Python Documentation

version

October 20, 2020

Contents

Welcome to moseq2-extract's documentation!

moseq2_extract package

CLI Module

moseq2-extract

```
moseq2-extract [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit. [default: False]

aggregate-results

Copies all extracted results (h5, yaml, avi) files from all extracted sessions to a new directory,

```
moseq2-extract aggregate-results [OPTIONS]
```

Options

-i, --input-dir <input_dir>
Directory to find h5 files [default: /Users/aymanzeine/Desktop/moseq/moseq2-extract/docs]

-f, --format <format>
New file name formats from resepective metadata [default: {start_time}_{session_name}_{subject_name}]

-o, --output-dir <output_dir>
Location for storing all results together [default: /Users/aymanzeine/Desktop/moseq/moseq2-extract/docs/aggregate_results/]

--mouse-threshold <mouse_threshold>
Threshold value for mean depth to include frames in aggregated results [default: 0]

convert-raw-to-avi

Converts/Compresses a raw depth file into an avi file (with depth values) that is 8x smaller.

```
moseq2-extract convert-raw-to-avi [OPTIONS] INPUT_FILE
```

Options

-t, --threads <threads>
Number of threads for encoding [default: 3]

--delete
Delete raw file if encoding is sucessful [default: False]

--fps <fps>
Video FPS [default: 30]

-b, --chunk-size <chunk_size>
Chunk size [default: 3000]

-o, --output-file <output_file>
Path to output file

Arguments

INPUT_FILE
Required argument

copy-slice

Copies a segment of an input depth recording into a new video file.

```
moseq2-extract copy-slice [OPTIONS] INPUT_FILE
```

Options

-t, --threads <threads>
Number of threads for encoding [default: 3]

--delete
Delete raw file if encoding is sucessful [default: False]

--fps <fps>
Video FPS [default: 30]

-b, --chunk-size <chunk_size>
Chunk size [default: 3000]

-o, --output-file <output_file>
Path to output file

-c, --copy-slice <copy_slice>
Slice to copy [default: 0, 1000]

Arguments

INPUT_FILE
Required argument

download-flip-file

Downloads Flip-correction model that helps with orienting the mouse during extraction.

```
moseq2-extract download-flip-file [OPTIONS] [CONFIG_FILE]
```

Options

--output-dir <output_dir>
Temp storage [default: /Users/aymanzeine/moseq2]

Arguments

CONFIG_FILE
Optional argument

extract

Processes raw input depth recordings to output a cropped and orientedvideo of the mouse and saves the output+metadata to h5 files in the given output directory.

```
moseq2-extract extract [OPTIONS] INPUT_FILE
```

Options

-p, --progress-bar
Show verbose progress bars. [default: False]

--config-file <config_file>

--use-plane-bground
Use a plane fit for the background. Useful for mice that don't move much [default: False]

--output-dir <output_dir>
Output directory to save the results h5 file

--noise-tolerance <noise_tolerance>
Extent of noise to accept during RANSAC Plane ROI computation. (Special Cases Only) [default: 30]

--erode-iterations <erode_iterations>
Number of erosion iterations to decrease bucket floor size. (Special Cases Only) [default: 0]

--bg-roi-erode <bg_roi_erode>
Size of cv2 Structure Element to erode roi. (Special Cases Only) [default: 1, 1]

--dilate-iterations <dilate_iterations>
Number of dilation iterations to increase bucket floor size. (Special Cases Only) [default: 0]

--bg-sort-roi-by-position-max-rois <bg_sort_roi_by_position_max_rois>
Max original ROIs to sort by position [default: 2]

--bg-sort-roi-by-position <bg_sort_roi_by_position>
Sort ROIs by position [default: False]

--bg-roi-fill-holes <bg_roi_fill_holes>
Fill holes in ROI [default: True]

--bg-roi-gradient-kernel <bg_roi_gradient_kernel>
Kernel size for Sobel gradient filtering [default: 7]

--bg-roi-gradient-threshold <bg_roi_gradient_threshold>
Gradient must be < this to include points [default: 3000]

--bg-roi-gradient-filter <bg_roi_gradient_filter>
Exclude walls with gradient filtering [default: False]

--bg-roi-depth-range <bg_roi_depth_range>
Range to search for floor of arena (in mm) [default: 650, 750]

--camera-type <camera_type>
Helper parameter: auto-sets bg-roi-weights to precomputed values for different camera types. Possible types: ["kinect", "azure", "realsense"] [default: kinect]

Options: kinect|azure|realsense

--bg-roi-weights <bg_roi_weights>
ROI feature weighting (area, extent, dist) [default: 1, 0.1, 1]

--bg-roi-index <bg_roi_index>
Index of which background mask(s) to use [default: 0]

--bg-roi-shape <bg_roi_shape>
Shape to use to dilate roi (ellipse or rect) [default: ellipse]

--bg-roi-dilate <bg_roi_dilate>
Size of strel to dilate roi [default: 10, 10]

-t, --threads <threads>
Number of threads for encoding [default: 3]

--delete
Delete raw file if encoding is sucessful [default: False]

--fps <fps>
Video FPS [default: 30]

-b, --chunk-size <chunk_size>
Chunk size [default: 3000]

-o, --output-file <output_file>
Path to output file

-c, --crop-size <crop_size>
Width and height of cropped mouse image [default: 80, 80]

-n, --num-frames <num_frames>
Number of frames to extract. Will extract full session if set to None.

--min-height <min_height>
Min mouse height from floor (mm) [default: 10]

--max-height <max_height>
Max mouse height from floor (mm) [default: 100]

--detected-true-depth <detected_true_depth>
Option to override automatic depth estimation during extraction. This is only a debugging parameter, for cases where dilate_iterations > 1, otherwise has no effect. Either "auto" or an int value. [default: auto]

--compute-raw-scalars
Compute scalar values from raw cropped frames. [default: False]

--flip-classifier <flip_classifier>
Location of the flip classifier used to properly orient the mouse (.pkl file)

--flip-classifier-smoothing <flip_classifier_smoothing>
Number of frames to smooth flip classifier probabilities [default: 51]

--use-cc <use_cc>
Extract features using largest connected components. [default: True]

--use-tracking-model <use_tracking_model>
Use an expectation-maximization style model to aid mouse tracking. Useful for data with cables [default: False]

--tracking-model-ll-threshold <tracking_model_ll_threshold>
Threshold on log-likelihood for pixels to use for update during tracking [default: -100]

--tracking-model-mask-threshold <tracking_model_mask_threshold>
Threshold on log-likelihood to include pixels for centroid and angle calculation [default: -16]

--tracking-model-ll-clip <tracking_model_ll_clip>
Clip log-likelihoods below this value [default: -100]

--tracking-model-segment <tracking_model_segment>
Segment likelihood mask from tracking model [default: True]

--tracking-model-init <tracking_model_init>
Method for tracking model initialization [default: raw]

--cable-filter-iters <cable_filter_iters>
Number of cable filter iterations [default: 0]

--cable-filter-shape <cable_filter_shape>
Cable filter shape (rectangle or ellipse) [default: rectangle]

--cable-filter-size <cable_filter_size>
Cable filter size (in pixels) [default: 5, 5]

--tail-filter-iters <tail_filter_iters>
Number of tail filter iterations [default: 1]

--tail-filter-size <tail_filter_size>
Tail filter size [default: 9, 9]

--tail-filter-shape <tail_filter_shape>
Tail filter shape [default: ellipse]

-s, --spatial-filter-size <spatial_filter_size>
Space prefilter kernel (median filter, must be odd) [default: 3]

-t, --temporal-filter-size <temporal_filter_size>
Time prefilter kernel (median filter, must be odd) [default: 0]

--chunk-overlap <chunk_overlap>
Frames overlapped in each chunk. Useful for cable tracking [default: 0]

--write-movie <write_movie>
Write a results output movie including an extracted mouse [default: True]

--frame-dtype <frame_dtype>
Data type for processed frames [default: uint8]

Options: uint8|uint16

--centroid-hampel-span <centroid_hampel_span>
Hampel filter span [default: 0]

--centroid-hampel-sig <centroid_hampel_sig>
Hampel filter sig [default: 3]

--angle-hampel-span <angle_hampel_span>
Angle filter span [default: 0]

--angle-hampel-sig <angle_hampel_sig>
Angle filter sig [default: 3]

--model-smoothing-clips <model_smoothing_clips>
Model smoothing clips [default: 0, 0]

--frame-trim <frame_trim>
Frames to trim from beginning and end of data [default: 0, 0]

--compress <compress>
Convert .dat to .avi after successful extraction [default: False]

--compress-chunk-size <compress_chunk_size>
Chunk size for .avi compression [default: 3000]

--compress-threads <compress_threads>
Number of threads for encoding [default: 3]

--skip-completed
Will skip the extraction if it is already completed. [default: False]

Arguments

INPUT_FILE
Required argument

find-roi

Finds the ROI and background distance to subtract from frames when extracting.

