MMHuman3D

Release 0.11.0

MMHuman3D Authors

Apr 05, 2023

GET STARTED

CHAPTER

ONE

INSTALLATION

- Requirements
- Prepare environment
- Install MMHuman3D
- A from-scratch setup script

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+
- PyTorch3D 0.4+
- MMCV (Please install mmcv-full>=1.3.17,<1.6.0 for GPU)

Optional:

- MMPOSE (Only for demo.)
- MMDETECTION (Only for demo.)
- MMTRACKING (Only for multi-person demo. If you use mmtrack, please install mmcls<0.23.1, mmcv-full>=1.3.17,<1.6.0 for GPU.)

1.2 Prepare environment

a. Create a conda virtual environment and activate it.

```
conda create -n open-mmlab python=3.8 -y
conda activate open-mmlab
```

b. Install ffmpeg

Install ffmpeg with conda directly and the libx264 will be built automatically.

```
conda install ffmpeg
```

c. Install PyTorch and torchvision following the official instructions.

```
conda install pytorch={torch_version} torchvision cudatoolkit={cu_version} -c pytorch
```

E.g., install PyTorch 1.8.0 & CUDA 10.2.

```
conda install pytorch=1.8.0 torchvision cudatoolkit=10.2 -c pytorch
```

Important: Make sure that your compilation CUDA version and runtime CUDA version match. Besides, for RTX 30 series GPU, cudatoolkit>=11.0 is required.

d. Install PyTorch3D from source.

For Linux:

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

conda install pytorch3d -c pytorch3d
```

Users may also refer to PyTorch3D-install for more details. However, our recent tests show that installing using `conda` sometimes runs into dependency conflicts. Hence, users may alternatively install Pytorch3D from source following the steps below.

```
git clone https://github.com/facebookresearch/pytorch3d.git
cd pytorch3d
pip install .
cd ..
```

For Windows:

Please refer to official installation for details. Here we provide an example for user reference. **Important:** This section is for users who want to install MMHuman3D on Windows.

Your installation is successful if you can do these in command line.

```
echo "import pytorch3d;print(pytorch3d.__version__); \
    from pytorch3d.renderer import MeshRenderer;print(MeshRenderer);\
    from pytorch3d.structures import Meshes;print(Meshes);\
    from pytorch3d.renderer import cameras;print(cameras);\
    from pytorch3d.transforms import Transform3d;print(Transform3d);"|python

echo "import torch;device=torch.device('cuda');\
    from pytorch3d.utils import torus;\
    Torus = torus(r=10, R=20, sides=100, rings=100, device=device);\
    print(Torus.verts_padded());"|python
```

1.3 Install MMHuman3D

a. Build mmcv-full & mmpose & mmdet & mmtrack

- mmcv-full

We recommend you to install the pre-build package as below.

For CPU:

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cpu/{torch_version}/
↪index.html
```

Please replace `{torch_version}` in the url to your desired one.

For GPU:

```
pip install "mmcv-full>=1.3.17,<=1.5.3" -f https://download.openmmlab.com/mmcv/dist/{cu_
↪version}/{torch_version}/index.html
```

Please replace `{cu_version}` and `{torch_version}` in the url to your desired one.

For example, to install mmcv-full with CUDA 10.2 and PyTorch 1.8.0, use the following command:

```
pip install "mmcv-full>=1.3.17,<=1.5.3" -f https://download.openmmlab.com/mmcv/dist/
↪cu102/torch1.8.0/index.html
```

See here for different versions of MMCV compatible to different PyTorch and CUDA versions. For more version download link, refer to openmmlab-download.

Optionally you can choose to compile mmcv from source by the following command

```
git clone https://github.com/open-mmlab/mmcv.git -b v1.5.3
cd mmcv
MMCV_WITH_OPS=1 pip install -e .  # package mmcv-full, which contains cuda ops, will be␣
↪installed after this step
# OR pip install -e .  # package mmcv, which contains no cuda ops, will be installed␣
↪after this step
cd ..
```

Important: You need to run `pip uninstall mmcv` first if you have mmcv installed. If mmcv and mmcv-full are both installed, there will be `ModuleNotFoundError`.

- mmdetection (optional)

```
pip install "mmdet<=2.25.1"
```

Alternatively, you can also build MMDetection from source in case you want to modify the code:

```
git clone https://github.com/open-mmlab/mmdetection.git -b v2.25.1
cd mmdetection
pip install -r requirements/build.txt
pip install -v -e .
```

- mmpose (optional)

```
pip install "mmpose<=0.28.1"
```

Alternatively, you can also build MMPose from source in case you want to modify the code:

```
git clone https://github.com/open-mmlab/mmpose.git -b v0.28.1
cd mmpose
pip install -r requirements.txt
pip install -v -e .
```

- mmtracking (optional)

```
pip install "mmcls<=0.23.2" "mmtrack<=0.13.0"
```

Alternatively, you can also build MMTracking from source in case you want to modify the code:

```
git clone https://github.com/open-mmlab/mmtracking.git -b v0.13.0
cd mmtracking
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

b. Clone the mmhuman3d repository.

```
git clone https://github.com/open-mmlab/mmhuman3d.git
cd mmhuman3d
```

c. Install build requirements and then install mmhuman3d.

```
pip install -v -e .  # or "python setup.py develop"
```

1.4 A from-scratch setup script

```
# Create conda environment
conda create -n open-mmlab python=3.8 -y
conda activate open-mmlab

# Install ffmpeg
conda install ffmpeg

# Install PyTorch
conda install pytorch==1.8.0 torchvision cudatoolkit=10.2 -c pytorch -y

# Install PyTorch3D
conda install -c fvcore -c iopath -c conda-forge fvcore iopath -y
conda install -c bottler nvidiacub -y
conda install pytorch3d -c pytorch3d -y
# Alternatively from source in case of dependency conflicts
# git clone https://github.com/facebookresearch/pytorch3d.git
# cd pytorch3d
# pip install .
# cd ..

# Install mmcv-full
pip install "mmcv-full>=1.3.17,<1.6.0" -f https://download.openmmlab.com/mmcv/dist/cu102/
↪torch1.8.0/index.html
```

(continues on next page)

(continued from previous page)

```
# Optional: install mmdetection & mmpose & mmtracking
pip install "mmdet<=2.25.1"
pip install "mmpose<=0.28.1"
pip install "mmcls<=0.23.2" "mmtrack<=0.13.0"

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

CHAPTER

TWO

GETTING STARTED

- Getting Started
  - Installation
  - Data Preparation
  - Body Model Preparation
  - Inference / Demo
    - Offline Demo
    - Online Demo
  - Evaluation
    - Evaluate with a single GPU / multiple GPUs
    - Evaluate with slurm
  - Training
    - Training with a single / multiple GPUs
    - Training with Slurm
  - More Tutorials

2.1 Installation

Please refer to *install.md* for installation.

2.2 Data Preparation

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

2.3 Body Model Preparation

- 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`
- J_regressor_extra.npy
- J_regressor_h36m.npy
- smpl_mean_params.npz

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

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── body_models
        ├── J_regressor_extra.npy
        ├── J_regressor_h36m.npy
        ├── smpl_mean_params.npz
        └── smpl
            ├── SMPL_FEMALE.pkl
            ├── SMPL_MALE.pkl
            └── SMPL_NEUTRAL.pkl
```

2.4 Inference / Demo

2.4.1 Offline Demo

We provide a demo script to estimate SMPL parameters for single-person or multi-person from the input image or video with the bounding box detected by MMDetection or MMTracking. With this demo script, you only need to choose a pre-trained model (we currently only support HMR, SPIN, VIBE and PARE, more SOTA methods will be added in the future) from our model zoo and specify a few arguments, and then you can get the estimated results.

Some useful configs are explained here:

- If you specify `--output` and `--show_path`, the demo script will save the estimated results into `human_data` and render the estimated human mesh.
- If you specify `--smooth_type`, the demo will be smoothed using specific method. We now support filters `gaus1d`,`oneeuro`, `savgol` and learning-based method `smoothnet`, more information can be find here.
- If you specify `--speed_up_type`, the demo will be processed more quickly using specific method. We now support learning-based method `deciwatch`, more information can be find here.

For single-person:

```
python demo/estimate_smpl.py \
    ${MMHUMAN3D_CONFIG_FILE} \
    ${MMHUMAN3D_CHECKPOINT_FILE} \
    --single_person_demo \
    --det_config ${MMDET_CONFIG_FILE} \
    --det_checkpoint ${MMDET_CHECKPOINT_FILE} \
    --input_path ${VIDEO_PATH_OR_IMG_PATH} \
    [--show_path ${VIS_OUT_PATH}] \
    [--output ${RESULT_OUT_PATH}] \
    [--smooth_type ${SMOOTH_TYPE}] \
    [--speed_up_type ${SPEED_UP_TYPE}] \
    [--draw_bbox] \
```

Example:

```
python demo/estimate_smpl.py \
    configs/hmr/resnet50_hmr_pw3d.py \
    data/checkpoints/resnet50_hmr_pw3d.pth \
    --single_person_demo \
    --det_config demo/mmdetection_cfg/faster_rcnn_r50_fpn_coco.py \
    --det_checkpoint https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_
↪rcnn_r50_fpn_1x_coco/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --input_path  demo/resources/single_person_demo.mp4 \
    --show_path vis_results/single_person_demo.mp4 \
    --output demo_result \
    --smooth_type savgol \
    --speed_up_type deciwatch \
    --draw_bbox
```

For multi-person:

```
python demo/estimate_smpl.py \
    ${MMHUMAN3D_CONFIG_FILE} \
    ${MMHUMAN3D_CHECKPOINT_FILE} \
    --multi_person_demo \
    --tracking_config ${MMTRACKING_CONFIG_FILE} \
    --input_path ${VIDEO_PATH_OR_IMG_PATH} \
    [--show_path ${VIS_OUT_PATH}] \
    [--output ${RESULT_OUT_PATH}] \
    [--smooth_type ${SMOOTH_TYPE}] \
    [--speed_up_type ${SPEED_UP_TYPE}] \
    [--draw_bbox]
```

Example:

```
python demo/estimate_smpl.py \
    configs/hmr/resnet50_hmr_pw3d.py \
    data/checkpoints/resnet50_hmr_pw3d.pth \
    --multi_person_demo \
    --tracking_config demo/mmtracking_cfg/deepsort_faster-rcnn_fpn_4e_mot17-private-half.
↪py \
    --input_path  demo/resources/multi_person_demo.mp4 \
    --show_path vis_results/multi_person_demo.mp4 \
```

(continues on next page)

(continued from previous page)

```
    --smooth_type savgol \
    --speed_up_type deciwatch \
    [--draw_bbox]
```

Note that the MMHuman3D checkpoints can be downloaded from the *model zoo*. Here we take HMR (resnet50_hmr_pw3d.pth) as an example.

2.4.2 Online Demo

We provide a webcam demo script to estimate SMPL parameters from the camera or a specified video file. You can simply run the following command:

```
python demo/webcam_demo.py
```

Some useful arguments are explained here:

- If you specify `--output`, the webcam demo script will save the visualization results into a file. This may reduce the frame rate.
- If you specify `--synchronous`, video I/O and inference will be temporally aligned. Note that this will reduce the frame rate.
- If you want run the webcam demo in offline mode on a video file, you should set `--cam-id=VIDEO_FILE_PATH`. Note that `--synchronous` should be set to `True` in this case.
- The video I/O and model inference are running asynchronously and the latter usually takes more time for a single frame. To allevidate the time delay, you can:
  - set `--display-delay=MILLISECONDS` to defer the video stream, according to the inference delay shown at the top left corner. Or,
  - set `--synchronous=True` to force video stream being aligned with inference results. This may reduce the frame rate.

2.5 Evaluation

We provide pretrained models in the respective method folders in config.

2.5.1 Evaluate with a single GPU / multiple GPUs

```
python tools/test.py ${CONFIG} --work-dir=${WORK_DIR} ${CHECKPOINT} --metrics=${METRICS}
```

Example:

```
python tools/test.py configs/hmr/resnet50_hmr_pw3d.py --work-dir=work_dirs/hmr work_dirs/
↪hmr/latest.pth --metrics pa-mpjpe mpjpe
```

2.5.2 Evaluate with slurm

If you can run MMHuman3D on a cluster managed with slurm, you can use the script `slurm_test.sh`.

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG} ${WORK_DIR} ${CHECKPOINT} --
↪metrics ${METRICS}
```

Example:

```
./tools/slurm_test.sh my_partition test_hmr configs/hmr/resnet50_hmr_pw3d.py work_dirs/
↪hmr work_dirs/hmr/latest.pth 8 --metrics pa-mpjpe mpjpe
```

2.6 Training

2.6.1 Training with a single / multiple GPUs

```
python tools/train.py ${CONFIG_FILE} ${WORK_DIR} --no-validate
```

Example: using 1 GPU to train HMR.

```
python tools/train.py ${CONFIG_FILE} ${WORK_DIR} --gpus 1 --no-validate
```

2.6.2 Training with Slurm

If you can run MMHuman3D on a cluster managed with slurm, you can use the script `slurm_train.sh`.

```
./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_DIR} ${GPU_NUM} --
↪no-validate
```

Common optional arguments include:

- `--resume-from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.
- `--no-validate`: Whether not to evaluate the checkpoint during training.

