter list
cam_param_list = [FisheyeCameraParameter.fromfile(
    f'fisheye_param_{idx:02d}.json') for idx in range(10)]
# multi-view image array
mview_img_arr = np.zeros(shape=(10, 150, 1080, 1920, 3), dtype=np.uint8)
# build an estimator
mview_sperson_smpl_estimator = build_estimator(estimator_config)

# run the estimator on image array
keypoints2d_list, keypoints3d, smpl_data = mview_sperson_smpl_estimator.run(
    cam_param=cam_param_list, img_arr=mview_img_arr)
```

CHAPTER

TWELVE

MULTI-VIEW MULTI-PERSON TOP-DOWN SMPL ESTIMATOR

12.1 Overview

This tool takes multi-view RGB sequences and multi-view calibrated camera parameters as input. By a simple call of `run()`, it outputs triangulated keypoints3d and SMPL parameters for the multi-person in the multi-view scene.

12.2 Arguments

- **output_dir**: `output_dir` is the path to the directory saving all possible output files, including keypoints3d, SMPLData and visualization videos.
- **estimator_config**: `estimator_config` is the path to a `MultiViewMultiPersonTopDownEstimator` config file, where `bbox_detector`, `kps2d_estimator`, `associator`, `triangulator` and `smplify` are necessary. Every element of `point_selectors` are configs for point selector defined in `xrmocap/ops/triangulation/point_selection`. `kps3d_optimizers` is a list of kps3d_optimizer, defined in `xrmocap/transform/keypoints3d/optim`. When inferring images stored on disk, set `load_batch_size` to a reasonable value will prevent your machine from out of memory. For more details, see config and the docstring in code.
- **image_and_camera_param**: `image_and_camera_param` is a text file contains the image path and the corresponding camera parameters. Line 0 is the image path of the first view, and line 1 is the corresponding camera parameter path. Line 2 is the image path of the second view, and line 3 is the corresponding camera parameter path, and so on.

```
xrmocap_data/Shelf_50/Shelf/Camera0/
xrmocap_data/Shelf_50/xrmocap_meta_testset_small/scene_0/camera_parameters/fisheye_param_
↪00.json
```

(continues on next page)

(continued from previous page)

```
xrmocap_data/Shelf_50/Shelf/Camera1/
xrmocap_data/Shelf_50/xrmocap_meta_testset_small/scene_0/camera_parameters/fisheye_param_
↪01.json
```

- **start_frame**: `start_frame` is the index of the start frame.
- **end_frame**: `end_frame` is the index of the end frame.
- **enable_log_file** By default, enable_log_file is False and the tool will only print log to console. Add `--enable_log_file` makes it True and a log file named `{smc_file_name}_{time_str}.txt` will be written.
- **disable_visualization** By default, disable_visualization is False and the tool will visualize keypoints3d and SMPLData with an orbit camera, overlay SMPL meshes on one view.

12.3 Run

Inside `run()`, there are three major steps of estimation, and details of each step are shown in the diagram below.

12.3.1 Step0: estimate perception2d

In this step, we perform a top-down keypoints2d estimation, detect bbox2d by `bbox_detector`, and detect keypoints2d in every bbox by `kps2d_estimator`. You can choose the model and weight you like by modifying the config file.

12.3.2 Step1: establish cross-frame and cross-person associations

In this step, we match the keypoints2d across views by `associator` and add temporal tracking and filtering. For recommended configs on `associator`, you can check out the README.md

12.3.3 Step2: estimate keypoints3d

In this step, we split the estimation into three sub-steps: point selection, triangulation and optimization. Every sub-step can be skipped by passing `None` in config except triangulation. We use cascaded point selectors in `point_selectors` to select 2D points from well-calibrated views, for triangulation. After multi-view triangulation, in the third sub-step, we use cascaded keypoints3d optimizers in `kps3d_optimizers` to optimize keypoints3d.

12.3.4 Step3: estimate smpl

In this step, we estimate SMPL parameters from keypoints3d. For details of smpl fitting, see smplify doc.

12.4 Example

```
python tools/mview_mperson_topdown_estimator.py \
      --image_and_camera_param 'data/image_and_camera_param.txt' \
      --start_frame 0 \
      --end_frame 10 \
      --enable_log_file
```

CHAPTER

THIRTEEN

LEARNING-BASED MODEL EVALUATION

- Overview
- Preparation
- Example

13.1 Overview

This tool takes a config file and MvP model checkpoints and performs evaluation on Shelf, Campus or CMU Panoptic dataset.

13.2 Preparation

1. Install `Deformable` package (Skip if you have done this step during model training)

Download the `./ops` folder, rename and place the folder as `ROOT/xrmocap/model/deformable`. Install `Deformable` by running:

```
cd ROOT/xrmocap/model/deformable/
sh make.sh
```

2. Prepare Datasets

Follow the *dataset tool* tutorial to prepare the train and test data. Some pre-processed datasets are available for download *here*. Place the `trainset_pesudo_gt` and `testset` data including meta data under `ROOT/xrmocap_data`.

3. Prepare pre-trained model weights and model checkpoints

Download pre-trained backbone weights or MvP model checkpoints from here. Place the model weights under `ROOT/weight`.

4. Prepare config files

Modify the config files in `ROOT/configs/mvp` if needed. Make sure the directories in config files match the directories and file names for your datasets and pre-traind model weights.

The final file structure ready for evaluation would be like:

```
xrmocap
├── xrmocap
├── tools
├── configs
```

(continues on next page)

(continued from previous page)

```
└── weight
    ├── xrmocap_mvp_campus.pth.tar
    ├── xrmocap_mvp_shelf.pth.tar
    ├── xrmocap_mvp_panoptic_5view.pth.tar
    ├── xrmocap_mvp_panoptic_3view_3_12_23.pth.tar
    └── xrmocap_pose_resnet50_panoptic.pth.tar
└── xrmocap_data
    └── meta
        └── shelf
            └── xrmocap_meta_testset
        ├── campus
        └── panoptic
    ├── Shelf
    ├── CampusSeq1
    └── panoptic
        ├── 160906_band4
        ├── 160906_ian5
        ├── ...
        └── 160906_pizza1
```

13.2.1 Example

Start evaluation with 8 GPUs with provided config file and pre-trained weights for Shelf dataset:

```
python -m torch.distributed.launch \
    --nproc_per_node=8 \
    --use_env tools/eval_model.py \
    --cfg configs/mvp/shelf_config/mvp_shelf.py \
    --model_path weight/xrmocap_mvp_shelf.pth.tar
```

Alternatively, you can also run the script directly:

```
sh ROOT/scripts/val_mvp.sh ${NUM_GPUS} ${CFG_FILE} ${MODEL_PATH}
```

Example:

```
sh ROOT/scripts/val_mvp.sh 8 configs/mvp/shelf_config/mvp_shelf.py weight/xrmocap_mvp_
↪shelf.pth.tar
```

If you can run XRMoCap on a cluster managed with slurm, you can use the script:

```
sh ROOT/scripts/slurm_eval_mvp.sh ${PARTITION} ${NUM_GPUS} ${CFG_FILE} ${MODEL_PATH}
```

Example:

```
sh ROOT/scripts/slurm_eval_mvp.sh MyPartition 8 configs/mvp/shelf_config/mvp_shelf.py␣
↪weight/xrmocap_mvp_shelf.pth.tar
```

CHAPTER

FOURTEEN

MULTI-VIEW MULTI-PERSON EVALUATION

- Overview
- Argument
- Example

14.1 Overview

This tool takes calibrated camera parameters, RGB sequences, 2d perception data and 3d ground-truth from `MviewMpersonDataset` as input, generate multi-view multi-person keypoints3d and evaluate on the Campus/Shelf/CMU-Panoptic datasets.

14.2 Argument

- **enable_log_file** By default, enable_log_file is False and the tool will only print log to console. Add `--enable_log_file` makes it True and a log file named `{smc_file_name}_{time_str}.txt` will be written.
- **evaluation_config**: `evaluation_config` is the path to a `TopDownAssociationEvaluation` config file. For more details, see docs for `TopDownAssociationEvaluation` and the docstring in code.

Also, you can find our prepared config files at `configs/mvpose/*/eval_keypoints3d.py` or `configs/mvpose_tracking/*/eval_keypoints3d.py`.

14.3 Example

Evaluate on the Shelf dataset and run the tool without tracking.

```
python tools/mview_mperson_evaluation.py \
      --enable_log_file \
      --evaluation_config configs/mvpose/shelf_config/eval_keypoints3d.py
```

Evaluate on the Shelf dataset and run the tool with tracking.

```
python tools/mview_mperson_evaluation.py \
      --enable_log_file \
      --evaluation_config configs/mvpose_tracking/shelf_config/eval_keypoints3d.py
```

CHAPTER

FIFTEEN

MULTI-VIEW MULTI-PERSON SMPLIFY3D

15.1 Overview

This tool could generate multi-view multi-person SMPLData from keypoints3d.

15.2 Argument

- **estimator_config**: `estimator_config` is the path to a `MultiPersonSMPLEstimator` config file. For more details, see docs for `MultiPersonSMPLEstimator` and the docstring in code.
- **start_frame**: `start_frame` is the index of the start frame.
- **end_frame**: `end_frame` is the index of the end frame.
- **bbox_thr**: `bbox_thr` is the threshold of the 2d bbox, which should be the same as the threshold used to generate the keypoints3d.
- **keypoints3d_path**: `keypoints3d_path` is the path to the keypoints3d file.
- **matched_kps2d_idx**: `matched_kps2d_idx` is the matched keypoints2d index from different views, where is generated by code.
- **image_and_camera_param**: `image_and_camera_param` is a text file contains the image path and the corresponding camera parameters. Line 0 is the image path of the first view, and line 1 is the corresponding camera parameter path. Line 2 is the image path of the second view, and line 3 is the corresponding camera parameter path, and so on.
- **perception2d_path**: `perception2d_path` is the path to the 2d perception data.
- **output_dir**: `output_dir` is the path to the directory saving all possible output files, including SMPLData and visualization videos.
- **visualize**: By default, visualize is False. Add `--visualize` makes it True and the tool will visualize SMPLData with an orbit camera, overlay SMPL meshes on one view.
- **enable_log_file** By default, enable_log_file is False and the tool will only print log to console. Add `--enable_log_file` makes it True and a log file named `{smc_file_name}_{time_str}.txt` will be written.

15.3 Example

Run the tool with visualization.