```
moseq2-extract find-roi [OPTIONS] INPUT_FILE
```

Options

-p, --progress-bar
Show verbose progress bars. [default: False]

--config-file <config_file>

--use-plane-bground
Use a plane fit for the background. Useful for mice that don't move much [default: False]

--output-dir <output_dir>
Output directory to save the results h5 file

--noise-tolerance <noise_tolerance>
Extent of noise to accept during RANSAC Plane ROI computation. (Special Cases Only) [default: 30]

--erode-iterations <erode_iterations>
Number of erosion iterations to decrease bucket floor size. (Special Cases Only) [default: 0]

--bg-roi-erode <bg_roi_erode>
Size of cv2 Structure Element to erode roi. (Special Cases Only) [default: 1, 1]

--dilate-iterations <dilate_iterations>
Number of dilation iterations to increase bucket floor size. (Special Cases Only) [default: 0]

--bg-sort-roi-by-position-max-rois <bg_sort_roi_by_position_max_rois>
Max original ROIs to sort by position [default: 2]

--bg-sort-roi-by-position <bg_sort_roi_by_position>
Sort ROIs by position [default: False]

--bg-roi-fill-holes <bg_roi_fill_holes>
Fill holes in ROI [default: True]

--bg-roi-gradient-kernel <bg_roi_gradient_kernel>
Kernel size for Sobel gradient filtering [default: 7]

--bg-roi-gradient-threshold <bg_roi_gradient_threshold>
Gradient must be < this to include points [default: 3000]

--bg-roi-gradient-filter <bg_roi_gradient_filter>
Exclude walls with gradient filtering [default: False]

--bg-roi-depth-range <bg_roi_depth_range>
Range to search for floor of arena (in mm) [default: 650, 750]

--camera-type <camera_type>
Helper parameter: auto-sets bg-roi-weights to precomputed values for different camera types. Possible types: ["kinect", "azure", "realsense"] [default: kinect]

Options: kinect|azure|realsense

--bg-roi-weights <bg_roi_weights>
ROI feature weighting (area, extent, dist) [default: 1, 0.1, 1]

--bg-roi-index <bg_roi_index>
Index of which background mask(s) to use [default: 0]

--bg-roi-shape <bg_roi_shape>
Shape to use to dilate roi (ellipse or rect) [default: ellipse]

--bg-roi-dilate <bg_roi_dilate>
Size of strel to dilate roi [default: 10, 10]

Arguments

INPUT_FILE
Required argument

generate-config

Generates a configuration file that holds editable options for extraction parameters.

```
moseq2-extract generate-config [OPTIONS]
```

Options

-o, --output-file <output_file>
[default: config.yaml]

generate-index

Generates an index YAML file containing all extracted session metadata.