Example: using 8 GPUs to train HMR on a slurm cluster.

```
./tools/slurm_train.sh my_partition my_job configs/hmr/resnet50_hmr_pw3d.py work_dirs/
↪hmr 8 --no-validate
```

You can check slurm_train.sh for full arguments and environment variables.

2.7 More Tutorials

- *Camera conventions*
- *Keypoint conventions*
- *Custom keypoint conventions*
- *HumanData*
- *Keypoint visualization*

- *Mesh visualization*

CHAPTER

THREE

BENCHMARK AND MODEL ZOO

We provide configuration files, log files and pretrained models for all supported methods. Moreover, all pretrain models are evaluated on three common benchmarks: 3DPW, Human3.6M, and MPI-INF-3DHP.

3.1 Baselines

3.1.1 HMR

Please refer to HMR for details.

3.1.2 SPIN

Please refer to SPIN for details.

3.1.3 VIBE

Please refer to VIBE for details.

3.1.4 HybrIK

Please refer to HybrIK for details.

3.1.5 PARE

Please refer to PARE for details.

3.1.6 ExPose

Please refer to ExPose for details.

3.1.7 PyMAF-X

Please refer to PyMAF-X for details.

CHAPTER

FOUR

HUMANDATA

4.1 Overview

HumanData is a subclass of python built-in class dict, containing single-view, image-based data for a human being. It has a well-defined base structure for universal data, but it is also compatible with customized data for new features. A native HumanData contains values in numpy.ndarray or python built-in types, it holds no data in torch.Tensor, but you can convert arrays to torch.Tensor(even to GPU Tensor) by `human_data.to()` easily.

4.2 Key/Value definition

4.2.1 The keys and values supported by HumanData are described as below.

- image_path: (N,), list of str, each element is a relative path from the root folder (exclusive) to the image.
- bbox_xywh: (N, 5), numpy array, bounding box with confidence, coordinates of bottom-left point x, y, width w and height h of bbox, score at last.
- config: (), str, the flag name of config for individual dataset.
- keypoints2d: (N, 190, 3), numpy array, 2d joints of smplx model with confidence, joints from each datasets are mapped to HUMAN_DATA joints.
- keypoints3d: (N, 190, 4), numpy array, 3d joints of smplx model with confidence. Same as above.
- smpl: (1,), dict, keys are [‘body_pose’: numpy array, (N, 23, 3), ‘global_orient’: numpy array, (N, 3), ‘betas’: numpy array, (N, 10), ‘transl’: numpy array, (N, 3)].
- smplx: (1,), dict, keys are [‘body_pose’: numpy array, (N, 21, 3),’global_orient’: numpy array, (N, 3), ‘betas’: numpy array, (N, 10), ‘transl’: numpy array, (N, 3), ‘left_hand_pose’: numpy array, (N, 15, 3), ‘right_hand_pose’: numpy array, (N, 15, 3), ‘expression’: numpy array (N, 10), ‘leye_pose’: numpy array (N, 3), ‘reye_pose’: (N, 3), ‘jaw_pose’: numpy array (N, 3)].
- meta: (1,), dict, its keys are meta data from dataset like ‘gender’.
- keypoints2d_mask: (190,), numpy array, mask for which keypoint is valid in keypoints2d. 0 means that the joint in this position cannot be found in original dataset.
- keypoints3d_mask: (190,), numpy array, mask for which keypoint is valid in keypoints3d. 0 means that the joint in this position cannot be found in original dataset.
- misc: (1,), dict, keys and values are defined by user. The space misc takes(sys.getsizeof(misc)) shall be no more than 6MB.

4.2.2 Key check in HumanData.

Only keys above are allowed as top level key in a default HumanData. If you cannot work with that, there's also a way out. Construct a HumanData instance with `__key_strict__ == False`:

```
human_data = HumanData.new(key_strict=False)
human_data['video_path'] = 'test.mp4'
```

The returned human_data will allow any customized keys, logging a warning at the first time HumanData sees a new key. Just ignore the warning if you do know that you are using a customized key, it will not appear again before the program ends.

If you have already constructed a HumanData, and you want to change the strict mode, use `set_key_strict`:

```
human_data = HumanData.fromfile('human_data.npz')
key_strict = human_data.get_key_strict()
human_data.set_key_strict(not key_strict)
```

4.2.3 Value check in HumanData.

Only values above will be check when `human_data[key] == value` is called, and the constraints are defined in `HumanData.SUPPORTED_KEYS`.

For each value, an exclusive type must be specified under its key:

```
'smpl': {
    'type': dict,
},
```

For value as numpy.ndarray, shape and dim shall be defined:

```
'keypoints3d': {
    'type': np.ndarray,
    'shape': (-1, -1, 4),
    # value.ndim==3, and value.shape[2]==4
    # value.shape[0:2] is arbitrary.
    'dim': 0
    # dimension 0 marks time(frame index, or second)
},
```

For value which is constant along frame axis, set dim to -1 to ignore frame check:

```
'keypoints3d_mask': {
    'type': np.ndarray,
    'shape': (-1, ),
    'dim': -1
},
```

4.3 Data compression

4.3.1 Compression with mask

As the keypoint convention named HUMAN_DATA is a union of keypoint definitions from various datasets, it is common that some keypoints are missing. In this situation, the missing ones are filtered by mask:

```
# keypoints2d_agora is a numpy array in shape [frame_num, 127, 3].
# There are 127 keypoints defined by agora.
keypoints2d_human_data, mask = convert_kps(keypoints2d_agora, 'agora', 'human_data')
# keypoints2d_human_data is a numpy array in shape [frame_num, 190, 3], only 127/190 are␣
↪valid
# mask is a numpy array in shape [190, ], with 127 ones and 63 zeros inside
```

Set `keypoints2d_mask` and `keypoints2d`. It is obvious that there are redundant zeros in keypoints2d:

```
human_data = HumanData()
human_data['keypoints2d_mask'] = mask
human_data['keypoints2d'] = keypoints2d_human_data
```

Call `compress_keypoints_by_mask()` to get rid of the zeros. This method checks if any key containing `keypoints` has a corresponding mask, and performs keypoints compression if both keypoints and masks are present. :

```
human_data.compress_keypoints_by_mask()
```

Call `get_raw_value()` to get the compressed raw value stored in HumanData instance. When getting item with `[]`, the keypoints padded with zeros will be returned:

```
keypoints2d_human_data = human_data.get_raw_value('keypoints2d')
print(keypoints2d_human_data.shape)  # [frame_num, 127, 3]
keypoints2d_human_data = human_data['keypoints2d']
print(keypoints2d_human_data.shape)  # [frame_num, 190, 3]
```

In `keypoints_compressed` mode, keypoints are allowed to be edited. There are two different ways, set with padded data or set the compressed data directly:

```
padded_keypoints2d = np.zeros(shape=[100, 190, 3])
human_data['keypoints2d'] = padded_keypoints2d  # [frame_num, 190, 3]
compressed_keypoints2d = np.zeros(shape=[100, 127, 3])
human_data.set_raw_value('keypoints2d', compressed_keypoints2d)  # [frame_num, 127, 3]
```

When a HumanData instance is in `keypoints_compressed` mode, all masks of keypoints are locked. If you are trying to edit it, a warning will be logged and the value won't change. To modify a mask, de-compress it with `decompress_keypoints()`:

```
human_data.decompress_keypoints()
```

Features above also work with any key pairs like `keypoints*` and `keypoints*_mask`.

4.3.2 Compression for file

Call `dump()` to save HumanData into a compressed `.npz` file.

The dumped file can be load by `load()` :

```
# save
human_data.dump('./dumped_human_data.npz')
# load
another_human_data = HumanData()
another_human_data.load('./dumped_human_data.npz')
```

Sometimes a HumanData instanse is too large to dump, an error will be raised by `numpy.savez_compressed()`. In this case, call `dump_by_pickle` and `load_by_pickle` for file operation.

4.3.3 Compression by key

If a HumanData instance is in not in key_strict mode, it may contains unsupported items which are not necessary. Call `pop_unsupported_items()` to remove those items will save space for you:

```
human_data = HumanData.fromfile('human_data_not_strict.npz')
human_data.pop_unsupported_items()
# set instance.__key_strict__ from True to False will also do
human_data.set_key_strict(True)
```

4.4 Data selection

4.4.1 Select by shape

Assume that `keypoints2d` is an array in shape [200, 190, 3], only the first 10 frames are needed:

```
first_ten_frames = human_data.get_value_in_shape('keypoints2d', shape=[10, -1, -1])
```

In some situation, we need to pad all arrays to a certain size:

```
# pad keypoints2d from [200, 190, 3] to [200, 300, 3] with zeros
padded_keypoints2d = human_data.get_value_in_shape('keypoints2d', shape=[200, 300, -1])
# padding value can be modified
padded_keypoints2d = human_data.get_value_in_shape('keypoints2d', shape=[200, 300, -1],␣
↪padding_constant=1)
```

4.4.2 Select temporal slice

Assume that there are 200 frames in a HumanData instance, only data between 10 and 20 are needed:

```
# all supported values will be sliced
sub_human_data = human_data.get_slice(10, 21)
```

Downsample is also supported, for example, select 33%:

```
# select [0, 3, 6, 9,..., 198]
sub_human_data = human_data.get_slice(0, 200, 3)
```

4.5 To torch.Tensor

As introduced, a native HumanData contains values in numpy.ndarray or python built-in types, but the numpy.ndarray can be easily convert to torch.Tensor:

```
# All values as ndarray will be converted to a cpu Tensor.
# Values in other types will not change.
# It returns a dict like HumanData.
dict_of_tensor = human_data.to()
# GPU is also supported
gpu0_device = torch.device('cuda:0')
dict_of_gpu_tensor = human_data.to(gpu0_device)
```

CHAPTER

FIVE

MULTIHUMANDATA

MulitHumanData is designed to support multi-human body mesh recovery, who inherits from HumanData. In HumanData, the data can be accessed directly through the index, because the data and the image are in one-to-one correspondence. However, data and image have a many-to-one correspondence in MultiHumanData.

Based on HumanData, MultiHumanData adds a new key named `'frame_range'` as follows:

```
'frame_range': {
        'type': np.ndarray,
        'shape': (-1, 2),
        'dim': 0
    }
```

`frame_range` and image are in one-to-one correspondence. Each element in `frame_range` has two pointers that point to a data-block.

Suppose we have an instance of MultiHumanData and we want to access the data corresponding to the i-th image. First, we index the `frame_range` using primary index i, which will return two points. We then use these two pointers to access all data corresponding to the i-th image.

```
image_0  ----> human_0      <--- frame_range[0][0]
         -       .
          -      .
           --> human_(n-1)  <--- frame_range[0][0] + (n-1)
            -> human_n      <--- frame_range[0][1]
    .
    .
    .


image_n  ----> human_0     <--- frame_range[n][0]
         -       .
          -      .
           --> human_(n-1)  <--- frame_range[n][0] + (n-1)
            -> human_n     <--- frame_range[n][1]
```

CHAPTER

SIX

DATA PREPARATION

- Datasets for supported algorithms
- Folder structure
  - AGORA
  - COCO
  - COCO-WholeBody
  - CrowdPose
  - EFT
  - GTA-Human
  - Human3.6M
  - Human3.6M Mosh
  - HybrIK
  - LSP
  - LSPET
  - MPI-INF-3DHP
  - MPII
  - PoseTrack18
  - Penn Action
  - PW3D
  - SPIN
  - SURREAL

6.1 Overview

Our data pipeline use *HumanData* structure for storing and loading. The proprocessed npz files can be obtained from raw data using our data converters, and the supported configs can be found here.