```
python tools/mview_mperson_smplify3d.py \
      --estimator_config 'configs/modules/core/estimation/mperson_smpl_estimator.py' \
      --start_frame 300 \
      --end_frame 600 \
      --keypoints3d_path 'output/mvpose_tracking/shelf/scene0_pred_keypoints3d.npz' \
      --matched_kps2d_idx 'output/mvpose_tracking/shelf/scene0_matched_kps2d_idx.npy' \
      --image_and_camera_param 'xrmocap_data/Shelf/image_and_camera_param.txt' \
      --perception2d_path 'xrmocap_data/Shelf/xrmocap_meta_test/scene_0/perception_2d.npz
↪' \
      --output_dir 'output/mvpose_tracking/shelf/smpl' \
      --visualize \
      --enable_log_file
```

CHAPTER

SIXTEEN

TOOL PREPARE_DATASET

- Overview
- Argument: converter_config
- Argument: overwrite
- Argument: disable_log_file
- Argument: paths
- Example

16.1 Overview

This tool converts original dataset to our unified meta-data, with data converters controlled by configs.

16.2 Argument: converter_config

`converter_config` is the path to a data_converter config file like below. If 2D perception data is not required by your method, set `bbox_detector` and `kps2d_estimator` to None. It will skip 2D perception and saves your time. For more details, see the docstring in code.

```
type = 'ShelfDataCovnerter'
data_root = 'datasets/Shelf'
bbox_detector = dict(
    type='MMtrackDetector',
    mmtrack_kwargs=dict(
        config='config/human_detection/' +
        'mmtrack_deepsort_faster-rcnn_fpn_4e_mot17-private-half.py',
        device='cuda'))
kps2d_estimator = dict(
    type='MMposeTopDownEstimator',
    mmpose_kwargs=dict(
        checkpoint='weight/hrnet_w48_coco_wholebody' +
        '_384x288_dark-f5726563_20200918.pth',
        config='config/human_detection/mmpose_hrnet_w48_' +
        'coco_wholebody_384x288_dark_plus.py',
        device='cuda'))
scene_range = [[300, 600]]
```

(continues on next page)

(continued from previous page)

```
meta_path = 'datasets/Shelf/xrmocap_meta_testset'
visualize = True
```

Also, you can find our prepared config files in `config/data/data_converter`, with or without perception.

16.3 Argument: overwrite

By default, overwrite is False and there is a folder found at `meta_path`, the tool will raise an error, to avoid removal of existed files. Add `--overwrite` makes it True and allows the tool to overwrite any file below `meta_path`.

16.4 Argument: disable_log_file

By default, disable_log_file is False and a log file named `converter_log_{time_str}.txt` will be written. Add `--disable_log_file` makes it True and the tool will only print log to console.

After the tool succeeds, you will find log file in `meta_path`, otherwise it will be in `logs/`.

16.5 Argument: paths

By default, `data_root` and `meta_path` are empty, the tool takes paths in converter config file. If both of them are set, the tool takes paths from argv.

16.6 Examples

Run the tool when paths configured in `campus_data_converter_testset.py`.

```
python tool/prepare_dataset.py \
        --converter_config config/data/data_converter/campus_data_converter_testset.py
```

Run the tool with explicit paths.

```
python tool/prepare_dataset.py \
  --converter_config config/data/data_converter/campus_data_converter_testset.py \
  --data_root datasets/Campus \
  --meta_path datasets/Campus/xrmocap_meta_testset
```

CHAPTER

SEVENTEEN

TOOL PROCESS_SMC

17.1 Overview

This tool takes calibrated camera parameters and RGB frames from a SenseMoCap file as input, generate multi-view keypoints2d, keypoints3d and SMPLData.

17.2 Argument: estimator_config

`estimator_config` is the path to a `MultiViewSinglePersonSMPLEstimator` config file. For more details, see docs for `MultiViewSinglePersonSMPLEstimator` and the docstring in code.

Also, you can find our prepared config files at `config/estimation/mview_sperson_smpl_estimator.py`.

17.3 Argument: output_dir

`output_dir` is the path to the directory saving all possible output files, including multi-view keypoints2d, keypoints3d and SMPLData, log and visualization videos.

17.4 Argument: disable_log_file

By default, disable_log_file is False and a log file named `{smc_file_name}_{time_str}.txt` will be written. Add `--disable_log_file` makes it True and the tool will only print log to console.

17.5 Argument: visualize

By default, visualize is False. Add `--visualize` makes it True and the tool will visualize keypoints3d with an orbit camera, overlay projected keypoints3d on some views, and overlay SMPL meshes on one view.

17.6 Example

Run the tool with visualization.