```
moseq2-extract generate-index [OPTIONS]
```

Options

-i, --input-dir <input_dir>
Directory to find h5 files [default: /Users/aymanzeine/Desktop/moseq/moseq2-extract/docs]

-o, --output-file <output_file>
Location for storing index [default: /Users/aymanzeine/Desktop/moseq/moseq2-extract/docs/moseq2-index.yaml]

GUI Module

GUI front-end operations accessible from a jupyter notebook.

This module contains all operations included in the CLI module, with some additional preprocessing steps and state-retrieval functionality to facilitate Jupyter notebook usage.

moseq2_extract.gui.**download_flip_command**(output_dir, config_file='', selection=1)
Downloads flip classifier and saves its path in the inputted config file

Parameters:
- **output_dir (str)** (*path to output directory to save flip classifier*)
- **config_file (str)** (*path to config file*)
- **selection (int)** (*index of which flip file to download (default is Adult male C57 classifer)*)

Returns:

Return type: None

`moseq2_extract.gui.`**`generate_index_command`**(`input_dir`, `output_file`)
Generates Index File based on aggregated sessions

Parameters:
- **input_dir (str)** (*path to aggregated_results/ dir*)
- **output_file (str)** (*index file name*)

Returns: **output_file (str)**

Return type: path to index file.

`moseq2_extract.gui.`**`get_selected_sessions`**(`to_extract`, `extract_all`)
Given user input, the function will return either selected sessions to extract, or all the sessions.

Parameters:
- **to_extract (list)** (*list of paths to sessions to extract*)
- **extract_all (bool)** (*boolean to include all sessions and skip user-input prompt.*)

Returns: **to_extract (list)**

Return type: new list of selected sessions to extract.

General Utilities Module

General helper/convenience utilities that are implemented throughout the extract package.

`moseq2_extract.util.`**`_load_h5_to_dict`**(`file: h5py._hl.files.File`, `path`) → dict
Loads h5 contents to dictionary object.

Parameters:
- **h5file (h5py.File)** (*file path to the given h5 file or the h5 file handle*)
- **path (str)** (*path to the base dataset within the h5 file*)

Returns: **out (dict)**

Return type: a dict with h5 file contents with the same path structure

`moseq2_extract.util.`**`build_path`**(`keys: dict`, `format_string: str`, `snake_case=True`) → str
Produce a new file name using keys collected from extraction h5 files. The format string must be using python's formatting specification, i.e. '{subject_name}_{session_name}'.

Parameters:
- **keys (dict)** (*dictionary specifying which keys used to produce the new file name*)
- **format_string (str)** (the string to reformat using the *keys* dictionary)
- **snake_case (bool)** (*whether to save the files with snake_case*)

Returns: **out (str)**

Return type: a newly formatted filename useable with any operating system

`moseq2_extract.util.`**`camel_to_snake`**(`s`)
Converts CamelCase to snake_case

Parameters: **s (str)** (*CamelCase string to convert to snake_case.*)

Returns: **(str)**

Return type: string in snake_case

`moseq2_extract.util.`**`check_filter_sizes`**(`config_data`)
Checks if inputted spatial and temporal filter kernel sizes are odd numbers. Incrementing the value if not.

Parameters: **config_data (dict)** (*Configuration dict holding all extraction parameters*)

Returns: **config_data (dict)**
Return type: Updated configuration dict

`moseq2_extract.util.`**`clean_dict`**(`dct`)
Standardizes types of dict value.

Parameters: **dct (dict)** (*dict object with mixed type value objects.*)
Returns: **dct (dict)**
Return type: dict object with list value objects.

`moseq2_extract.util.`**`clean_file_str`**(`file_str: str`, `replace_with: str = '-'`) → str
Removes invalid characters for a file name from a string.

Parameters:
- **file_str (str)** (*filename substring to replace*)
- **replace_with (str)** (*value to replace str with*)

Returns: **out (str)**
Return type: cleaned file string

`moseq2_extract.util.`**`click_param_annot`**(`click_cmd`)
Given a click.Command instance, return a dict that maps option names to help strings. Currently skips click.Arguments, as they do not have help strings.

Parameters: **click_cmd (click.Command)** (*command to introspect*)
Returns: **annotations (dict)**
Return type: click.Option.human_readable_name as keys; click.Option.help as values

`moseq2_extract.util.`**`command_with_config`**(`config_file_param_name`)

`moseq2_extract.util.`**`convert_pxs_to_mm`**(`coords`, `resolution=(512, 424)`, `field_of_view=(70.6, 60)`, `true_depth=673.1`)
Converts x, y coordinates in pixel space to mm.
http://stackoverflow.com/questions/17832238/kinect-intrinsic-parameters-from-field-of-view/18199938#18199938
http://www.imaginativeuniversal.com/blog/post/2014/03/05/quick-reference-kinect-1-vs-kinect-2.aspx
http://smeenk.com/kinect-field-of-view-comparison/

Parameters:
- **coords (list)** (*list of x,y pixel coordinates*)
- **resolution (tuple)** (*image dimensions*)
- **field_of_view (tuple)** (*width and height scaling params*)
- **true_depth (float)** (*detected true depth*)

Returns: **new_coords (list)**
Return type: x,y coordinates in mm

`moseq2_extract.util.`**`convert_raw_to_avi_function`**(`input_file`, `chunk_size=2000`, `fps=30`, `delete=False`, `threads=3`)
Converts depth file to avi file.

Parameters:
- **input_file (str)** (*path to depth file*)
- **chunk_size (int)** (*size of chunks to process at a time*)
- **fps (int)** (*frames per second*)
- **delete (bool)** (*whether to delete original depth file*)
- **threads (int)** (*number of threads to write video.*)

Returns:
Return type: None

`moseq2_extract.util.`**`dict_to_h5`**(`h5`, `dic`, `root='/'`, `annotations=None`)
Save an dict to an h5 file, mounting at root. Keys are mapped to group names recursively.

Parameters:
- **h5 (h5py.File instance)** (*h5py.file object to operate on*)
- **dic (dict)** (*dictionary of data to write*)
- **root (string)** (*group on which to add additional groups and datasets*)
- **annotations (dict)** (*annotation data to add to corresponding h5 datasets. Should contain same keys as dic.*)

Returns:

Return type: None

`moseq2_extract.util.`**`escape_path`** (`path`)
Given current path, will return a path to return to original base directory. (Used in recursive h5 search, etc.)

Parameters: **path (str)** (*path to current working dir*)

Returns: **path (str)**

Return type: path to original base_dir

`moseq2_extract.util.`**`filter_warnings`** (`func`)

`moseq2_extract.util.`**`gen_batch_sequence`** (`nframes`, `chunk_size`, `overlap`, `offset=0`)
Generates batches used to chunk videos prior to extraction.

Parameters:
- **nframes (int)** (*total number of frames*)
- **chunk_size (int)** (*desired chunk size*)
- **overlap (int)** (*number of overlapping frames*)
- **offset (int)** (*frame offset*)

Returns:

Return type: Yields list of batches

`moseq2_extract.util.`**`get_bucket_center`** (`img`, `true_depth`, `threshold=650`)
https://stackoverflow.com/questions/19768508/python-opencv-finding-circle-sun-coordinates-of-center-the-circle-from-pictu Finds Centroid coordinates of circular bucket.

Parameters:
- **img (2d np.ndaarray)** (*original background image.*)
- **true_depth (float)** (*distance value from camera to bucket floor (automatically pre-computed)*)
- **threshold (float)** (*distance values to accept region into detected circle. (used to reduce fall noise interference)*)

Returns: **cX (int)** (*x-coordinate of circle centroid*) **cY (int)** (*y-coordinate of circle centroid*)

`moseq2_extract.util.`**`get_frame_range_indices`** (`config_data`, `nframes`)
Computes the total number of frames to be extracted, given the total number of frames and an initial frame index starting point.

Parameters:
- **config_data (dict)** (*dictionary holding all extraction parameters*)
- **nframes (int)** (*total number of requested frames to extract*)

Returns: **nframes (int)** (*total number of frames to extract*) **first_frame_idx (int)** (*index of the frame to begin extraction from*) **last_frame_idx (int)** (*index of the last frame in the extraction*)

`moseq2_extract.util.`**`get_strels`** (`config_data`)
Get dictionary object of cv2 StructuringElements for image filtering given a dict of configurations parameters.

Parameters: **config_data (dict)** (*dict containing cv2 Structuring Element parameters*)

Returns: **str_els (dict)**

Return type: dict containing cv2 StructuringElements used for image filtering

`moseq2_extract.util.`**`graduate_dilated_wall_area`** (`bground_im`, `config_data`, `strel_dilate`, `output_dir`)

Creates a gradient to represent the dilated (now visible) bucket wall regions. Only is used if background is dilated to capture larger rodents in convex shaped buckets (_/). This is done to handle noise attributed by bucket walls being slanted, and thus being picked up as large noise depth values. Moreover, to appropriately subtract the background from input images during extraction without obscuring the rodent, or including unwanted wall regions.

Parameters:
- **bground_im (2d np.ndarray)** (*the bucket floor image computed as the median distance throughout the session.*)
- **config_data (dict)** (*dictionary containing helper user configuration parameters.*)
- **strel_dilate (cv2.structuringElement)** (*dilation structuring element used to dilate background image.*)
- **output_dir (str)** (*path to save newly computed background to use.*)

Returns: **bground_im (2d np.ndarray)**

Return type: the new background image with a gradient around the floor from high to low depth values.

`moseq2_extract.util.`**`h5_to_dict`** (`h5file`, `path`) → dict

Loads h5 contents to dictionary object.

Parameters:
- **h5file (str or h5py.File)** (*file path to the given h5 file or the h5 file handle*)
- **path (str)** (*path to the base dataset within the h5 file*)

Returns: **out (dict)**

Return type: a dict with h5 file contents with the same path structure

`moseq2_extract.util.`**`load_found_session_paths`** (`input_dir`, `exts`)

Given an input directory and file extensions, this function will return all depth file paths found in the inputted parent (input) directory. :Parameters: * **input_dir (str)** (*path to parent directory holding all the session folders.*)

- **exts (list or str)** (*list of extensions to search for, or a single extension in string form.*)

Returns: **files (list)**

Return type: sorted list of all paths to found depth files

`moseq2_extract.util.`**`load_metadata`** (`metadata_file`)

Loads metadata from session metadata.json file.

Parameters: **metadata_file (str)** (*path to metadata file*)

Returns: **metadata (dict)**

Return type: key-value pairs of JSON contents

`moseq2_extract.util.`**`load_textdata`** (`data_file`, `dtype=<class 'numpy.float32'>`)

Loads timestamp from txt/csv file. Timestamps are separated by newlines and have a space-separated data indicator, (in most cases, the indicator equals 0)

Parameters:
- **data_file (str)** (*path to timestamp file*)
- **dtype (dtype)** (*data type of timestamps*)

Returns: **data (np.ndarray)** (*timestamp data*) **timestamps (np.array)** (*time stamp keynames.*)

`moseq2_extract.util.`**`load_timestamps`** (`timestamp_file`, `col=0`, `alternate=False`)

Read timestamps from space delimited text file.

Parameters:
- **timestamp_file (str)** (*path to timestamp file*)
- **col (int)** (*column in ts file read.*)
- **alternate (boolean)** (*flag set to true if timestamps were saved in a csv file*)

Returns: **ts (1D array)**

Return type: list of timestamps

`moseq2_extract.util.`**`make_gradient`** (`width`, `height`, `h`, `k`, `a`, `b`, `theta=0`)