These are our supported converters and their respective `dataset-name`:

- AgoraConverter (`agora`)
- AmassConverter (`amass`)
- CocoConverter (`coco`)
- CocoHybrIKConverter (`coco_hybrik`)
- CocoWholebodyConverter (`coco_wholebody`)
- CrowdposeConverter (`crowdpose`)
- EftConverter (`eft`)
- GTAHumanConverter (`gta_human`)
- H36mConverter (`h36m_p1`, `h36m_p2`)
- H36mHybrIKConverter (`h36m_hybrik`)
- InstaVibeConverter (`instavariety_vibe`)
- LspExtendedConverter (`lsp_extended`)
- LspConverter (`lsp_original`, `lsp_dataset`)
- MpiiConverter (`mpii`)
- MpiInf3dhpConverter (`mpi_inf_3dhp`)
- MpiInf3dhpHybrIKConverter (`mpi_inf_3dhp_hybrik`)
- PennActionConverter (`penn_action`)
- PosetrackConverter (`posetrack`)
- Pw3dConverter (`pw3d`)
- Pw3dHybrIKConverter (`pw3d_hybrik`)
- SurrealConverter (`surreal`)
- SpinConverter (`spin`)
- Up3dConverter (`up3d`)

6.2 Datasets for supported algorithms

For all algorithms, the root path for our datasets and output path for our preprocessed npz files are stored in `data/datasets` and `data/preprocessed_datasets`. As such, use this command with the listed `dataset-names`:

```
python tools/convert_datasets.py \
  --datasets <dataset-name> \
  --root_path data/datasets \
  --output_path data/preprocessed_datasets
```

For HMR training and testing, the following datasets are required:

- COCO
- Human3.6M
- Human3.6M Mosh
- MPI-INF-3DHP
- MPII
- LSP
- LSPET
- PW3D

Convert datasets with the following `dataset-names`:

```
coco, pw3d, mpii, mpi_inf_3dhp, lsp_original, lsp_extended, h36m
```

Alternatively, you may download the preprocessed files directly:

- cmu_mosh.npz
- coco_2014_train.npz
- h36m_train.npz
- lsp_train.npz
- lspet_train.npz
- mpi_inf_3dhp_train.npz
- mpii_train.npz
- pw3d_test.npz

Unfortunately, we are unable to distribute `h36m_mosh_train.npz` due to license limitations. However, we provide the conversion tools should you possess the raw mosh data. Prefer refer to Human3.6M Mosh on details for conversion.

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
        ├── coco_2014_train.npz
        ├── h36m_train.npz or h36m_mosh_train.npz (if mosh is available)
        ├── lspet_train.npz
        ├── lsp_train.npz
        ├── mpi_inf_3dhp_train.npz
        ├── mpii_train.npz
        └── pw3d_test.npz
```

For SPIN training, the following datasets are required:

- COCO

- Human3.6M
- Human3.6M Mosh
- MPI-INF-3DHP
- MPII
- LSP
- LSPET
- PW3D
- SPIN

Convert datasets with the following `dataset-names`:

```
spin, h36m
```

Alternatively, you may download the preprocessed files directly:

- spin_coco_2014_train.npz
- h36m_train.npz
- spin_lsp_train.npz
- spin_lspet_train.npz
- spin_mpi_inf_3dhp_train.npz
- spin_mpii_train.npz
- spin_pw3d_test.npz

Unfortunately, we are unable to distribute `h36m_mosh_train.npz` due to license limitations. However, we provide the conversion tools should you posses the raw mosh data. Prefer refer to Human3.6M Mosh on details for conversion.

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
        ├── spin_coco_2014_train.npz
        ├── h36m_train.npz or h36m_mosh_train.npz (if mosh is available)
        ├── spin_lsp_train.npz
        ├── spin_lspet_train.npz
        ├── spin_mpi_inf_3dhp_train.npz
        ├── spin_mpii_train.npz
        └── spin_pw3d_test.npz
```

For VIBE training and testing, the following datasets are required:

- MPI-INF-3DHP
- PW3D

The data converters are currently not available.

Alternatively, you may download the preprocessed files directly:

- vibe_insta_variety.npz
- vibe_mpi_inf_3dhp_train.npz
- vibe_pw3d_test.npz

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
        ├── vibe_insta_variety.npz
        ├── vibe_mpi_inf_3dhp_train.npz
        └── vibe_pw3d_test.npz
```

For HYBRIK training and testing, the following datasets are required:

- HybrIK
- COCO
- Human3.6M
- MPI-INF-3DHP
- PW3D

Convert datasets with the following `dataset-names`:

```
h36m_hybrik, pw3d_hybrik, mpi_inf_3dhp_hybrik, coco_hybrik
```

Alternatively, you may download the preprocessed files directly:

- hybriK_coco_2017_train.npz
- hybrik_h36m_train.npz
- hybrik_mpi_inf_3dhp_train.npz
- hybrik_pw3d_test.npz

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
```

(continues on next page)

(continued from previous page)

```
        ├── hybriK_coco_2017_train.npz
        ├── hybrik_h36m_train.npz
        ├── hybrik_mpi_inf_3dhp_train.npz
        └── hybrik_pw3d_test.npz
```

For PARE training, the following datasets are required:

- Human3.6M
- Human3.6M Mosh
- MPI-INF-3DHP
- EFT-COCO
- EFT-MPII
- EFT-LSPET
- PW3D

Convert datasets with the following `dataset-names`:

```
h36m, coco, mpii, lspet, mpi-inf-3dhp, pw3d
```

Alternatively, you may download the preprocessed files directly:

- h36m_train.npz
- mpi_inf_3dhp_train.npz
- eft_mpii.npz
- eft_lspet.npz
- eft_coco_all.npz
- pw3d_test.npz

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
        ├── h36m_mosh_train.npz
        ├── h36m_train.npz
        ├── mpi_inf_3dhp_train.npz
        ├── eft_mpii.npz
        ├── eft_lspet.npz
        ├── eft_coco_all.npz
        └── pw3d_test.npz
```

For ExPose training, the following datasets are required:

- Human3.6M

- FreiHand
- EHF
- FFHQ
- ExPose-Curated-fits
- SPIN_SMPLX
- Stirling-ESRC3D
- PW3D

Convert datasets with the following `dataset-names`:

```
h36m, EHF, FreiHand, 3DPW, stirling, spin_in_smplx, ffhq, ExPose_curated_fits
```

Alternatively, you may download the preprocessed files directly:

- curated_fits_train.npz
- ehf_val.npz
- ffhq_flame_train.npz
- freihand_test.npz
- freihand_train.npz
- freihand_val.npz
- h36m_smplx_train.npz
- spin_smplx_train.npz
- stirling_ESRC3D_HQ.npz
- pw3d_test.npz

The preprocessed datasets should have this structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── datasets
    └── preprocessed_datasets
        ├── curated_fits_train.npz
        ├── ehf_val.npz
        ├── ffhq_flame_train.npz
        ├── freihand_test.npz
        ├── freihand_train.npz
        ├── freihand_val.npz
        ├── h36m_smplx_train.npz
        ├── pw3d_test.npz
        ├── spin_smplx_train.npz
        └── stirling_ESRC3D_HQ.npz
```

6.3 Folder structure

6.3.1 AGORA

```
@inproceedings{Patel:CVPR:2021,
 title = {{AGORA}: Avatars in Geography Optimized for Regression Analysis},
 author = {Patel, Priyanka and Huang, Chun-Hao P. and Tesch, Joachim and Hoffmann, David␣
↪T. and Tripathi, Shashank and Black, Michael J.},
 booktitle = {Proceedings IEEE/CVF Conf.~on Computer Vision and Pattern Recognition (
↪{CVPR})},
 month = jun,
 year = {2021},
 month_numeric = {6}
}
```

For AGORA, please download the dataset and place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── agora
            ├── camera_dataframe # smplx annotations
            │   ├── train_0_withjv.pkl
            │   ├── validation_0_withjv.pkl
            │   └── ...
            ├── camera_dataframe_smpl # smpl annotations
            │   ├── train_0_withjv.pkl
            │   ├── validation_0_withjv.pkl
            │   └── ...
            ├── images
            │   ├── train
            │   │   ├── ag_trainset_3dpeople_bfh_archviz_5_10_cam00_00000_1280x720.png
            │   │   ├── ag_trainset_3dpeople_bfh_archviz_5_10_cam00_00001_1280x720.png
            │   │   └── ...
            │   ├── validation
            │   └── test
            ├── smpl_gt
            │   ├── trainset_3dpeople_adults_bfh
            │   │   ├── 10004_w_Amaya_0_0.mtl
            │   │   ├── 10004_w_Amaya_0_0.obj
            │   │   ├── 10004_w_Amaya_0_0.pkl
            │   │   └── ...
            │   └── ...
            └── smplx_gt
                    ├── 10004_w_Amaya_0_0.obj
                    ├── 10004_w_Amaya_0_0.pkl
                    └── ...
```

6.3.2 AMASS

```
@inproceedings{AMASS:2019,
  title={AMASS: Archive of Motion Capture as Surface Shapes},
  author={Mahmood, Naureen and Ghorbani, Nima and F. Troje, Nikolaus and Pons-Moll,␣
↪Gerard and Black, Michael J.},
  booktitle = {The IEEE International Conference on Computer Vision (ICCV)},
  year={2019},
  month = {Oct},
  url = {https://amass.is.tue.mpg.de},
  month_numeric = {10}
}
```

Details for direct preprocessing will be added in the future.

Alternatively, you may download the preprocessed files directly:

- amass_smplh.npz
- amass_smplx.npz

6.3.3 COCO

```
@inproceedings{lin2014microsoft,
  title={Microsoft coco: Common objects in context},
  author={Lin, Tsung-Yi and Maire, Michael and Belongie, Serge and Hays, James and␣
↪Perona, Pietro and Ramanan, Deva and Doll{\'a}r, Piotr and Zitnick, C Lawrence},
  booktitle={European conference on computer vision},
  pages={740--755},
  year={2014},
  organization={Springer}
}
```

For COCO data, please download from COCO download. COCO'2014 Train is needed for HMR training and COCO'2017 Train is needed for HybrIK trainig. Download and extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── coco
            ├── annotations
            |   ├── person_keypoints_train2014.json
            |   ├── person_keypoints_val2014.json
            ├── train2014
            |   ├── COCO_train2014_000000000009.jpg
            |   ├── COCO_train2014_000000000025.jpg
            |   ├── COCO_train2014_000000000030.jpg
```

(continues on next page)

(continued from previous page)

```
|   └── ...
└── train_2017
    ├── annotations
    |   ├── person_keypoints_train2017.json
    |   └── person_keypoints_val2017.json
    ├── train2017
    |   ├── 000000000009.jpg
    |   ├── 000000000025.jpg
    |   ├── 000000000030.jpg
    |   └── ...
    └── val2017
        ├── 000000000139.jpg
        ├── 000000000285.jpg
        ├── 000000000632.jpg
        └── ...
```

6.3.4 COCO-WholeBody

```
@inproceedings{jin2020whole,
  title={Whole-Body Human Pose Estimation in the Wild},
  author={Jin, Sheng and Xu, Lumin and Xu, Jin and Wang, Can and Liu, Wentao and Qian,␣
↪Chen and Ouyang, Wanli and Luo, Ping},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  year={2020}
}
```

For COCO-WholeBody dataset, images can be downloaded from COCO download, 2017 Train/Val is needed for COCO keypoints training and validation. Download and extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── coco
            ├── annotations
            |   ├── coco_wholebody_train_v1.0.json
            |   └── coco_wholebody_val_v1.0.json
            └── train_2017
                ├── train2017
                |   ├── 000000000009.jpg
                |   ├── 000000000025.jpg
                |   ├── 000000000030.jpg
                |   └── ...
                └── val2017
                    ├── 000000000139.jpg
```

(continues on next page)

(continued from previous page)

```
                    ├── 000000000285.jpg
                    ├── 000000000632.jpg
                    └── ...
```

6.3.5 CrowdPose