```
python tools/process_smc.py \
        --estimator_config configs/humman_mocap/mview_sperson_smpl_estimator.py \
        --smc_path xrmocap_data/humman/raw_smc/p000105_a000195.smc \
        --output_dir xrmocap_data/humman/p000105_a000195_output \
        --visualize
```

CHAPTER

EIGHTEEN

LEARNING-BASED MODEL TRAINING

- Overview
- Preparation
- Example

18.1 Overview

This tool takes a config file and starts trainig MvP model with Shelf, Campus or CMU Panoptic dataset.

18.2 Preparation

1. Install `Deformable` package (Skip if you have done this step during model evaluation)

Download the `./ops` folder, rename and place the folder as `ROOT/xrmocap/model/deformable`. Install `Deformable` by running:

```
cd ROOT/xrmocap/model/deformable/
sh make.sh
```

2. Prepare Datasets

Follow the *dataset tool* tutorial to prepare the train and test data. Some pre-processed datasets are available for download *here*. Place the `trainset_pesudo_gt` and `testset` data including meta data under `ROOT/xrmocap_data`.

3. Prepare pre-trained model weights and model checkpoints

Download pre-trained backbone weights or MvP model checkpoints from here. Place the model weights under `ROOT/weight`.

4. Prepare config files

Modify the config files in `ROOT/configs/mvp` if needed. Make sure the directories in config files match the directories and file names for your datasets and pre-traind weights.

The final file structure ready for training would be like:

```
xrmocap
├── xrmocap
├── tools
├── configs
└── weight
```

(continues on next page)

(continued from previous page)

```
    ├── xrmocap_mvp_campus.pth.tar
    ├── xrmocap_mvp_shelf.pth.tar
    ├── xrmocap_mvp_panoptic_5view.pth.tar
    ├── xrmocap_mvp_panoptic_3view_3_12_23.pth.tar
    └── xrmocap_pose_resnet50_panoptic.pth.tar
└── xrmocap_data
    └── meta
        └── shelf
            ├── xrmocap_meta_testset
            └── xrmocap_meta_trainset_pesudo_gt
        ├── campus
        └── panoptic
    ├── Shelf
    ├── CampusSeq1
    └── panoptic
        ├── 160906_band4
        ├── 160906_ian5
        ├── ...
        └── 160906_pizza1
```

18.3 Example

Start training with 8 GPUs with provided config file for Campus dataset:

```
python -m torch.distributed.launch \
        --nproc_per_node= 8 \
        --use_env tools/train_model.py \
        --cfg configs/mvp/campus_config/mvp_campus.py \
```

Alternatively, you can also run the script directly:

```
sh ROOT/scripts/train_mvp.sh ${NUM_GPUS} ${CFG_FILE}
```

Example:

```
sh ROOT/scripts/train_mvp.sh 8 configs/mvp/campus_config/mvp_campus.py
```

If you can run XRMoCap on a cluster managed with slurm, you can use the script:

```
sh ROOT/scripts/slurm_train_mvp.sh ${PARTITION} ${NUM_GPUS} ${CFG_FILE}
```

Example:

```
sh ROOT/scripts/slurm_train_mvp.sh MyPartition 8 configs/mvp/shelf_config/mvp_shelf.py
```

CHAPTER

NINETEEN

TOOL VISUALIZE_DATASET

19.1 Overview

This tool loads our converted meta-data, visualize meta-data with background frames from original dataset, scene by scene.

19.2 Argument: vis_config

`vis_config` is the path to a data_visualization config file like below. Visualization for perception 2D and groudtruth 3D is optional, controlled by `vis_percep2d` and `vis_gt_kps3d`. Visualization for predicted 3D will be done if `pred_kps3d_paths` is not empty, and each element is path to a keypoints3d npz file. For more details, see the docstring in code.

```
type = 'MviewMpersonDataVisualization'
data_root = 'Shelf'
meta_path = 'datasets/Shelf/xrmocap_meta_testset'
output_dir = 'datasets/Shelf/xrmocap_meta_testset_visualization'
pred_kps3d_paths = ['datasets/Shelf/xrmocap_meta_testset/predicted_keypoints3d.npz']
bbox_thr = 0.96
vis_percep2d = True
vis_gt_kps3d = True
```

Also, you can find our prepared config files in `config/data/data_visualization`.

19.3 Argument: overwrite

By default, overwrite is False and there is a folder found at `output_dir`, the tool will raise an error, to avoid removal of existed files. Add `--overwrite` makes it True and allows the tool to overwrite any file below `output_dir`.

19.4 Argument: disable_log_file

By default, disable_log_file is False and a log file named `visualization_log_{time_str}.txt` will be written. Add `--disable_log_file` makes it True and the tool will only print log to console.

After the tool succeeds, you will find log file in `output_dir`, otherwise it will be in `logs/`.

19.5 Argument: paths

By default, `data_root`, `meta_path` and `output_dir` are empty, the tool takes paths in data_visualization config file. If all of them are set, the tool takes paths from argv.

19.6 Examples

Run the tool when paths configured in `shelf_data_visualization_testset.py`.

```
python tool/visualize_dataset.py \
        --converter_config config/data/data_visualization/shelf_data_visualization_
↪testset.py
```

Run the tool with explicit paths.