https://stackoverflow.com/questions/49829783/draw-a-gradual-change-ellipse-in-skimage/49848093#49848093
Creates gradient around bucket floor representing slanted wall values. This is done by drawing an "ellipse" of equal x,y radii, resulting in a circle with weighted depth values from highest to lowest surrounding the circumference of the circle

Parameters:
- **width (int)** (*bounding box width*)
- **height (int) bounding box height**
- **h (int)** (*centroid x coordinate*)
- **k (int)** (*centroid y coordinate*)
- **a (int)** (*x-radius of drawn ellipse*)
- **b (int)** (*y-radius of drawn ellipse*)
- **theta (float)** (*degree to rotate ellipse in radians. (has no effect if drawing a circle)*)

Returns: **(2d np.ndarray)** (*numpy array with weighted values from 0.08 -> 0.8 representing the proportion of values*) *to create a gradient from. 0.8 being the proportioned values closest to the circle wall.*

`moseq2_extract.util.`**`mouse_threshold_filter`**`(h5file, thresh=0)`
Filters frames in h5 files by threshold value

Parameters:
- **h5file (str)** (*path to h5 file*)
- **thresh (int)** (*threshold at which to apply filter*)

Returns: **(3d-np boolean array)**

Return type: array of regions to include after threshold filter.

`moseq2_extract.util.`**`read_yaml`**`(yaml_file)`
Reads yaml file into dict object

Parameters: **yaml_file (str)** (*path to yaml file*)

Returns: **return_dict (dict)**

Return type: dict of yaml contents

`moseq2_extract.util.`**`recursive_find_h5s`**
`(root_dir='/Users/aymanzeine/Desktop/moseq/moseq2-extract/docs', ext='.h5', yaml_string='{}.yaml')`
Recursively find h5 files, along with yaml files with the same basename

Parameters:
- **root_dir (str)** (*path to base directory to begin recursive search in.*)
- **ext (str)** (*extension to search for*)
- **yaml_string (str)** (*string for filename formatting when saving data*)

Returns: **h5s (list)** (*list of found h5 files*) **dicts (list)** (*list of found metadata files*) **yamls (list)** (*list of found yaml files*)

`moseq2_extract.util.`**`recursive_find_unextracted_dirs`**
`(root_dir='/Users/aymanzeine/Desktop/moseq/moseq2-extract/docs', session_pattern='session_\\d+\\.(?:tgz|tar\\.gz)', filename='.dat', yaml_path='proc/results_00.yaml', metadata_path='metadata.json', skip_checks=False)`
Recursively find unextracted (or incompletely extracted) directories

Parameters:
- **root_dir (os Path-like)** (*path to base directory to start recursive search from.*)
- **session_pattern (str)** (*folder name pattern to search for*)
- **filename (str)** (*file extension to search for*)
- **yaml_path (str)** (*path to respective extracted metadata*)
- **metadata_path (str)** (*path to relative metadata.json files*)
- **skip_checks (bool)** (*indicates whether to check if the files exist at the given relative paths*)

Returns: **proc_dirs (1d-list)**

Return type: list of paths to each unextracted session's proc/ directory

`moseq2_extract.util.`**`scalar_attributes`**()
Gets scalar attributes

Returns: **attributes (dict)**

Return type: collection of metadata keys and descriptions.

`moseq2_extract.util.`**`select_strel`**(`string='e'`, `size=(10, 10)`)
Returns structuring element of specified shape.

Parameters:
- **string (str)** (*indicates whether to use ellipse or rectangle*)
- **size (tuple)** (*size of structuring element*)

Returns:

Return type: strel (cv2.StructuringElement)

`moseq2_extract.util.`**`set_bg_roi_weights`**(`config_data`)
Reads any inputted camera type and sets the bg_roi_weights to some precomputed values. If no camera_type is inputted, program will assume a kinect camera is being used.

Parameters: **config_data (dict)** (*dictionary containing all input parameters to a CLI/GUI command.*)

Returns: **config_data (dict)**

Return type: updated dictionary with bg-roi-weights to use in extraction/ROI retrieval.

`moseq2_extract.util.`**`set_bground_to_plane_fit`**(`bground_im`, `plane`, `output_dir`)
Replaces median-computed background image with plane fit. Only occurs if config_data['use_plane_bground'] == True.

Parameters:
- **bground_im (2D numpy array)** (*Background image computed via median value of depth video.*)
- **plane (2D numpy array)** (*Computed ROI Plane using RANSAC.*)
- **output_dir (str)** (*Path to write updated background image to.*)

Returns: **bground_im (2D numpy array)**

Return type: Plane fit version of the background image.

`moseq2_extract.util.`**`strided_app`**(a, L, S)
from https://stackoverflow.com/questions/40084931/taking-subarrays-from-numpy-array-with-given-stride-stepsize/40085052#40085052

Parameters:
- **a (np.ndarray) - array to get subarrarys from.**
- **L (int) - Window Length**
- **S (int) - Stride size**

Returns:

Return type: (np.ndarray) - array of subarrays at stride S.

`moseq2_extract.util.`**`time_str_for_filename`**(`time_str: str`) → str

Process the time string supplied by moseq to be used in a filename. This removes colons, milliseconds, and timezones.

Parameters: **time_str (str)** (*time str to format*)
Returns: **out (str)**
Return type: formatted timestamp str

Subpackages

moseq2_extract.extract package

Extract - Extract Module

Extraction helper utility for performing multiple cleaning, cropping and rotating operations.

Given a "chunk" or segment of an input depth video, this function will first find and subtract the ROI, then optionally perform Expectation Maximization tracking for mouse tracking with occlusions such as photometry fibers. Next, it will apply some spatial/temporal filtering before cropping, and aligning the rodent such that it is always facing east (with the help of the flip classifier).

It will store each chunk's extracted data and metadata in a dictionary that will be later written to the corresponding results_00.h5 file.