```
@article{li2018crowdpose,
  title={CrowdPose: Efficient Crowded Scenes Pose Estimation and A New Benchmark},
  author={Li, Jiefeng and Wang, Can and Zhu, Hao and Mao, Yihuan and Fang, Hao-Shu and␣
↪Lu, Cewu},
  journal={arXiv preprint arXiv:1812.00324},
  year={2018}
}
```

For CrowdPose data, please download from CrowdPose. Download and extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── crowdpose
            ├── crowdpose_train.json
            ├── crowdpose_val.json
            ├── crowdpose_trainval.json
            ├── crowdpose_test.json
            └── images
                ├── 100000.jpg
                ├── 100001.jpg
                ├── 100002.jpg
                └── ...
```

6.3.6 EFT

```
@inproceedings{joo2020eft,
 title={Exemplar Fine-Tuning for 3D Human Pose Fitting Towards In-the-Wild 3D Human Pose␣
↪Estimation},
 author={Joo, Hanbyul and Neverova, Natalia and Vedaldi, Andrea},
 booktitle={3DV},
 year={2020}
}
```

For EFT data, please download from EFT. Download and extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── eft
            ├── coco_2014_train_fit
            |   ├── COCO2014-All-ver01.json
            |   └── COCO2014-Part-ver01.json
            |── LSPet_fit
            |   └── LSPet_ver01.json
            └── MPII_fit
                └── MPII_ver01.json
```

6.3.7 GTA-Human

```
@article{cai2021playing,
  title={Playing for 3D Human Recovery},
  author={Cai, Zhongang and Zhang, Mingyuan and Ren, Jiawei and Wei, Chen and Ren,␣
↪Daxuan and Li, Jiatong and Lin, Zhengyu and Zhao, Haiyu and Yi, Shuai and Yang, Lei␣
↪and others},
  journal={arXiv preprint arXiv:2110.07588},
  year={2021}
}
```

More details are coming soon!

6.3.8 Human3.6M

```
@article{h36m_pami,
  author = {Ionescu, Catalin and Papava, Dragos and Olaru, Vlad and Sminchisescu, ␣
↪Cristian},
  title = {Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing␣
↪in Natural Environments},
  journal = {IEEE Transactions on Pattern Analysis and Machine Intelligence},
  publisher = {IEEE Computer Society},
  volume = {36},
  number = {7},
  pages = {1325-1339},
  month = {jul},
  year = {2014}
}
```

For Human3.6M, please download from the official website and run the preprocessing script, which will extract pose annotations at downsampled framerate (10 FPS). The processed data should have the following structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── h36m
            ├── annot
            ├── S1
            |   ├── images
            |   |    |── S1_Directions_1.54138969
            |   |    |   ├── S1_Directions_1.54138969_00001.jpg
            |   |    |   ├── S1_Directions_1.54138969_00006.jpg
            |   |    |   └── ...
            |   |    └── ...
            |   ├── MyPoseFeatures
            |   |    |── D2Positions
            |   |    └── D3_Positions_Mono
            |   ├── MySegmentsMat
            |   |    └── ground_truth_bs
            |   └── Videos
            |        |── Directions 1.54138969.mp4
            |        |── Directions 1.55011271.mp4
            |        └── ...
            ├── S5
            ├── S6
            ├── S7
            ├── S8
            ├── S9
            ├── S11
            └── metadata.xml
```

To extract images from Human3.6M original videos, modify the `h36m_p1` config in DATASET_CONFIG:

```
h36m_p1=dict(
    type='H36mConverter',
    modes=['train', 'valid'],
    protocol=1,
    extract_img=True, # set to true to extract images from raw videos
    prefix='h36m'),
```

6.3.9 Human3.6M Mosh

For data preparation of Human3.6M for HMR, SPIN and PARE training, we use the MoShed data provided in HMR for training. However, due to license limitations, we are not allowed to redistribute the data. Even if you do not have access to these parameters, you can still generate the preprocessed h36m npz file without mosh parameters using our converter.

You will need to extract images from raw videos for training. Do note that preprocessing can take a long time if image extraction is required. To do so, modify the `h36m_p1` config in DATASET_CONFIG:

Config without mosh:

```
h36m_p1=dict(
    type='H36mConverter',
    modes=['train', 'valid'],
    protocol=1,
    extract_img=True,  # this is to specify you want to extract images from videos
    prefix='h36m'),
```

Config with mosh:

```
h36m_p1=dict(
    type='H36mConverter',
    modes=['train', 'valid'],
    protocol=1,
    extract_img=True,  # this is to specify you want to extract images from videos
    mosh_dir='data/datasets/h36m_mosh', # supply the directory to the mosh if available
    prefix='h36m'),
```

If you have MoShed data available, it should have the following structure:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── h36m_mosh
            ├── annot
            ├── S1
            |   ├── images
            |   |    ├── Directions 1_cam0_aligned.pkl
            |   |    ├── Directions 1_cam1_aligned.pkl
            |   |    └── ...
            ├── S5
            ├── S6
            ├── S7
            ├── S8
            ├── S9
            └── S11
```

6.3.10 HybrIK

```
@inproceedings{li2020hybrikg,
  author = {Li, Jiefeng and Xu, Chao and Chen, Zhicun and Bian, Siyuan and Yang, Lixin␣
↪and Lu, Cewu},
  title = {HybrIK: A Hybrid Analytical-Neural Inverse Kinematics Solution for 3D Human␣
↪Pose and Shape Estimation},
  booktitle={CVPR 2021},
  pages={3383--3393},
  year={2021},
  organization={IEEE}
}
```

For HybrIK, please download the parsed json annotation files and place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── hybrik_data
            ├── Sample_5_train_Human36M_smpl_leaf_twist_protocol_2.json
            ├── Sample_20_test_Human36M_smpl_protocol_2.json
            ├── 3DPW_test_new.json
            ├── annotation_mpi_inf_3dhp_train_v2.json
            └── annotation_mpi_inf_3dhp_test.json
```

To convert the preprocessed json files into npz files used for our pipeline, run the following preprocessing scripts:

- Human3.6M
- PW3D
- Mpi-Inf-3dhp
- COCO

6.3.11 LSP

```
@inproceedings{johnson2010clustered,
  title={Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation.},
  author={Johnson, Sam and Everingham, Mark},
  booktitle={bmvc},
  volume={2},
  number={4},
  pages={5},
  year={2010},
  organization={Citeseer}
}
```

For LSP, please download the high resolution version LSP dataset original. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── lsp
            ├── images
            |   ├── im0001.jpg
            |   ├── im0002.jpg
            |   └── ...
            └── joints.mat
```

6.3.12 LSPET

```
@inproceedings{johnson2011learning,
  title={Learning effective human pose estimation from inaccurate annotation},
  author={Johnson, Sam and Everingham, Mark},
  booktitle={CVPR 2011},
  pages={1465--1472},
  year={2011},
  organization={IEEE}
}
```

For LSPET, please download its high resolution form HR-LSPET. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── lspet
            ├── im00001.jpg
            ├── im00002.jpg
            ├── im00003.jpg
            ├── ...
            └── joints.mat
```

6.3.13 MPI-INF-3DHP

```
@inproceedings{mono-3dhp2017,
 author = {Mehta, Dushyant and Rhodin, Helge and Casas, Dan and Fua, Pascal and␣
↪Sotnychenko, Oleksandr and Xu, Weipeng and Theobalt, Christian},
 title = {Monocular 3D Human Pose Estimation In The Wild Using Improved CNN Supervision},
 booktitle = {3D Vision (3DV), 2017 Fifth International Conference on},
 url = {http://gvv.mpi-inf.mpg.de/3dhp_dataset},
 year = {2017},
 organization={IEEE},
 doi={10.1109/3dv.2017.00064},
}
```

You will need to extract images from raw videos for training. Do note that preprocessing can take a long time if image extraction is required. To do so, modify the `mpi_inf_3dhp` config in DATASET_CONFIG:

Config:

```
mpi_inf_3dhp=dict(
  type='MpiInf3dhpConverter',
  modes=['train', 'test'],
  extract_img=True),  # this is to specify you want to extract images from videos
```

For MPI-INF-3DHP, download and extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── mpi_inf_3dhp
            ├── mpi_inf_3dhp_test_set
            │   ├── TS1
            │   ├── TS2
            │   ├── TS3
            │   ├── TS4
            │   ├── TS5
            │   └── TS6
            ├── S1
            │   ├── Seq1
            │   └── Seq2
            ├── S2
            │   ├── Seq1
            │   └── Seq2
            ├── S3
            │   ├── Seq1
            │   └── Seq2
            ├── S4
            │   ├── Seq1
            │   └── Seq2
            ├── S5
```

(continues on next page)

(continued from previous page)

```
        ├── Seq1
        └── Seq2
    ├── S6
        ├── Seq1
        └── Seq2
    ├── S7
        ├── Seq1
        └── Seq2
    └── S8
        ├── Seq1
        └── Seq2
```

6.3.14 MPII

```
@inproceedings{andriluka14cvpr,
  author = {Mykhaylo Andriluka and Leonid Pishchulin and Peter Gehler and Schiele, Bernt}
↪,
  title = {2D Human Pose Estimation: New Benchmark and State of the Art Analysis},
  booktitle = {IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year = {2014},
  month = {June}
}
```

For MPII data, please download images from [MPII Human Pose Dataset](http://human-pose.mpi-inf.mpg.de/ and annotations from here. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── mpii
            |── train.h5
            └── images
                |── 000001163.jpg
                |── 000003072.jpg
                └── ...
```

6.3.15 PoseTrack18

```
@inproceedings{andriluka2018posetrack,
  title={Posetrack: A benchmark for human pose estimation and tracking},
  author={Andriluka, Mykhaylo and Iqbal, Umar and Insafutdinov, Eldar and Pishchulin,␣
↪Leonid and Milan, Anton and Gall, Juergen and Schiele, Bernt},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={5167--5176},
  year={2018}
}
```

For PoseTrack18 data, please download from PoseTrack18. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── posetrack
            ├── images
            │   ├── train
            │   │   ├── 000001_bonn_train
            │   │   │   ├── 000000.jpg
            │   │   │   ├── 000001.jpg
            │   │   │   └── ...
            │   │   └── ...
            │   ├── val
            │   │   ├── 000342_mpii_test
            │   │   │   ├── 000000.jpg
            │   │   │   ├── 000001.jpg
            │   │   │   └── ...
            │   │   └── ...
            │   └── test
            │       ├── 000001_mpiinew_test
            │       │   ├── 000000.jpg
            │       │   ├── 000001.jpg
            │       │   └── ...
            │       └── ...
            └── posetrack_data
                └── annotations
                    ├── train
                    │   ├── 000001_bonn_train.json
                    │   ├── 000002_bonn_train.json
                    │   └── ...
                    ├── val
                    │   ├── 000342_mpii_test.json
                    │   ├── 000522_mpii_test.json
                    │   └── ...
```

(continues on next page)

(continued from previous page)

```
                    └── test
                        ├── 000001_mpiinew_test.json
                        ├── 000002_mpiinew_test.json
                        └── ...
```

6.3.16 Penn Action

```
@inproceedings{zhang2013pennaction,
 title={From Actemes to Action: A Strongly-supervised Representation for Detailed Action␣
↪Understanding},
 author={Zhang, Weiyu and Zhu, Menglong and Derpanis, Konstantinos},
 booktitle={ICCV},
 year={2013}
}
```

For Penn Action data, please download from Penn Action. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── penn_action
            ├── frames
            │   ├── 0001
            │   │   ├── 000001.jpg
            │   │   ├── 000002.jpg
            │   │   └── ...
            │   └── ...
            └── labels
                ├── 0001.mat
                ├── 0002.mat
                └── ...
```

6.3.17 PW3D

```
@inproceedings{vonMarcard2018,
title = {Recovering Accurate 3D Human Pose in The Wild Using IMUs and a Moving Camera},
author = {von Marcard, Timo and Henschel, Roberto and Black, Michael and Rosenhahn, Bodo␣
↪and Pons-Moll, Gerard},
booktitle = {European Conference on Computer Vision (ECCV)},
year = {2018},
month = {sep}
}
```