```
python tool/prepare_dataset.py \
  --converter_config config/data/data_converter/shelf_data_visualization_testset.py \
  --data_root datasets/Shelf \
  --meta_path datasets/Shelf/xrmocap_meta_testset \
  --output_dir datasets/Shelf/xrmocap_meta_testset/visualization
```

CHAPTER

TWENTY

INTRODUCTION

This file introduces the framework design and file structure of xrmocap.

20.1 Framework

20.1.1 Optimization-based framework

[framework for single person]

The construction pipeline starts with frame-by-frame 2D keypoint detection and manual camera estimation. Then triangulation and bundle adjustment are applied to optimize the camera parameters as well as the 3D keypoints. Finally we sequentially fit the SMPL model to 3D keypoints to get a motion sequence represented using joint angles and a root trajectory. The following figure shows our pipeline overview.

[framework for multiple person]

For multiple person, two challenges will be posed. One is to find correspondence between different views The other is solve person-person occlusion

From the figure above, it illustrates that two modules are adding, namely matching module and tracking module.

20.1.2 Learning-based framework

describe the component of each module (as in the paper)

how to incorporate optimization and learning-based methods into one framework

20.2 File structures

```
.
├── Dockerfile                    # Dockerfile for quick start
├── README.md                     # README
├── README_CN.md                  # README in Chinese
├── configs                       # Recommended configuration files for tools and modules
├── docs                          # docs
├── requirements                  # pypi requirements
├── scripts                       # scripts for downloading data, training and evaluation
├── tests                         # unit tests
├── tools                         # utility tools
```

(continues on next page)

(continued from previous page)

```
└── xrmocap
    ├── core
    │   ├── estimation          # multi-view single-person or multi-pserson SMPL␣
↪estimator
    │   ├── evaluation          # evluation on datasets
    │   ├── hook                # hooks to registry
    │   ├── train               # end-to-end model trainer
    │   └── visualization       # visualization functions for data structures
    ├── data
    │   ├── data_converter      # modules for dataset converting into XRMoCap annotation
    │   ├── data_visualization  # modules for dataset visualization
    │   ├── dataloader          # implementation of torch.utils.data.Dataloader
    │   └── dataset             # implementation of torch.utils.data.Dataset
    ├── data_structure          # data structure for single-person SMPL(X/XD), multi-
↪person keypoints etc.
    ├── human_perception        # modules for human perception
    ├── io                      # functions for Input/Output
    ├── model                   # neural network modules
    │   ├── architecture        # high-level models for a specific task
    │   ├── body_model          # re-implementation of SMPL(X) body models
    │   ├── loss                # loss functions
    │   ├── mvp                 # models for MVP
    │   └── registrant          # re-implementation of SMPLify(X)
    ├── ops                     # operators for multi-view MoCap
    │   ├── projection          # modules for projecting 3D points to 2D points
    │   ├── top_down_association # multi-view human association and tracking on top-down-
↪detection data
    │   └── triangulation       # modules for triangulating 2D points to 3D points
    │       └── point_selection # modules for selecting good 2D points before␣
↪triangulation
    ├── transform               # functions and classes for data transform, e.g., bbox,␣
↪image, keypoints3d
    ├── utils                   # utility functions for camera, geomotry computation␣
↪and others
    └── version.py              # digital version of XRMocap
```

Usage of each module/folder

CHAPTER

TWENTYONE

LEARN ABOUT CONFIGS

We incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments.

21.1 Modify config through script arguments

Take MVPose and MVPose tracking as an example

If you want to use tracker, you need to create a variable of dictionary type containing `type='KalmanTracking'` and others needed in `__init__()`. Then you need to build it and will get a Kalman tracking module, otherwise you just need to set `kalman_tracking_config=None`.

Example:

```
kalman_tracking_config=dict(type='KalmanTracking', n_cam_min=2, logger=logger)

if isinstance(kalman_tracking_config, dict):
      kalman_tracking = build_kalman_tracking(kalman_tracking_config)
else:
      kalman_tracking = kalman_tracking_config
```

Using trackers

tracker is only needed for multiple person, for single person, it can also be used but may slow down the speed.

CHAPTER

TWENTYTWO

ADD NEW DATASETS

22.1 Overview

This doc is a tutorial for how to support a new public dataset, or data collected by user.

22.2 Online conversion

For online conversion, program does not write any file to disk. You have to define a new sub-class of `torch.utils.data.Dataset`, loading data from origin files, and return the same values in same sequence in `__getitem__()` as our datasets.

22.3 Offline conversion (recommend)

For offline conversion, we convert the origin dataset into annotations in a unified format, save the annotations to disk, before training or evaluation starts. Such a conversion module is called `data_converter` in XRMoCap, and you can find examples in xrmocap/data/data_converter.

22.3.1 File tree of our unified format

```
Dataset_xxx
├── (files in Dataset_xxx)
└── xrmocap_meta_xxxx
    ├── dataset_name.txt
    ├── scene_0
    │   ├── camera_parameters
    │   │   ├── fisheye_param_00.json
    │   │   ├── fisheye_param_01.json
    │   │   ├── ...
    │   │   └── fisheye_param_{n_view-1}.json
    │   ├── image_list_view_00.txt
    │   ├── image_list_view_01.txt
    │   ├── ...
    │   ├── image_list_view_{n_view-1}.txt
    │   ├── keypoints3d_GT.npz
    │   └── perception_2d.npz
    ├── scene_1
```

(continues on next page)

(continued from previous page)