```
moseq2_extract.extract.extract.extract_chunk(chunk, use_tracking_model=False,
spatial_filter_size=(3,), temporal_filter_size=None, tail_filter_iters=1,
iters_min=0, strel_tail=array([[0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 1, 1,
0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1,
1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1,
0], [0, 0, 0, 0, 1, 0, 0, 0, 0]], dtype=uint8), strel_min=array([[1, 1, 1, 1, 1], [1,
1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]], dtype=uint8),
min_height=10, max_height=100, mask_threshold=-20, use_cc=False, bground=None,
roi=None, rho_mean=0, rho_cov=0, tracking_ll_threshold=-100,
tracking_model_segment=True, tracking_init_mean=None, tracking_init_cov=None,
tracking_init_strel=array([[0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0]], dtype=uint8), flip_classifier=None,
flip_classifier_smoothing=51, frame_dtype='uint8', progress_bar=True, crop_size=(80,
80), true_depth=673.1, centroid_hampel_span=5, centroid_hampel_sig=3,
angle_hampel_span=5, angle_hampel_sig=3, model_smoothing_clips=(-300, -150),
tracking_model_init='raw', compute_raw_scalars=False, **kwargs)
```

This function looks for a mouse in background-subtracted frames from a chunk of depth video. It is called from the moseq2_extract.helpers.extract module.

Parameters:

- **chunk (3d np.ndarray)** (*chunk to extract - (chunksize, height, width)*)
- **use_tracking_model (bool)** (*The EM tracker uses expectation-maximization to fit a 3D gaussian on a frame-by-frame*) – basis to the mouse's body and determine if pixels are mouse vs cable.
- **spatial_filter_size (tuple)** (*spatial kernel size*)
- **temporal_filter_size (tuple)** (*temporal kernel size*)
- **tail_filter_iters (int)** (*number of filtering iterations on mouse tail*)
- **iters_min (int)** (*minimum tail filtering filter kernel size*)
- **strel_tail (cv2::StructuringElement - Ellipse)** (*filtering kernel size to filter out mouse tail.*)
- **strel_min (cv2::StructuringElement - Rectangle)** (*filtering kernel size to filter mouse body in cable recording cases.*)
- **min_height (int)** (*minimum (mm) distance of mouse to floor.*)
- **max_height (int)** (*maximum (mm) distance of mouse to floor.*)
- **mask_threshold (int)** (*Threshold on log-likelihood to include pixels for centroid and angle calculation*)
- **use_cc (bool)** (*boolean to use connected components in cv2 structuring elements*)
- **bground (np.ndarray)** (*numpy array represented previously computed background*)
- **roi (np.ndarray)** (*numpy array represented previously computed roi*)
- **rho_mean (int)** (*smoothing parameter for the mean*)
- **rho_cov (int)** (*smoothing parameter for the covariance*)
- **tracking_ll_threshold (float)** (*threshold for calling pixels a cable vs a mouse (usually between -16 to -12).*) – If the log-likelihood falls below this value, pixels are considered cable.
- **tracking_model_segment (bool)** (*boolean for whether to use only the largest blob for EM updates.*)
- **tracking_init_mean (float)** (*Initialized mean value for EM Tracking*)
- **tracking_init_cov (float)** (*Initialized covariance value for EM Tracking*)
- **tracking_init_strel (cv2::StructuringElement - Ellipse)**
- **flip_classifier (str)** (*path to pre-selected flip classifier.*)
- **flip_classifier_smoothing (int)** (*amount of smoothing to use for flip classifier.*)
- **frame_dtype (str)** (*Data type for processed frames*)
- **save_path** (*(str): Path to save extracted results*)
- **progress_bar (bool)** (*Display progress bar*)
- **crop_size (tuple)** (*size of the cropped mouse image.*)
- **true_depth (float)** (*previously computed detected true depth value.*)
- **centroid_hampel_span (int)** (*Hampel filter span kernel size*)
- **centroid_hampel_sig (int)** (*Hampel filter standard deviation*)
- **angle_hampel_span (int)** (*Angle filter span kernel size*)
- **angle_hampel_sig (int)** (*Angle filter standard deviation*)
- **model_smoothing_clips (tuple)** (*Model smoothing clips*)
- **tracking_model_init (str)** (*Method for tracking model initialization*)
- **compute_raw_scalars (bool)** (*Compute scalars from unfiltered crop-rotated data.*)

Returns: **results** (*(3d np.ndarray) - (nframes, crop_height, crop_width)*) *extracted cropped, oriented and centered RGB video chunk to be written to file.*

Extract - Proc Module

Video pre-processing utilities for detecting ROIs and extracting raw data.

`moseq2_extract.extract.proc.`**`apply_roi`**(`frames`, `roi`)

Apply ROI to data, consider adding constraints (e.g. mod32==0).

Parameters:
- **frames (3d np.ndarray)** (*input frames to apply ROI.*)
- **roi (2d np.ndarray)** (*selected ROI to extract from input images.*)

Returns: **cropped_frames (3d np.ndarray)**

Return type: Frames cropped around ROI Bounding Box.

`moseq2_extract.extract.proc.`**`clean_frames`**(`frames, prefilter_space=(3,), prefilter_time=None, strel_tail=array([[0, 0, 0, 1, 0, 0, 0], [0, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1, 0], [0, 0, 0, 1, 0, 0, 0]], dtype=uint8), iters_tail=None, frame_dtype='uint8', strel_min=array([[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]], dtype=uint8), iters_min=None, progress_bar=False`)