For PW3D data, please download from PW3D Dataset. Extract them under `$MMHUMAN3D/data/datasets`, and make them look like this:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── pw3d
            |── imageFiles
            |   |    └── courtyard_arguing_00
            |   |       ├── image_00000.jpg
            |   |       ├── image_00001.jpg
            |   |       └── ...
            └── sequenceFiles
                ├── train
                |   ├── downtown_arguing_00.pkl
                |   └── ...
                ├── val
                |   ├── courtyard_arguing_00.pkl
                |   └── ...
                └── test
                    ├── courtyard_basketball_00.pkl
                    └── ...
```

6.3.18 SPIN

```
@inproceedings{kolotouros2019spin,
  author = {Kolotouros, Nikos and Pavlakos, Georgios and Black, Michael J and Daniilidis,
↪ Kostas},
  title = {Learning to Reconstruct 3D Human Pose and Shape via Model-fitting in the Loop}
↪,
  booktitle={ICCV},
  year={2019}
}
```

For SPIN, please download the preprocessed npz files and place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── spin_data
            ├── coco_2014_train.npz
            ├── hr-lspet_train.npz
```

(continues on next page)

(continued from previous page)

```
            ├── lsp_dataset_original_train.npz
            ├── mpi_inf_3dhp_train.npz
            └── mpii_train.npz
```

6.3.19 SURREAL

```
@inproceedings{varol17_surreal,
 title     = {Learning from Synthetic Humans},
 author    = {Varol, G{\"u}l and Romero, Javier and Martin, Xavier and Mahmood, Naureen␣
↪and Black, Michael J. and Laptev, Ivan and Schmid, Cordelia},
 booktitle = {CVPR},
 year      = {2017}
}
```

For SURREAL, please download the [dataset] (https://www.di.ens.fr/willow/research/surreal/data/) and place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── surreal
            ├── train
            |   ├── run0
            |   |   ├── 03_01
            |   |   |   ├── 03_01_c0001_depth.mat
            |   |   |   ├── 03_01_c0001_info.mat
            |   |   |   ├── 03_01_c0001_segm.mat
            |   |   |   ├── 03_01_c0001.mp4
            |   |   |   └── ...
            |   |   └── ...
            |   ├── run1
            |   └── run2
            ├── val
            |   ├── run0
            |   ├── run1
            |   └── run2
            └── test
                ├── run0
                ├── run1
                └── run2
```

6.3.20 VIBE

```
@inproceedings{VIBE,
  author    = {Muhammed Kocabas and
               Nikos Athanasiou and
               Michael J. Black},
  title     = {{VIBE}: Video Inference for Human Body Pose and Shape Estimation},
  booktitle = {CVPR},
  year      = {2020}
}
```

For VIBE, please download the preprocessed mpi_inf_3dhp and pw3d npz files from SPIN and pretrained frame feature extractor spin.pth. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    ├── checkpoints
    |   └── spin.pth
    └── datasets
        └── vibe_data
            ├── mpi_inf_3dhp_train.npz
            └── pw3d_test.npz
```

6.3.21 FreiHand

```
@inproceedings{zimmermann2019freihand,
  title={Freihand: A dataset for markerless capture of hand pose and shape from single␣
↪rgb images},
  author={Zimmermann, Christian and Ceylan, Duygu and Yang, Jimei and Russell, Bryan and␣
↪Argus, Max and Brox, Thomas},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision},
  pages={813--822},
  year={2019}
}
```

For FreiHand data, please download from FreiHand Dataset. Extract them under `$MMHUMAN3D/data/datasets`. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── FreiHand
```

(continues on next page)

(continued from previous page)

```
            |── evaluation
            |   └── rgb
            |       ├── 00000000.jpg
            |       ├── 00000001.jpg
            |       └── ...
            |── training
            |   └── rgb
            |       ├── 00000000.jpg
            |       ├── 00000001.jpg
            |       └── ...
            |── evaluation_K.json
            |── evaluation_mano.json
            |── evaluation_scale.json
            |── evaluation_verts.json
            |── evaluation_xyz.json
            |── training_K.json
            |── training_mano.json
            |── training_scale.json
            |── training_verts.json
            └── training_xyz.json
```

6.3.22 EHF

```
@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)},
  pages     = {10975--10985},
  year = {2019}
}
```

For EHF data, please download from EHF Dataset. Extract them under `$MMHUMAN3D/data/datasets`. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── EHF
            |── 01_2Djnt.json
            |── 01_2Djnt.png
            |── 01_align.ply
            |── 01_img.jpg
            |── 01_img.png
            |── 01_scan.obj
            └── ...
```

6.3.23 FFHQ

```
@inproceedings{karras2019style,
  title={A style-based generator architecture for generative adversarial networks},
  author={Karras, Tero and Laine, Samuli and Aila, Timo},
  booktitle={Proceedings of the IEEE/CVF conference on computer vision and pattern␣
↪recognition},
  pages={4401--4410},
  year={2019}
}
```

For FFHQ data, please download from FFHQ Dataset. We present ffhq_annotations.npz by running RingNet on FFHQ and then fitting to FAN 2D landmarks by flame-fitting. Extract them under `$MMHUMAN3D/data/datasets`. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── FFHQ
            |── ffhq_global_images_1024
            |       ├── 00000.png
            |       ├── 00001.png
            |       └── ...
            └── ffhq_annotations.npz
```

6.3.24 ExPose

```
@inproceedings{ExPose:2020,
  title = {Monocular Expressive Body Regression through Body-Driven Attention},
  author = {Choutas, Vasileios and Pavlakos, Georgios and Bolkart, Timo and Tzionas,␣
↪Dimitrios and Black, Michael J.},
  booktitle = {European Conference on Computer Vision (ECCV)},
  pages = {20--40},
  year = {2020},
  url = {https://expose.is.tue.mpg.de}
}
```

For ExPose data, please download from Curated Fits Dataset and SPIN IN SMPLX Dataset. Extract them under `$MMHUMAN3D/data/datasets`. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
```

(continues on next page)

(continued from previous page)

```
    └── datasets
        |── ExPose_curated_fits
        |   |── train.npz
        |   └── val.npz
        └── spin_in_smplx
            |── coco.npz
            |── lsp.npz
            |── lspet.npz
            └── mpii.npz
```

6.3.25 Stirling

```
@inproceedings{feng2018evaluation,
  title={Evaluation of dense 3D reconstruction from 2D face images in the wild},
  author={Feng, Zhen-Hua and Huber, Patrik and Kittler, Josef and Hancock, Peter and Wu,␣
↪Xiao-Jun and Zhao, Qijun and Koppen, Paul and R{\"a}tsch, Matthias},
  booktitle={2018 13th IEEE International Conference on Automatic Face \& Gesture␣
↪Recognition (FG 2018)},
  pages={780--786},
  year={2018},
  organization={IEEE}
}
```

For Stirling ESRC3D Face data, please download from Stirling ESRC3D Face Dataset. Extract them under `$MMHUMAN3D/data/datasets`. Place them in the folder structure below:

```
mmhuman3d
├── mmhuman3d
├── docs
├── tests
├── tools
├── configs
└── data
    └── datasets
        └── stirling
            |── annotations
            |   ├── F_3D_N
            |   |   ├── F1001_N.lnd
            |   |   ├── F1002_N.lnd
            |   |   └── ...
            |   └── M_3D_N
            |       ├── M1001_N.lnd
            |       ├── M1002_N.lnd
            |       └── ...
            |── F_3D_N
            |   ├── F1001_N.obj
            |   ├── F1002_N.obj
            |   └── ...
            |── M_3D_N
            |   ├── M1001_N.obj
```

(continues on next page)

(continued from previous page)

```
|   ├── M1002_N.obj
|   └── ...
└── Subset_2D_FG2018
    ├── HQ
    |   ├── F1001_001.jpg
    |   ├── F1001_002.jpg
    |   └── ...
    └── LQ
        ├── F1001_008.jpg
        ├── F1001_009.jpg
        └── ...
```

CHAPTER

SEVEN

KEYPOINTS CONVENTION

7.1 Overview

Our convention tries to consolidate the different keypoints definition across various commonly used datasets. Due to differences in data-labelling procedures, keypoints across datasets with the same name might not map to semantically similar locations on the human body. Conversely, keypoints with different names might correspond to the same location on the human body. To unify the different keypoints correspondences across datasets, we adopted the `human_data` convention as the base convention for converting and storing our keypoints.

7.2 How to use

7.2.1 Converting between conventions

Keypoints can be converted between different conventions easily using the `convert_kps` function.

To convert a `human_data` keypoints to `coco` convention, specify the source and destination convention for conversion.

```
from mmhuman3d.core.conventions.keypoints_mapping import convert_kps

keypoints_human_data = np.zeros((100, 190, 3))
keypoints_coco, mask = convert_kps(keypoints_human_data, src='human_data', dst='coco')
assert mask.all()==1
```

The output `mask` should be all ones if the `dst` convention is the subset of the `src` convention. You can use the `mask` as the confidence of the keypoints since those keypoints with no correspondence are set to a default value with 0 confidence.

7.2.2 Converting with confidence

If you have confidential information of your keypoints, you can use an original mask to mark it, then the information will be updated into the returned mask. E.g., you want to convert a `smpl` keypoints to `coco` keypoints, and you know its `left_shoulder` is occluded. You want to carry forward this information during the converting. So you can set an original_mask and convert it to `coco` by doing:

```
import numpy as np
from mmhuman3d.core.conventions.keypoints_mapping import KEYPOINTS_FACTORY, convert_kps

keypoints = np.zeros((1, len(KEYPOINTS_FACTORY['smpl']), 3))
```

(continues on next page)

(continued from previous page)

```
confidence = np.ones((len(KEYPOINTS_FACTORY['smpl'])))

# assume that 'left_shoulder' point is invalid.
confidence[KEYPOINTS_FACTORY['smpl'].index('left_shoulder')] = 0

_, conf_coco = convert_kps(
    keypoints=keypoints, confidence=confidence, src='smpl', dst='coco')
_, conf_coco_full = convert_kps(
    keypoints=keypoints, src='smpl', dst='coco')

assert conf_coco[KEYPOINTS_FACTORY['coco'].index('left_shoulder')] == 0
conf_coco[KEYPOINTS_FACTORY['coco'].index('left_shoulder')] = 1
assert (conf_coco == conf_coco_full).all()
```

Our mask represents valid information, its dtype is uint8, while keypoint confidence usually ranges from 0 to 1. E.g., you want to convert a `smpl` keypoints to `coco` keypoints, and you know its `left_shoulder` is occluded. You want to carry forward this information during the converting. So you can set an original_mask and convert it to `coco` by doing:

```
confidence = np.ones((len(KEYPOINTS_FACTORY['smpl'])))
confidence[KEYPOINTS_FACTORY['smpl'].index('left_shoulder')] = 0.5
kp_smpl = np.concatenate([kp_smpl, confidence], -1)
kp_smpl_converted, mask = convert_kps(kp_smpl, src='smpl', dst='coco')
new_confidence =  kp_smpl_converted[..., 2:]
assert new_confidence[KEYPOINTS_FACTORY['smpl'].index('left_shoulder')] == 0.5
```

7.3 Supported Conventions

These are the supported conventions:

- AGORA
- COCO
- COCO-WHOLEBODY
- CrowdPose
- GTA-Human
- Human3.6M
- human_data
- HybrIK
- LSP
- MPI-INF-3DHP
- MPII
- openpose
- PennAction
- PoseTrack18
- PW3D

- SMPL
- SMPL-X

7.3.1 HUMANDATA

The first 144 keypoints in HumanData correspond to that in `SMPL-X`. Keypoints with suffix `_extra` refer to those obtained from Jregressor_extra. Keypoints with suffix `_openpose` refer to those obtained from `OpenPose` predictions.

There are several keypoints from `MPI-INF-3DHP`, `Human3.6M` and `Posetrack` that has the same name but were semantically different from keypoints in `SMPL-X`. As such, we added an extra suffix to differentiate those keypoints i.e. `head_h36m`.