```
    │     └── ...
    └── scene_{n_scene-1}
          └── ...
```

22.3.2 Camera parameters of our unified format

Each scene has its independent multi-view camera parameters, and each json file is dumped by `class FisheyeCameraParameter` in XRPrimer.

22.3.3 Image list of our unified format

In a scene whose frame length is `n_frame`, number of cameras is `n_view`, there are `n_view` image list files, and every file has `n_frame` lines inside, take the `frame_idx`-th line in file `image_list_view_{view_idx}.txt`, we get a path of image relative to dataset_root(Dataset_xxx).

22.3.4 Keypoints3d groundtruth of our unified format

`keypoints3d_GT.npz` is a file dumped by `class Keypoints`, and it can be load by `keypoints3d = Keypoints.fromfile()`. In a scene whose frame length is `n_frame`, max number of objects is `n_person`, number of single person's keypoints is `n_kps`, `keypoints3d.get_keypoints()` returns an ndarray in shape [n_frame, n_person, n_kps, 4], and `keypoints3d.get_mask()` is an ndarray in shape [n_frame, n_person, n_kps] which indicates which person and which keypoint is valid at a certain frame.

22.3.5 Perception 2D of our unified format

`perception_2d.npz` is an compressed npz file of a python dict, whose structure lies below:

```
perception_2d_dict = dict(
        bbox_tracked=True,
  # True if bbox indexes have nothing to do with identity
  bbox_convention='xyxy',
  # xyxy or xywh
  kps2d_convention='coco',
  # or any other convention defined in KEYPOINTS_FACTORY
  bbox2d_view_00=bbox_arr_0,
  ...
  # an ndarray of bboxes detected in view 0, in shape (n_frame, n_person_max, 5)
  # bbox_arr[..., 4] are bbox scores
  # if bbox_arr[f_idx, p_idx, 4] == 0, bbox at bbox_arr[f_idx, p_idx, :4] is invalid
  kps2d_view_00=kps2d_arr_0,
  ...
  # an ndarray of keypoints2d detected in view 0, in shape (n_frame, n_person_max, n_kps,
↪ 3)
  # kps2d_arr[..., 2] are keypoints scores
  kps2d_mask_view_00=kps2d_mask_0,
  ...
  # a mask ndarray of keypoints2d in view 0, in shape (n_frame, n_person_max, n_kps)
```

(continues on next page)

(continued from previous page)

```
  # if kps2d_mask[f_idx, p_idx, kps_idx] == 0, kps at kps2d_arr[f_idx, p_idx, kps_idx,␣
↪:] is invalid
)
```

22.4 Class data_converter

For frequent conversion, it's better to define a data_converter class inherited from `BaseDataCovnerter`. After that, you can use our prepare_dataset tool to convert the new dataset. See the tool tutorial for details.

CHAPTER

TWENTYTHREE

ADD NEW MODULE

If you want to add a new module, write a class and register it in builder. Here we take triangulator as example.

23.1 Develop PytorchTriangulator class

1. Inherit from base class

Inherit from `BaseTriangulator` and assign correct values for class attributes.

```
class PytorchTriangulator(BaseTriangulator):
    CAMERA_CONVENTION = 'opencv'
    CAMERA_WORLD2CAM = True
```

Complete `__init__` and do not forget to add arguments of super-class.

```
    def __init__(self,
                 camera_parameters: List[FisheyeCameraParameter],
                 logger: Union[None, str, logging.Logger] = None) -> None:
        self.logger = get_logger(logger)
        super().__init__(camera_parameters=camera_parameters, logger=logger)
```

2. Complete necessary methods defined by base class

```
    def triangulate(
            self,
            points: Union[torch.Tensor, list, tuple],
            points_mask: Union[torch.Tensor, list, tuple] = None) -> np.ndarray:

    def get_reprojection_error(
        self,
        points2d: torch.Tensor,
        points3d: torch.Tensor,
        points_mask: torch.Tensor = None,
        reduction: Literal['mean', 'sum', 'none'] = 'none'
    ) -> Union[torch.Tensor, float]:

    def get_projector(self) -> PytorchProjector:
```

3. Add special methods of this class(Optional)

```
    def get_device(
            self) -> torch.device:
```

4. Register the class in builder

Insert the following lines into `xrmocap/ops/triangulation/builder.py`.

```
from .pytorch_triangulator import PytorchTriangulator

TRIANGULATORS.register_module(
    name='PytorchTriangulator', module=PytorchTriangulator)
```

23.2 Build and use

Test whether the new module is OK to build.

```
from xrmocap.ops.triangulation.builder import build_triangulator

triangulator = build_triangulator(dict(type='PytorchTriangulator'))
```

CHAPTER

TWENTYFOUR

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!

24.1 Installation

- 'ImportError: libpng16.so.16: cannot open shared object file: No such file or directory'

  Please refer to xrprimer faq.

- 'ImportError: liblapack.so.3: cannot open shared object file: No such file or directory'

  Please refer to xrprimer faq.

- 'ModuleNotFoundError: No module named mmhuman3d.core.conventions.joints_mapping'

  Package `joints_mapping` actually exists in github, but it is not installed by pip for absence of `joints_mapping/__init__.py`. Install mmhuman3d from source will solve it:

  ```
  cd PATH_FOR_MMHUMAN3D
  git clone https://github.com/open-mmlab/mmhuman3d.git
  pip install -e ./mmhuman3d
  ```

- 'BrokenPipeError: ../../lib/python3.8/site-packages/xrprimer/utils/ffmpeg_utils.py:189: BrokenPipeError'

  You've installed a wrong version of ffmpeg. Try to install it by the following command, and do not specify any channel:

  ```
  conda install ffmpeg
  ```

CHAPTER

TWENTYFIVE

CHANGELOG

25.1 v0.5.0 (01/09/2022/)

Highlights

- Support HuMMan Mocap toolchain for multi-view single person SMPL estimation
- Reproduce MvP, a deep-learning-based SOTA for multi-view multi-human 3D pose estimation
- Reproduce MVPose (single frame) and MVPose (temporal tracking and filtering), two optimization-based methods for multi-view multi-human 3D pose estimation
- Support SMPLify, SMPLifyX, SMPLifyD and SMPLifyXD

New Features

- Add peception module based on mmdet, mmpose and mmtrack
- Add Shape-aware 3D Pose Optimization
- Add Keypoints3d optimizer and multi-view single-person api
- Add data_converter and data_visualization for shelf, campus and cmu panoptic datasets
- Add multiple selectors to support more point selection strategies for triangulation
- Add Keypoints and Limbs data structure
- Add multi-way matching registry
- Refactor the pictorial block (c/c++) in python

CHAPTER TWENTYSIX

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 XRMoCap 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 XRMoCap 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
   limitations under the License.
```

The XRMoCap license applies to all parts of XRMoCap that are not externally maintained libraries. The dependencies of XRMoCap are licensed under following licenses.

HuMMan MoCap is licensed for use as follows:

```
S-Lab License 1.0

Copyright 2022 S-Lab

Redistribution and use for non-commercial purpose in source and binary forms, with or␣
↪without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of␣
↪conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list␣
↪of conditions and the following disclaimer in the documentation and/or other materials␣
↪provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be␣
↪used to endorse or promote products derived from this software without specific prior␣
↪written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY␣
↪EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES␣
↪OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT␣
↪SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,␣
↪INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED␣
↪TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,␣
↪WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)␣
↪ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY␣
↪OF SUCH DAMAGE.
4. In the event that redistribution and/or use for commercial purpose in source or␣
↪binary forms, with or without modification is required, please contact the␣
↪contributor(s) of the work.
```

SMPLify-X is licensed for use as follows:

```
License

Software Copyright License for non-commercial scientific research purposes
```

(continues on next page)

(continued from previous page)

```
Please read carefully the following terms and conditions and any accompanying
documentation before you download and/or use the SMPL-X/SMPLify-X model, data and
software, (the "Model & Software"), including 3D meshes, blend weights, blend shapes,
textures, software, scripts, and animations. By downloading and/or using the Model &
Software (including downloading, cloning, installing, and any other use of this github
repository), you acknowledge that you have read these terms and conditions, understand
them, and agree to be bound by them. If you do not agree with these terms and
conditions, you must not download and/or use the Model & Software. Any infringement of
the terms of this agreement will automatically terminate your rights under this License

Ownership / Licensees
The Software and the associated materials has been developed at the

Max Planck Institute for Intelligent Systems (hereinafter "MPI").

Any copyright or patent right is owned by and proprietary material of the

Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V. (hereinafter “MPG”; MPI
and MPG hereinafter collectively “Max-Planck”)

hereinafter the “Licensor”.

License Grant
Licensor grants you (Licensee) personally a single-user, non-exclusive, non-transferable,
 free of charge right:

To install the Model & Software on computers owned, leased or otherwise controlled by
you and/or your organization;
To use the Model & Software for the sole purpose of performing non-commercial scientific
research, non-commercial education, or non-commercial artistic projects;
Any other use, in particular any use for commercial, pornographic, military, or
surveillance, purposes is prohibited. This includes, without limitation, incorporation
in a commercial product, use in a commercial service, or production of other artifacts
for commercial purposes. The Data & Software may not be used to create fake, libelous,
misleading, or defamatory content of any kind excluding analyses in peer-reviewed
scientific research. The Data & Software may not be reproduced, modified and/or made
available in any form to any third party without Max-Planck’s prior written permission.

The Data & Software may not be used for pornographic purposes or to generate
pornographic material whether commercial or not. This license also prohibits the use
of the Software to train methods/algorithms/neural networks/etc. for commercial,
pornographic, military, surveillance, or defamatory use of any kind. By downloading
the Data & Software, you agree not to reverse engineer it.

No Distribution
The Model & Software and the license herein granted shall not be copied, shared,
distributed, re-sold, offered for re-sale, transferred or sub-licensed in whole or in
part except that you may make one copy for archive purposes only.

Disclaimer of Representations and Warranties
You expressly acknowledge and agree that the Model & Software results from basic
research, is provided “AS IS”, may contain errors, and that any use of the Model &
Software is at your sole risk. LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY
KIND CONCERNING THE MODEL & SOFTWARE, NEITHER EXPRESS NOR IMPLIED, AND THE ABSENCE OF
ANY LEGAL OR ACTUAL DEFECTS, WHETHER DISCOVERABLE OR NOT. Specifically, and not to
limit the foregoing, licensor makes no representations or warranties (i) regarding the
merchantability or fitness for a particular purpose of the Model & Software, (ii) that
the use of the Model & Software will not infringe any patents, copyrights or other
intellectual property rights of a third party, and (iii) that the use of the Model &
```

(continues on next page)