Simple filtering, median filter and morphological opening.

Parameters:
- **frames (3d np.ndarray)** (*Frames (nframes x r x c) to filter.*)
- **prefilter_space (tuple)** (*kernel size for spatial filtering*)
- **prefilter_time (tuple)** (*kernel size for temporal filtering*)
- **strel_tail (cv2.StructuringElement)** (*Element for tail filtering.*)
- **iters_tail (int)** (*number of iterations to run opening*)
- **frame_dtype (str)** (*frame encodings*)
- **strel_min (int)** (*minimum kernel size*)
- **iters_min (int)** (*minimum number of filtering iterations*)
- **progress_bar (bool)** (*display progress bar*)

Returns: **filtered_frames (3d np array)**

Return type: frame x r x c

`moseq2_extract.extract.proc.`**`compute_scalars`**(`frames`, `track_features`, `min_height=10`, `max_height=100`, `true_depth=673.1`)

Computes scalars.

Parameters:
- **frames (3d np.ndarray)** (*frames x r x c, uncropped mouse*)
- **track_features (dict)** (*dictionary with tracking variables (centroid and orientation)*)
- **min_height (float)** (*minimum height of the mouse*)
- **max_height (float)** (*maximum height of the mouse*)
- **true_depth (float)** (*detected true depth*)

Returns: **features (dict)**

Return type: dictionary of scalars

`moseq2_extract.extract.proc.`**`crop_and_rotate_frames`**(`frames`, `features`, `crop_size=(80, 80)`, `progress_bar=False`)

Crops mouse from image and orients it s.t it is always facing east.

Parameters:
- **frames (3d np.ndarray)** (*frames to crop and rotate*)
- **features (dict)** (*dict of extracted features, found in result_00.h5 files.*)
- **crop_size (tuple)** (*size of cropped image.*)
- **progress_bar (bool)** (*Display progress bar.*)
- **gui (bool)** (*indicate GUI is executing function*)

Returns: **cropped_frames (3d np.ndarray)**

Return type: Crop and rotated frames.

`moseq2_extract.extract.proc.`**`feature_hampel_filter`**`(features, centroid_hampel_span=None, centroid_hampel_sig=3, angle_hampel_span=None, angle_hampel_sig=3)`

Filters computed extraction features using Hampel Filtering. Used to detect and filter out outliers.

Parameters:
- **features (dict)** (*dictionary of video features*)
- **centroid_hampel_span (int)** (*Centroid Hampel Span Filtering Kernel Size*)
- **centroid_hampel_sig (int)** (*Centroid Hampel Signal Filtering Kernel Size*)
- **angle_hampel_span (int)** (*Angle Hampel Span Filtering Kernel Size*)
- **angle_hampel_sig (int)** (*Angle Hampel Span Filtering Kernel Size*)

Returns: **features (dict)**

Return type: filtered version of input dict.

`moseq2_extract.extract.proc.`**`get_bbox`**`(roi)`

Given a binary mask, return an array with the x and y boundaries

Parameters: **roi (2d np.ndarray)** (*ROI boolean mask to calculate bounding box.*)

Returns: **bbox (2d np.ndarray)**

Return type: Bounding Box around ROI

`moseq2_extract.extract.proc.`**`get_bground_im`**`(frames)`

Returns background

Parameters: **frames (3d numpy array)** (*frames x r x c, uncropped mouse*)

Returns: **bground (2d numpy array)**

Return type: r x c, background image

`moseq2_extract.extract.proc.`**`get_bground_im_file`**`(frames_file, frame_stride=500, med_scale=5, **kwargs)`

Returns background from file

Parameters:
- **frames_file (str)** (*path to data with frames*)
- **frame_stride (int)** (*stride size between frames for median bground calculation*)
- **med_scale (int)** (*kernel size for median blur for background images.*)
- **kwargs**

Returns: **bground (2d numpy array)**

Return type: r x c, background image

`moseq2_extract.extract.proc.`**`get_flips`**`(frames, flip_file=None, smoothing=None)`

Predicts frames where mouse orientation is flipped to later correct.

Parameters:
- **frames (3d numpy array)** (*frames x r x c, cropped mouse*)
- **flip_file (str)** (*path to joblib dump of scipy random forest classifier*)
- **smoothing (int)** (*kernel size for median filter smoothing of random forest probabilities*)

Returns: **flips (bool array)**

Return type: true for flips

`moseq2_extract.extract.proc.`**`get_frame_features`**`(frames, frame_threshold=10, mask=array([], dtype=float64), mask_threshold=-30, use_cc=False, progress_bar=False)`

Use image moments to compute features of the largest object in the frame

Parameters:

- **frames (3d np.ndarray)** (*input frames*)
- **frame_threshold (int)** (*threshold in mm separating floor from mouse*)
- **mask (3d np.ndarray)** (*input frame mask for parts not to filter.*)
- **mask_threshold (int)** (*threshold to include regions into mask.*)
- **use_cc (bool)** (*Use connected components.*)
- **progress_bar (bool)** (*Display progress bar.*)

Returns: **features (dict of lists)** (*dictionary with simple image features*) **mask (3d np.ndarray)** (*input frame mask.*)

`moseq2_extract.extract.proc.`**`get_largest_cc`**`(frames, progress_bar=False)`

Returns largest connected component blob in image

Parameters:

- **frames (3d numpy array)** (*frames x r x c, uncropped mouse*)
- **progress_bar (bool)** (*display progress bar*)

Returns: **flips (3d bool array)**

Return type: frames x r x c, true where blob was found