7.3.2 AGORA

```
@inproceedings{Patel:CVPR:2021,
  title = {{AGORA}: Avatars in Geography Optimized for Regression Analysis},
  author = {Patel, Priyanka and Huang, Chun-Hao P. and Tesch, Joachim and Hoffmann,␣
↪David T. and Tripathi, Shashank and Black, Michael J.},
  booktitle = {Proceedings IEEE/CVF Conf.~on Computer Vision and Pattern Recognition (
↪{CVPR})},
  month = jun,
  year = {2021},
  month_numeric = {6}
}
```

7.3.3 COCO

```
@inproceedings{lin2014microsoft,
  title={Microsoft coco: Common objects in context},
  author={Lin, Tsung-Yi and Maire, Michael and Belongie, Serge and Hays, James and␣
↪Perona, Pietro and Ramanan, Deva and Doll{\'a}r, Piotr and Zitnick, C Lawrence},
  booktitle={European conference on computer vision},
  pages={740--755},
  year={2014},
  organization={Springer}
}
```

7.3.4 COCO-WHOLEBODY

```
@inproceedings{jin2020whole,
  title={Whole-Body Human Pose Estimation in the Wild},
  author={Jin, Sheng and Xu, Lumin and Xu, Jin and Wang, Can and Liu, Wentao and Qian,␣
↪Chen and Ouyang, Wanli and Luo, Ping},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  year={2020}
}
```

7.3.5 CrowdPose

```
@article{li2018crowdpose,
  title={CrowdPose: Efficient Crowded Scenes Pose Estimation and A New Benchmark},
  author={Li, Jiefeng and Wang, Can and Zhu, Hao and Mao, Yihuan and Fang, Hao-Shu and␣
↪Lu, Cewu},
  journal={Proceedings IEEE/CVF Conf.~on Computer Vision and Pattern Recognition ({CVPR}
↪)},
  year={2019}
}
```

7.3.6 Human3.6M

```
@article{h36m_pami,
  author = {Ionescu, Catalin and Papava, Dragos and Olaru, Vlad and Sminchisescu, ␣
↪Cristian},
  title = {Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing␣
↪in Natural Environments},
  journal = {IEEE Transactions on Pattern Analysis and Machine Intelligence},
  publisher = {IEEE Computer Society},
  volume = {36},
  number = {7},
  pages = {1325-1339},
  month = {jul},
  year = {2014}
}
```

7.3.7 GTA-Human

```
@article{cai2021playing,
  title={Playing for 3D Human Recovery},
  author={Cai, Zhongang and Zhang, Mingyuan and Ren, Jiawei and Wei, Chen and Ren,␣
↪Daxuan and Li, Jiatong and Lin, Zhengyu and Zhao, Haiyu and Yi, Shuai and Yang, Lei␣
↪and others},
  journal={arXiv preprint arXiv:2110.07588},
  year={2021}
}
```

7.3.8 HybrIK

```
@inproceedings{li2020hybrikg,
  author = {Li, Jiefeng and Xu, Chao and Chen, Zhicun and Bian, Siyuan and Yang, Lixin␣
↪and Lu, Cewu},
  title = {HybrIK: A Hybrid Analytical-Neural Inverse Kinematics Solution for 3D Human␣
↪Pose and Shape Estimation},
  booktitle={CVPR 2021},
  pages={3383--3393},
  year={2021},
```

(continues on next page)

(continued from previous page)

```
  organization={IEEE}
}
```

7.3.9 LSP

```
@inproceedings{johnson2010clustered,
  title={Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation.},
  author={Johnson, Sam and Everingham, Mark},
  booktitle={bmvc},
  volume={2},
  number={4},
  pages={5},
  year={2010},
  organization={Citeseer}
}
```

7.3.10 MPI-INF-3DHP

```
@inproceedings{mono-3dhp2017,
 author = {Mehta, Dushyant and Rhodin, Helge and Casas, Dan and Fua, Pascal and␣
↪Sotnychenko, Oleksandr and Xu, Weipeng and Theobalt, Christian},
 title = {Monocular 3D Human Pose Estimation In The Wild Using Improved CNN Supervision},
 booktitle = {3D Vision (3DV), 2017 Fifth International Conference on},
 url = {http://gvv.mpi-inf.mpg.de/3dhp_dataset},
 year = {2017},
 organization={IEEE},
 doi={10.1109/3dv.2017.00064},
}
```

7.3.11 MPII

```
@inproceedings{andriluka14cvpr,
  author = {Mykhaylo Andriluka and Leonid Pishchulin and Peter Gehler and Schiele, Bernt}
↪,
  title = {2D Human Pose Estimation: New Benchmark and State of the Art Analysis},
  booktitle = {IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year = {2014},
  month = {June}
}
```

7.3.12 PoseTrack18

```
@inproceedings{andriluka2018posetrack,
  title={Posetrack: A benchmark for human pose estimation and tracking},
  author={Andriluka, Mykhaylo and Iqbal, Umar and Insafutdinov, Eldar and Pishchulin,␣
↪Leonid and Milan, Anton and Gall, Juergen and Schiele, Bernt},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={5167--5176},
  year={2018}
}
```

7.3.13 OpenPose

```
@article{8765346,
  author = {Z. {Cao} and G. {Hidalgo Martinez} and T. {Simon} and S. {Wei} and Y. A.
↪{Sheikh}},
  journal = {IEEE Transactions on Pattern Analysis and Machine Intelligence},
  title = {OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields}
↪,
  year = {2019}
}
```

7.3.14 PennAction

```
@inproceedings{zhang2013,
  title={From Actemes to Action: A Strongly-supervised Representation for Detailed␣
↪Action Understanding},
  author={Zhang, Weiyu and Zhu, Menglong and Derpanis, Konstantinos},
  booktitle={Proceedings of the International Conference on Computer Vision},
  year={2013}
}
```

7.3.15 SMPL

```
@article{SMPL:2015,
      author = {Loper, Matthew and Mahmood, Naureen and Romero, Javier and Pons-Moll,␣
↪Gerard and Black, Michael J.},
      title = {{SMPL}: A Skinned Multi-Person Linear Model},
      journal = {ACM Trans. Graphics (Proc. SIGGRAPH Asia)},
      month = oct,
      number = {6},
      pages = {248:1--248:16},
      publisher = {ACM},
      volume = {34},
      year = {2015}
    }
```

7.3.16 SMPL-X

```
@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)},
  pages     = {10975--10985},
  year = {2019}
}
```

7.3.17 Customizing keypoint convention

Please refer to *customize_keypoints_convention*.

CHAPTER

EIGHT

CUSTOMIZE KEYPOINTS CONVENTION

8.1 Overview

If your dataset use an unsupported convention, a new convention can be added following this documentation.

These are the conventions that our project currently support:

- agora
- coco
- coco_wholebody
- crowdpose
- h36m
- human_data
- hybrik
- lsp
- mpi_inf_3dhp
- mpii
- openpose
- penn_action
- posetrack
- pw3d
- smpl
- smplx

1. Create a new convention

Please follow `mmhuman3d/core/conventions/keypoints_mapping/human_data.py` to create a file named NEW_CONVENTION.py. In this file, `NEW_KEYPOINTS` is a list containing keypoints naming and order specific to the new convention.

For instance, if we want to create a new convention for AGORA dataset, `agora.py` would contain:

```
AGORA_KEYPOINTS = [
  'pelvis',
  'left_hip',
```

(continues on next page)

(continued from previous page)

```
  'right_hip'
  ...
]
```

2. Search for keypoint names in human_data.

In this project, keypoints that share the same naming across datasets should have the exact same semantic definition in the human body. human_data convention has already consolidated the different keypoints naming and correspondences across our supported datasets.

For each keypoint in NEW_KEYPOINTS, we have to check (1) if the keypoint name exists in mmhuman3d/core/conventions/keypoints_mapping/human_data.py and (2) if the keypoint has a correspondence i.e. maps to the same location as the ones defined in human_data.

If both conditions are met, retain the keypoint name in NEW_CONVENTION.py.

3. Search for keypoints correspondence in human_data.

If a keypoint in NEW_KEYPOINTS shares the same correspondence as a keypoint that is named differently in the human_data convention i.e. head in NEW_CONVENTION.py maps to head_extra in human_data, rename the keypoint to follow the new one in our convention i.e. head-> head_extra.

4. Add a new keypoint to human_data

If the keypoint has no correspondence nor share an existing name to the ones defined in human_data, please list it as well but add a prefix to the original name to differentiate it from those with existing correspondences i.e. spine_3dhp

We may expand human_data to the new keypoint if necessary. However, this can only be done after checking that the new keypoint do not have a correspondence and there is no conflicting names.

5. Initialise the new set of keypoint convention

Add import for NEW_CONVENTION.py in mmhuman3d/core/conventions/keypoints_mapping/__init__.py, and add the identifier to dict KEYPOINTS_FACTORY.

For instance, if our new convention is agora:

```
# add import
from mmhuman3d.core.conventions.keypoints_mapping import (
    agora,
    ...
)

# add to factory
KEYPOINTS_FACTORY = {
    'agora': agora.AGORA_KEYPOINTS,
    ...
}
```

6. Using keypoints convention for keypoints mapping

To convert keypoints from any existing convention to your newly defined convention (or vice versa), you can use the convert_kps function mmhuman3d/core/conventions/keypoints_mapping/__init__.py, which produce a mask containing 0 or 1 indicating if the corresponding point should be filtered or retained.

To convert from coco to new convention:

```
new_kps, mask = convert_kps(smplx_keypoints, src='coco', dst='NEW_CONVENTION')
```

To convert from new convention to human_data:

```
new_kps, mask = convert_kps(smplx_keypoints, src='NEW_CONVENTION', dst='human_data')
```

CHAPTER

NINE

CAMERAS

9.1 Camera Initialization

We follow `Pytorch3D` cameras. The camera extrinsic matrix is defined as the camera to world transformation, and uses right matrix multiplication, whereas the intrinsic matrix uses left matrix multiplication. Nevertheless, our interface provides `opencv` convention that defines the camera the same way as an `OpenCV` camera, would be helpful if you are more familiar with that.

- **Slice cameras:**

  In mmhuman3d, the recommended way to initialize a camera is by passing `K`, `R`, `T` matrix directly. You can slice the cameras by index. You can also concat the cameras in batch dim.

```
from mmhuman3d.core.cameras import PerspectiveCameras
import torch
K = torch.eye(4, 4)[None]
R = torch.eye(3, 3)[None]
T = torch.zeros(100, 3)
# Batch of K, R, T should all be the same or some of them could be 1. The final␣
↪batch size will be the biggest one.
cam = PerspectiveCameras(K=K, R=R, T=T)
assert cam.R.shape == (100, 3, 3)
assert cam.K.shape == (100, 4, 4)
assert cam.T.shape == (100, 3)
assert (cam[:10].K == cam.K[:10]).all()
```

- **Build cameras:**

  Wrapped by mmcv.Registry. In mmhuman3d, the recommended way to initialize a camera is by passing `K`, `R`, `T` matrix directly, but you also have the options to pass `focal_length` and `principle_point` as the input.

  Take the usually used `PerspectiveCameras` as examples. If `K`, `R`, `T` are not specified, the `K` will use default `K` by `compute_default_projection_matrix` with default `focal_length` and `principal_point` and `R` will be identical matrix, `T` will be zeros. You can also specify by overwriting the parameters for `compute_default_projection_matrix`.

```
from mmhuman3d.core.cameras import build_cameras

# Initialize a perspective camera with given K, R, T matrix.
# It is recommended that the batches of K, R, T either the same or be 1.
K = torch.eye(4, 4)[None]
R = torch.eye(3, 3)[None]
T = torch.zeros(10, 3)
```

(continues on next page)

(continued from previous page)

```
height, width = 1000
cam1 = build_cameras(
    dict(
        type='PerspectiveCameras',
        K=K,
        R=R,
        T=T,
        in_ndc=True,
        image_size=(height, width),
        convention='opencv',
        ))

# This is the same as:
cam2 = PerspectiveCameras(
        K=K,
        R=R,
        T=T,
        in_ndc=True,
        image_size=1000, # single number represents square images.
        convention='opencv',
        )
assert cam1.K.shape == cam2.K.shape == (10, 4, 4)
assert cam1.R.shape == cam2.R.shape == (10, 3, 3)
assert cam1.T.shape == cam2.T.shape == (10, 3)

# Initialize a perspective camera with specific `image_size`, `principal_points`,␣
↪`focal_length`.
# `in_ndc = False` means the intrinsic matrix `K` defined in screen space. The `focal_
↪length` and `principal_point` in `K` is defined in scale of pixels. This `principal_
↪points` is (500, 500) pixels and `focal_length` is 1000 pixels.
cam = build_cameras(
    dict(
        type='PerspectiveCameras',
        in_ndc=False,
        image_size=(1000, 1000),
        principal_points=(500, 500),
        focal_length=1000,
        convention='opencv',
        ))

assert (cam.K[0] == torch.Tensor([[1000., 0.,  500.,    0.],
                                  [0., 1000.,  500.,    0.],
                                  [0.,    0.,    0.,    1.],
                                  [0.,    0.,    1.,    0.]]).view(4, 4)).all()

# Initialize a weakperspective camera with given K, R, T. weakperspective camera␣
↪support `in_ndc = True` only.
cam = build_cameras(
    dict(
        type='WeakPerspectiveCameras',
        K=K,
```

(continues on next page)

(continued from previous page)

```
        R=R,
        T=T,
        image_size=(1000, 1000)
        ))

# If no `K`, `R`, `T` information provided
# Initialize a `in_ndc` perspective camera with default matrix.
cam = build_cameras(
    dict(
        type='PerspectiveCameras',
        in_ndc=True,
        image_size=(1000, 1000),
        ))
# Then convert it to screen. This operation requires `image_size`.
cam.to_screen_()
```

9.2 Camera Projection Matrixs

- **Perspective:**

  format of intrinsic matrix: fx, fy is focal_length, px, py is principal_point.