(continued from previous page)

````
Limitation of Liability
Because this Model & Software License Agreement qualifies as a donation, according to Section 521 of the German Civil Code (Bürgerliches Gesetzbuch – BGB) Licensor as a donor is liable for intent and gross negligence only. If the Licensor fraudulently conceals a legal or material defect, they are obliged to compensate the Licensee for the resulting damage.
Licensor shall be liable for loss of data only up to the amount of typical recovery costs which would have arisen had proper and regular data backup measures been taken. For the avoidance of doubt Licensor shall be liable in accordance with the German Product Liability Act in the event of product liability. The foregoing applies also to Licensor's legal representatives or assistants in performance. Any further liability shall be excluded.
Patent claims generated through the usage of the Model & Software cannot be directed towards the copyright holders.
The Model & Software is provided in the state of development the licensor defines. If modified or extended by Licensee, the Licensor makes no claims about the fitness of the Model & Software and is not responsible for any problems such modifications cause.

No Maintenance Services
You understand and agree that Licensor is under no obligation to provide either maintenance services, update services, notices of latent defects, or corrections of defects with regard to the Model & Software. Licensor nevertheless reserves the right to update, modify, or discontinue the Model & Software at any time.

Defects of the Model & Software must be notified in writing to the Licensor with a comprehensible description of the error symptoms. The notification of the defect should enable the reproduction of the error. The Licensee is encouraged to communicate any use, results, modification or publication.

Publications using the Model & Software
You acknowledge that the Model & Software is a valuable scientific resource and agree to appropriately reference the following paper in any publication making use of the Model & Software.

Citation:

```bibtex
@inproceedings{SMPL-X:2019,
  title = {Expressive Body Capture: 3D Hands, Face, and Body from a Single Image},
  author = {Pavlakos, Georgios and Choutas, Vasileios and Ghorbani, Nima and Bolkart, Timo and Osman, Ahmed A. A. and Tzionas, Dimitrios and Black, Michael J.},
  booktitle = {Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)},
  year = {2019}
}
```

Commercial licensing opportunities
For commercial uses of the Software, please send email to ps-license@tue.mpg.de

This Agreement shall be governed by the laws of the Federal Republic of Germany except for the UN Sales Convention.
````

CHAPTER

TWENTYSEVEN

APIS

- Multi-view single-person SMPL Estimator
- Multi-view multi-person SMPL Estimator

27.1 Multi-view single-person SMPL Estimator

`MultiViewSinglePersonSMPLEstimator` is an API class for multi-view single-person scenario, taking multi-view videos and multi-view camera parameters as input, estimating SMPL parameters and some other important information. See the *estimator doc* for details.

27.2 Multi-view multi-person SMPL Estimator

`MultiPersonSMPLEstimator` is an API class for multi-view multi-person scenario, taking multi-person keypoints3d and multi-view camera parameters as input, estimating SMPL parameters and some other important information. See the *estimator doc* for details.

CHAPTER

TWENTYEIGHT

XRMOCAP.CORE

28.1 estimation

28.2 evaluation

28.3 smplify hook

28.4 train

28.5 visualization

CHAPTER TWENTYNINE

XRMOCAP.DATA

29.1 data_converter

29.2 dataloader

29.3 dataset

29.4 data_visualization

CHAPTER

THIRTY

XRMOCAP.DATA_STRUCTURE

CHAPTER

THIRTYONE

XRMOCAP.HUMAN_PERCEPTION

31.1 bbox_detection

31.2 keypoints_estimation

CHAPTER THIRTYTWO

XRMOCAP.IO

CHAPTER

THIRTYTHREE

XRMOCAP.MODEL

33.1 architecture

CHAPTER

THIRTYFOUR

XRMOCAP.OPS

34.1 projection

CHAPTER

THIRTYFIVE

XRMOCAP.TRANSFORM

35.1 keypoints3d.optim

CHAPTER

THIRTYSIX

XRMOCAP.UTILS

xrmocap.utils.**project_point_radial**(*x*, *R*, *T*, *f*, *c*, *k*, *p*)

This function is to project a point in 3D space to 2D pixel space with given camera parameters.

Parameters

- **x** – Nx3 points in world coordinates
- **R** – 3x3 Camera rotation matrix
- **T** – 3x1 Camera translation parameters
- **f** – (scalar) Camera focal length
- **c** – 2x1 Camera center
- **k** – 3x1 Camera radial distortion coefficients
- **p** – 2x1 Camera tangential distortion coefficients

Returns **ypixel.T** – Nx2 points in pixel space

xrmocap.utils.**unfold_camera_param**(*camera: dict*)

This function is to extract camera extrinsic, intrinsic and distorsion parameters from dictionary.

Parameters **camera** (*dict*) – Dictionary to store the camera parameters.

Returns

- **R** (*Union[np.ndarray, torch.Tensor]*) – Extrinsic parameters, rotation matrix.
- **T** (*Union[np.ndarray, torch.Tensor]*) – Extrinsic parameters, translation matrix.
- **f** (*Union[np.ndarray, torch.Tensor]*) – Focal length in x, y direction.
- **c** (*Union[np.ndarray, torch.Tensor]*) – Camera center.
- **k** (*Union[list, torch.Tensor]*) – Radial distortion coefficients.
- **p** (*Union[list, torch.Tensor]*) – Tangential distortion coefficients.

CHAPTER

THIRTYSEVEN

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

X

INDEX