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

  Detailed information refer to Pytorch3D.

- **WeakPerspective:**

  format of intrinsic matrix:

```
K = [
        [sx*r,   0,    0,   tx*sx*r],
        [0,     sy,    0,   ty*sy],
        [0,     0,     1,       0],
        [0,     0,     0,       1],
    ]
```

  `WeakPerspectiveCameras` is orthographics indeed, mainly for SMPL(x) projection. Detailed information refer to mmhuman3d cameras. This can be converted from SMPL predicted camera parameter by:

```
from mmhuman3d.core.cameras import WeakPerspectiveCameras
K = WeakPerspectiveCameras.convert_orig_cam_to_matrix(orig_cam)
```

  The pred_cam is array/tensor of shape (frame, 4) consists of [scale_x, scale_y, transl_x, transl_y]. See in VIBE.

- **FoVPerspective:**

```
format of intrinsic matrix:
K = [
        [s1,   0,   w1,   0],
        [0,   s2,   h1,   0],
        [0,    0,   f1,  f2],
        [0,    0,    1,   0],
    ]
```

s1, s2, w1, h1, f1, f2 are defined by FoV parameters (`fov`, `znear`, `zfar`, etc.), detailed information refer to Pytorch3D.

- **Orthographics:**

  format of intrinsic matrix:

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

  Detailed information refer to Pytorch3D.

- **FoVOrthographics:**

```
K = [
        [scale_x,        0,         0,  -mid_x],
        [0,        scale_y,         0,  -mix_y],
        [0,              0,  -scale_z,  -mid_z],
        [0,              0,         0,       1],
]
```

  scale_x, scale_y, scale_z, mid_x, mid_y, mid_z are defined by FoV parameters(`min_x`, `min_y`, `max_x`, `max_y`, `znear`, `zfar`, etc.), related information refer to Pytorch3D.

9.3 Camera Conventions

- **Convert between different cameras:**

  We name intrinsic matrix as `K`, rotation matrix as `R` and translation matrix as `T`. Different camera conventions have different axis directions, and some use left matrix multiplication and some use right matrix multiplication. Intrinsic and extrinsic matrix should be of the same multiplication convention, but some conventions like `Pytorch3D` uses right matrix multiplication in computation procedure but passes left matrix multiplication `K` when initializing the cameras(mainly for better understanding). Conversion between `NDC` (normalized device coordinate) and `screen` also influence the intrinsic matrix, this is independent of camera conventions but should also be included. If you want to use an existing convention, choose in `['opengl', 'opencv', 'pytorch3d', 'pyrender', 'open3d']`. E.g., you want to convert your opencv calibrated camera to Pytorch3D NDC defined camera for rendering, you can do:

```
from mmhuman3d.core.conventions.cameras import convert_cameras
import torch
```

(continues on next page)

(continued from previous page)

```
K = torch.eye(4, 4)[None]
R = torch.eye(3, 3)[None]
T = torch.zeros(10, 3)
height, width = 1080, 1920
K, R, T = convert_cameras(
    K=K,
    R=R,
    T=T,
    in_ndc_src=False,
    in_ndc_dst=True,
    resolution_src=(height, width),
    convention_src='opencv',
    convention_dst='pytorch3d')
```

Input K could be None, or `array/tensor` of shape (batch_size, 3, 3) or (batch_size, 4, 4). Input R could be None, or `array/tensor` of shape (batch_size, 3, 3). Input T could be None, or `array/tensor` of shape (batch_size, 3). If the original `K` is `None`, it will remain `None`. If the original `R` is `None`, it will be set as identity matrix. If the original `T` is `None`, it will be set as zeros matrix. Please refer to Pytorch3D for more information about cameras in `NDC` and in `screen` space..

- **Define your new camera convention:**

  If want to use a new convention, define your convention in CAMERA_CONVENTION_FACTORY by the order of right to, up to, and off screen. E.g., the first one is pyrender and its convention should be '+x+y+z'. '+' could be ignored. The second one is opencv and its convention should be '+x-y-z'. The third one is Pytorch3D and its convention should be '-xyz'.

```
OpenGL(PyRender)        OpenCV              Pytorch3D
    y                     z                     y
    |                    /                      |
    |                   /                       |
    |_______x         /________x     x________  |
    /                 |                       /
   /                  |                      /
z /                 y |                   z /
```

9.4 Some Conversion Functions

Convert functions are also defined in conventions.cameras.

- **NDC & screen:**

```
from mmhuman3d.core.conventions.cameras import (convert_ndc_to_screen,
                                                convert_screen_to_ndc)

K = convert_ndc_to_screen(K, resolution=(1080, 1920), is_perspective=True)
K = convert_screen_to_ndc(K, resolution=(1080, 1920), is_perspective=True)
```

- **3x3 & 4x4 intrinsic matrix**

```
from mmhuman3d.core.conventions.cameras import (convert_K_3x3_to_4x4,
                                                convert_K_4x4_to_3x3)
```

(continues on next page)

(continued from previous page)

```
K = convert_K_3x3_to_4x4(K, is_perspective=True)
K = convert_K_4x4_to_3x3(K, is_perspective=True)
```

- **world & view:**

  Convert between world & view coordinates.

```
from mmhuman3d.core.conventions.cameras import convert_world_view
R, T = convert_world_view(R, T)
```

- **weakperspective & perspective:**

  Convert between weakperspective & perspective. zmean is needed. WeakperspectiveCameras is in_ndc, so you should pass resolution if perspective not in ndc.

```
from mmhuman3d.core.conventions.cameras import (
    convert_perspective_to_weakperspective,
    convert_weakperspective_to_perspective)

K = convert_perspective_to_weakperspective(
    K, zmean, in_ndc=False, resolution, convention='opencv')
K = convert_weakperspective_to_perspective(
    K, zmean, in_ndc=False, resolution, convention='pytorch3d')
```

9.5 Some Compute Functions

- **Project 3D coordinates to screen:**

```
points_xydepth = cameras.transform_points_screen(points)
points_xy = points_xydepth[..., :2]
```

- **Compute depth of points:**

  You can simply convert points to the view coordinates and get the z value as depth. Example could be found in DepthRenderer.

```
points_depth = cameras.compute_depth_of_points(points)
```

- **Compute normal of meshes:**

  Use `Pytorch3D` to compute normal of meshes. Example could be found in NormalRenderer.

```
normals = cameras.compute_normal_of_meshes(meshes)
```

- **Get camera plane normal:**

  Get the normalized normal tensor which points out of the camera plane from camera center.

```
normals = cameras.get_camera_plane_normals()
```

CHAPTER

TEN

VISUALIZE KEYPOINTS

10.1 Visualize 2d keypoints

- **simple example for visualize 2d keypoints:**

  You have 2d coco_wholebody keypoints of shape(10, 133, 2).

```
from mmhuman3d.core.visualization.visualize_keypoints2d import visualize_kp2d

visualize_kp2d(
    kp2d_coco_wholebody,
    data_source='coco_wholebody',
    output_path='some_video.mp4',
    resolution=(1024, 1024))
```

  Then a 1024x1024 sized video with 10 frames would be save as 'some_video.mp4'

- **data_source and mask:**

  If your keypoints have some nonsense points, you should provide the mask. `data_source` is mainly used to search the limb connections and palettes. You should specify the data_source if you dataset is in convention. E.g., convert coco_wholebody keypoints to the convention of smpl and visualize it:

```
from mmhuman3d.core.conventions.keypoints_mapping import convert_kps
from mmhuman3d.core.visualization.visualize_keypoints2d import visualize_kp2d

kp2d_smpl, mask = convert_kps(kp2d_coco_wholebody, src='coco_wholebody', dst='smpl')
visualize_kp2d(
    kp2d_smpl,
    mask=mask,
    output_path='some_video.mp4',
    resolution=(1024, 1024))
```

  mask is `None` by default. This is the same as all ones mask, then no keypoints will be excluded. Ignore it when you are sure that all the keypoints are valid.

- **whether plot on backgrounds:**

  Maybe you want to use numpy input backgrounds.

  E.g., you want to visualize you coco_wholebody kp2d as smpl convention. You have 2d coco_wholebody keypoints of shape(10, 133, 2).

```
from mmhuman3d.core.conventions.keypoints_mapping import convert_kps
from mmhuman3d.core.visualization.visualize_keypoints2d import visualize_kp2d

background = np.random.randint(low=0, high=255, shape=(10, 1024, 1024, 4))
# multi_person, shape is (num_person, num_joints, 2)
out_image = visualize_kp2d(
    kp2d=kp2d, image_array=background, data_source='coco_wholebody', return_
↪array=True)
```

This is just an example, you can use this function flexibly.

If want to plot keypoints on frame files, you could provide `frame_list`(list of image path). **Be aware that the order of the frame will be sorted by name.** or `origin_frames`(mp4 path or image folder path), **Be aware that you should provide the correct `img_format` for `ffmpeg` to read the images.**.

```
frame_list = ['im1.png', 'im2.png', ...]
visualize_kp2d(
    kp2d_coco_wholebody,
    data_source='coco_wholebody',
    output_path='some_video.mp4',
    resolution=(1024, 1024),
    frame_list=frame_list)

origin_frames = 'some_folder'
visualize_kp2d(
    kp2d_coco_wholebody,
    data_source='coco_wholebody',
    output_path='some_video.mp4',
    resolution=(1024, 1024),
    origin_frames=origin_frames)

origin_frames = 'some.mp4'
array = visualize_kp2d(
    kp2d_coco_wholebody,
    data_source='coco_wholebody',
    output_path='some_video.mp4',
    resolution=(1024, 1024),
    return_array=True,
    origin_frames=origin_frames)
```

The superiorty of background images: `frame_list`

- **output a video or frames:**

  If `output_path` is a folder, this function will output frames. If `output_path` is a '.mp4' path, this function will output a video. `output_path` could be set as `None` when `return_array` is True. The function will return an array of shape (frame, width, height, 3).

- **whether plot origin file name on images:**

  Specify `with_file_name=True` then origin frame name will be plotted on the image.

- **dataset not in existing convention or want to visualize some specific limbs:**

  You should provide limbs like `limbs=[[0, 1], ..., [10, 11]]` if you dataset is not in convention.

- **other parameters:**

  Easy to understand, please read the doc strings in the function.

10.2 Visualize 3d keypoints

- **simple example for visualize single person:**

  You have kp3d in smplx convention of shape (num_frame, 144, 3).

```
visualize_kp3d(kp3d=kp3d, data_source='smplx', output_path='some_video.mp4')
```

  The result video would have one person dancing, each body part has its own color.

- **simple example for visualize multi person:**

  You have kp3d_1 and kp3d_2 which are both in smplx convention of shape (num_frame, 144, 3).

```
kp3d = np.concatenate([kp3d_1[:, np.newaxis], kp3d_2[:, np.newaxis]], axis=1)
# kp3d.shape is now (num_frame, num_person, 144, 3)
visualize_kp3d(kp3d=kp3d, data_source='smplx', output_path='some_video.mp4')
```

  The result video would have two person dancing, each in a pure color, and the there will be a color legend describing the index of each person.

- **data_source and mask:**

  The same as visualize_kp2d

- **dataset not in existing convention or want to visualize some specific limbs:**

  The same as visualize_kp2d

- **output:** If `output_path` is a folder, this function will output frames. If `output_path` is a '.mp4' path, this function will output a video. `output_path` could be set as `None` when `return_array` is True. The function will return an array of shape (frame, width, height, 3).

- **other parameters:**

  Easy to understand, please read the doc strings in the function.

10.3 About ffmpeg_utils

- In ffmpeg_utils , each function has abundant doc strings, and the semantically defined function names could be easily understood.

- **read files:**

  images_to_array, video_to_array

- **write files:**

  array_to_images, array_to_video

- **convert formats:**

  gif_to_images, gif_to_video, video_to_images, video_to_gif, images_to_gif, images_to_video

- **temporally crop/concat:**

  slice_video, temporal_concat_video

- **spatially crop/concat:**

  crop_video, spatial_concat_video

- **compress:**

  compress_gif, compress_video

CHAPTER

ELEVEN

VISUALIZE SMPL MESH

- **fast visualize smpl(x) pose without background images:**

  You have smpl pose tensor or array shape of which is (frame, 72)

```
from mmhuman3d.core.visualization.visualize_smpl import visualize_smpl_pose
body_model_config = dict(
    type='smpl', model_path=model_path)
visualize_smpl_pose(
    poses=poses,
    output_path='smpl.mp4',
    resolution=(1024, 1024))
```

  Or you have smplx pose tensor or array shape of which is (frame, 165)

```
body_model_config = dict(
    type='smplx', model_path=model_path)
visualize_smpl_pose(
    poses=poses,
    body_model_config=body_model_config,
    output_path='smplx.mp4',
    resolution=(1024, 1024))
```

  You could also feed dict tensor of smplx definitions. You could check that in visualize_smpl or original smplx.

- **visualize T-pose:** If you want to visualize a T-pose smpl or your poses do not have global_orient, you can do:

```
import torch
from mmhuman3d.core.visualization.visualize_smpl import visualize_T_pose
body_model_config = dict(
    type='smpl', model_path=model_path)
visualize_T_pose(
    num_frames=100,
    body_model_config=body_model_config,
    output_path='smpl_tpose.mp4',
    orbit_speed=(1, 0.5),
    resolution=(1024, 1024))
```

- **visualize smpl with predicted VIBE camera:** You have poses (numpy/tensor) of shape (frame, 72), betas of shape (frame, 10), pred_cam of shape (10, 4). E.g., we use vibe sample_video.mp4 as an example.

```
import pickle
from mmhuman3d.core.visualization.visualize_smpl import visualize_smpl_vibe
```

(continues on next page)

(continued from previous page)

```
with open('vibe_output.pkl', 'rb') as f:
    d = pickle.load(f, encoding='latin1')
poses = d[1]['pose']
orig_cam = d[1]['orig_cam']
pred_cam = d[1]['pred_cam']
bbox = d[1]['bboxes']
gender = 'female'

# pass pred_cam & bbox
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_vibe(
    poses=poses,
    betas=betas,
    body_model_config=body_model_config,
    pred_cam=pred_cam,
    bbox=bbox,
    output_path='vibe_demo.mp4',
    origin_frames='sample_video.mp4',
    resolution=(1024, 1024))

# or pass orig_cam
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_vibe(
    poses=poses,
    betas=betas,
    body_model_config=body_model_config,
    orig_cam=orig_cam,
    output_path='vibe_demo.mp4',
    origin_frames='sample_video.mp4',
    resolution=(1024, 1024))
```

- **visualize smpl with predicted HMR/SPIN camera:** You have poses (numpy/tensor) of shape (frame, 72), betas of shape (frame, 10), cam_translation of shape (10, 4). E.g., we use vibe sample_video.mp4 as an example.

```
import pickle
from mmhuman3d.core.visualization.visualize_smpl import visualize_smpl_hmr
gender = 'female'
focal_length = 5000
det_width = 224
det_height = 224

# you can pass smpl poses & betas & gender
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_hmr(
    poses=poses,
    betas=betas,
    bbox=bbox,
    body_model_config=body_model_config,
```

(continues on next page)

(continued from previous page)

```
    focal_length=focal_length,
    det_width=det_width,
    det_height=det_height,
    T=cam_translation,
    output_path='hmr_demo.mp4',
    origin_frames=image_folder,
    resolution=(1024, 1024))

# or you can pass verts
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_hmr(
    verts=verts,
    bbox=bbox,
    focal_length=focal_length,
    body_model_config=body_model_config,
    det_width=det_width,
    det_height=det_height,
    T=cam_translation,
    output_path='hmr_demo.mp4',
    origin_frames=image_folder,
    resolution=(1024, 1024))

# you can also pass kp2d in replace of bbox.
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_hmr(
    verts=verts,
    body_model_config=body_model_config,
    kp2d=kp2d,
    focal_length=focal_length,
    det_width=det_width,
    det_height=det_height,
    T=cam_translation,
    output_path='hmr_demo.mp4',
    origin_frames=image_folder,
    resolution=(1024, 1024))
```

- **visualize smpl with opencv camera:** You should pass the opencv defined intrinsic matrix K and extrinsic matrix R, T.

```
from mmhuman3d.core.visualization.visualize_smpl import visualize_smpl_calibration
body_model_config = dict(
    type='smpl', model_path=model_path, gender=gender)
visualize_smpl_calibration(
    poses=poses,
    betas=betas,
    transl=transl,
    body_model_config=body_model_config,
    K=K,
    R=R,
    T=T,
```

(continues on next page)

(continued from previous page)

```
    output_path='opencv.mp4',
    origin_frames='bg_video.mp4',
    resolution=(1024, 1024))
```

11.1 Different render_choice:

- **visualize mesh:** This is independent of cameras and you could directly set `render_choice` as `hq`(high quality), `mq`(medium quality) or `lq`(low quality).
- **visualize binary silhouettes:** This is independent of cameras and you could directly set `render_choice` as `silhouette`. The output video/images will be binary masks.
- **visualize body part silhouette:** This is independent of cameras and you could directly set `render_choice` as `part_silhouette`. The output video/images will be body part segmentation masks.
- **visualize depth map:** This is independent of cameras and you could directly set `render_choice` as `depth`. The output video/images will be gray depth maps.
- **visualize normal map:** This is independent of cameras and you could directly set `render_choice` as `normal`. The output video/images will be colorful normal maps.
- **visualize point clouds:** This is independent of cameras and you could directly set `render_choice` as `pointcloud`. The output video/images will be point clouds with keypoints.
- **Choose your color:** Set palette as 'white', 'black', 'blue', 'green', 'red', 'yellow', and pass a list of string with the length of num_person. Or send a numpy.ndarray of shape (num_person, 3). Should be normalized color: (1.0, 1.0, 1.0) represents white. The color channel is RGB.
- **Differentiable render:** Set `no_grad=False` and `return_tensor=True`.

11.2 Important parameters:

- **background images:** You could pass `image_array`(`numpy.ndarray` of shape (frame, h, w, 3)) or `frame_list`(`list` of paths of images(.png or .jpg)) or `origin_frames`(str of video path or image folder path). The priority order is `image_array` > `frame_list` > `origin_frames`. If the background images are too big, you should set `read_frames_batch` as `True` to relieve the IO burden. This will be done automatically in the code when you number of frame is large than 500.
- **smpl pose & verts:** There area two ways to pass smpl mesh information: 1). You pass `poses`, `betas`(optional) and `transl`(optional) and `gender`(optional). 2). You pass `verts` directly and the above three will be ignored. The `body_model` or `model_path` is still required if you pass`verts` since we need to get the `faces`. The priority order is `verts` > (`poses` & `betas` & `transl` & `gender`). Check the docstring for details. 3). for multi-person, you should have an extra dim for num_person. E.g., shape of smpl `verts` should be (num_frame, num_person, 6890, 3), shape of smpl `poses` should be (num_frame, num_person, 72), shape of smpl `betas` should be (num_frame, num_person, 10), shape of vibe `pred_cam` should be (num_frame, num_person, 3). This doesn't have influence on `K`, `R`, `T` since they are for every frame.
- **body model:** There are two ways to pass body model: 1). You pass a dict `body_model_config` which containing the same configs as build_body_model 2). You pass `body_model` directly and the above three will be ignored. The priority order is `body_model` > (`model_path` & `model_type` & `gender`). Check the docstring for details.
- **output path:** Output_path could be `None` or `str` of video path or `str` of image folder path. 1). If `None`, no output file will be wrote. 2). If a video path like `xxx.mp4`, a video file will be wrote. Make sure you have enough

space for temporal images. The images will be removed automatically. 3). If a image folder path like `xxx/`, a folder will be created and the images will be wrote into it.

CHAPTER

TWELVE

ADDITIONAL LICENSES

We would like to pay tribute to open-source implementations to which we make reference. Note that they may carry additional license requiresments.

12.1 SMPLify-X

License

Software Copyright License for non-commercial scientific research purposes 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 & Software will not cause any damage of any kind to you or a third party.

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:

@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.

12.2 VIBE

License

Software Copyright License for non-commercial scientific research purposes Please read carefully the following terms and conditions and any accompanying documentation before you download and/or use the VIBE model, data and software, (the "Model & Software"), including 3D meshes, software, and scripts. 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".

This software includes the SMPL Body Model. By downloading this software, you are agreeing to be bound by the terms of the SMPL Model License

```
https://smpl.is.tue.mpg.de/modellicense
```

which is necessary to create SMPL body models.

SMPL bodies that are generated with VIBE can be distributed freely under the SMPL Body License

```
https://smpl.is.tue.mpg.de/bodylicense
```

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 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 Model & 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 Model & 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 Model & Software to train methods/algorithms/neural networks/etc. for commercial use of any kind. By downloading the Model & 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 & Software will not cause any damage of any kind to you or a third party.

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:

@inproceedings{VIBE:CVPR:2020, title = {{VIBE}: Video Inference for Human Body Pose and Shape Estimation}, author = {Kocabas, Muhammed and Athanasiou, Nikos and Black, Michael J.}, booktitle = {Computer Vision and Pattern Recognition (CVPR)}, month = jun, year = {2020}, month_numeric = {6} }

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.

12.3 SPIN

Copyright (c) 2019, University of Pennsylvania, Max Planck Institute for Intelligent Systems All rights reserved.

Redistribution and use 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.

12.4 PARE

License

Software Copyright License for non-commercial scientific research purposes Please read carefully the following terms and conditions and any accompanying documentation before you download and/or use the PARE model, data and software, (the "Model & Software"), including 3D meshes, software, and scripts. 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 Model & 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".

This software includes the SMPL Body Model. By downloading this software, you are agreeing to be bound by the terms of the SMPL Model License

```
https://smpl.is.tue.mpg.de/modellicense
```

which is necessary to create SMPL body models.

SMPL bodies that are generated with PARE can be distributed freely under the SMPL Body License

```
https://smpl.is.tue.mpg.de/bodylicense
```

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 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 Model & 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 Model & 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 Model & Software to train methods/algorithms/neural networks/etc. for commercial use of any kind. By downloading the Model & Software, you agree not to reverse engineer it.

No Distribution The Model & Software and the license herein granted shall not be copied, shared, distributed, resold, 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 & Software will not cause any damage of any kind to you or a third party.

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:

@inproceedings{Kocabas_PARE_2021, title = {{PARE}: Part Attention Regressor for {3D} Human Body Estimation}, author = {Kocabas, Muhammed and Huang, Chun-Hao P. and Hilliges, Otmar and Black, Michael J.}, booktitle = {Proc. International Conference on Computer Vision (ICCV)}, pages = {11127–11137}, month = oct, year = {2021}, doi = {}, month_numeric = {10} }

Commercial licensing opportunities For commercial uses of the Model & 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.

12.5 STAR

License

Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V. (MPG) is holder of all proprietary rights on this computer program. You can only use this computer program if you have closed a license agreement with MPG or you get the right to use the computer program from someone who is authorized to grant you that right. Any use of the computer program without a valid license is prohibited and liable to prosecution.

Contact: ps-license@tuebingen.mpg.de

CHAPTER

THIRTEEN

MMHUMAN3D.APIS

CHAPTER

FOURTEEN

MMHUMAN3D.CORE

CHAPTER

FIFTEEN

MMHUMAN3D.MODELS

CHAPTER

SIXTEEN

MMHUMAN3D.DATA

16.1 data

16.2 datasets

16.3 data_converters

16.4 data_structures

CHAPTER

SEVENTEEN

MMHUMAN3D.UTILS

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- search