```
moseq2_extract.extract.proc.get_roi(depth_image, strel_dilate=array([[1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]],
dtype=uint8), dilate_iterations=0, erode_iterations=0, strel_erode=None,
noise_tolerance=30, bg_roi_weights=(1, 0.1, 1), overlap_roi=None,
bg_roi_gradient_filter=False, bg_roi_gradient_kernel=7,
bg_roi_gradient_threshold=3000, bg_roi_fill_holes=True, get_all_data=False, **kwargs)
```

Get an ROI using RANSAC plane fitting and simple blob features

Parameters:
- **depth_image (2d np.ndarray)** (*Singular depth image frame.*)
- **strel_dilate (cv2.StructuringElement - Rectangle)** (*dilation shape to use.*)
- **dilate_iterations (int)** (*number of dilation iterations.*)
- **erode_iterations (int)** (*number of erosion iterations.*)
- **strel_erode (int)** (*image erosion kernel size.*)
- **noise_tolerance (int)** (*threshold to use for noise filtering.*)
- **bg_roi_weights (tuple)** (*weights describing threshold to accept ROI.*)
- **overlap_roi (np.ndarray)** (*list of ROI boolean arrays to possibly combine.*)
- **bg_roi_gradient_filter (bool)** (*Boolean for whether to use a gradient filter.*)
- **bg_roi_gradient_kernel (tuple)** (*Kernel size of length 2, e.g. (1, 1.5)*)
- **bg_roi_gradient_threshold (int)** (*Threshold for noise gradient filtering*)
- **bg_roi_fill_holes (bool)** (*Boolean to fill any missing regions within the ROI.*)
- **get_all_data (bool)** (*If True, returns all ROI data, else, only return ROIs and computed Planes*)
- **kwargs (dict) Dictionary containing `bg_roi_depth_range` parameter for plane_ransac()**

Returns: **rois (list)** (*list of 2d roi images.*) **roi_plane (2d np.ndarray)** (*computed ROI Plane using RANSAC.*) **bboxes (list)** (*list of computed bounding boxes for each respective ROI.*) **label_im (list)** (*list of scikit-image image properties*) **ranks (list)** (*list of ROI ranks.*) **shape_index (list)** (*list of rank means.*)

`moseq2_extract.extract.proc.`**`im_moment_features`**`(IM)`
Use the method of moments and centralized moments to get image properties.

Parameters: **IM (2d numpy array)** (*depth image*)
Returns: **features (dict)** – centroid, and ellipse axis length
Return type: returns a dictionary with orientation,

`moseq2_extract.extract.proc.`**`model_smoother`**`(features, ll=None, clips=(-300, -125))`
Spatial feature filtering.

Parameters:
- **features (dict)** (*dictionary of extraction scalar features*)
- **ll (np.array)** (*list of loglikelihoods of pixels in frame*)
- **clips (tuple)** (*tuple to ensure video is indexed properly*)

Returns:
Return type: features (dict) - smoothed version of input features

`moseq2_extract.extract.proc.`**`threshold_chunk`**`(chunk, min_height, max_height)`
Thresholds out depth values that are less than min_height and larger than max_height.

Parameters:
- **chunk (3D np.ndarray)** (*Chunk of frames to threshold (nframes, width, height)*)
- **min_height (int)** (*Minimum depth values to include after thresholding.*)
- **max_height (int)** (*Maximum depth values to include after thresholding.*)
- **dilate_iterations (int)** (*Number of iterations the ROI was dilated.*)

Returns: **chunk (3D np.ndarray)**
Return type: Updated frame chunk.

Extract - ROI Module

ROI detection pre-processing utilities for fitting a plane to an input depth image.

`moseq2_extract.extract.roi.`**`plane_fit3`**(`points`)
Fit a plane to 3 points (min number of points for fitting a plane)

Parameters: **points (2d numpy array)** (*each row is a group of points, columns correspond to x,y,z.*)
Returns: **plane (1d numpy array)**
Return type: linear plane fit–>a*x+b*y+c*z+d

`moseq2_extract.extract.roi.`**`plane_ransac`**(`depth_image`, `bg_roi_depth_range=(650, 750)`, `iters=1000`, `noise_tolerance=30`, `in_ratio=0.1`, `progress_bar=False`, `mask=None`, `**kwargs`)
Naive RANSAC implementation for plane fitting

Parameters:
- **depth_image (2d numpy array)** (*hxw, background image to fit plane to*)
- **bg_roi_depth_range (tuple)** (*min/max depth (mm) to consider pixels for plane*)
- **iters (int)** (*number of RANSAC iterations*)
- **noise_tolerance (float)** (*dist. from plane to consider a point an inlier*)
- **in_ratio (float)** (*frac. of points required to consider a plane fit good*)
- **progress_bar (bool)** (*display progress bar*)
- **mask (bool 2d np.array)** (*boolean mask to find region to use*)
- **kwargs (dict)** (*dictionary containing extra keyword arguments from moseq2_extract.proc.get_roi()*)

Returns: **best_plane (1d numpy array)** (*plane fit to data*) **dist (1d numpy array)** (*distance of the calculated coordinates and "best plane"*)

Extract - Track Module

Expectation-Maximization mouse tracking utilities.

`moseq2_extract.extract.track.`**`em_get_ll`**(`frames`, `mean`, `cov`, `progress_bar=False`)
Returns likelihoods for each frame given tracker parameters

Parameters:
- **frames (3d numpy array)** (*depth frames*)
- **mean (2d numpy array)** (*frames x d, mean estimates*)
- **cov (3d numpy array)** (*frames x d x d, covariance estimates*)
- **progress_bar (bool)** (*use a progress bar*)

Returns: **ll (3d numpy array)**
Return type: frames x rows x columns, log likelihood of all pixels in each frame

`moseq2_extract.extract.track.`**`em_init`**(`depth_frame, depth_floor, depth_ceiling, init_strel=array([[0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0]], dtype=uint8), strel_iters=1`)
Initialize EM Mask.
Estimates depth frame contours using OpenCV, and selects the largest chosen contour to create a mask.

Parameters:
- **depth_frame (2d numpy array)** (*depth frame to initialize mask with.*)
- **depth_floor (float)** (*distance from camera to bucket floor.*)
- **depth_ceiling (float)** (*max depth value.*)
- **init_strel (cv2.structuringElement)** (*structuring Element to compute mask.*)
- **strel_iters (int)** (*number of morphological iterations.*)

Returns: **mouse_mask (2d numpy array)**
Return type: mask of depth frame.

`moseq2_extract.extract.track.`**`em_iter`**(`data`, `mean`, `cov`, `lamd=0.1`, `epsilon=0.1`, `max_iter=25`)

EM tracker iteration function. Function will iteratively update the mean and covariance variables using Expectation Maximization up to the max inputted number of iterations.
Note: the rate/probability at which the mean and cov are updated are dependent on the tolerance variable epsilon.

Parameters:
- **data (3d numpy array)** (*nx3, x, y, z coordinates to use*)
- **mean (1d numpy array)** (*dx1, current mean estimate*)
- **cov (2d numpy array)** (*dxd, current covariance estimate*)
- **lambd (float)** (*constant to add to diagonal of covariance matrix*)
- **epsilon (float)** (*tolerance on change in likelihood to terminate iteration*)
- **max_iter (int)** (*maximum number of EM iterations*)

Returns: **mean (1d numpy array)** (*updated mean*) **cov (2d numpy array)** (*updated covariance*)

`moseq2_extract.extract.track.`**`em_tracking`**(`frames, raw_frames, segment=True, ll_threshold=-30, rho_mean=0, rho_cov=0, depth_floor=10, depth_ceiling=100, progress_bar=True, init_mean=None, init_cov=None, init_frames=10, init_method='raw', init_strel=array([[0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0]], dtype=uint8)`)

Naive tracker, use EM update rules to follow a 3D Gaussian

around the room.

Parameters:
- **frames (3d numpy array)** (*filtered frames - nframes x r x c.*)
- **raw_frames (3d numpy array)** (*chunk to track mouse in.*)
- **segment (bool)** (*use only the largest blob for em updates*)
- **ll_threshold (float)** (*threshold on log likelihood for segmentation*)
- **rho_mean (float)** (*smoothing parameter for the mean*)
- **rho_cov (float)** (*smoothing parameter for the covariance*)
- **depth_floor (float)** (*height in mm for separating mouse from floor*)
- **depth_ceiling (float)** (*max height in mm for mouse from floor.*)
- **progress_bar (bool)** (*display progress bar.*)
- **init_mean (np.ndarray)** (*array of inital frame pixel means.*)
- **init_cov (np.ndarray)** (*array of inital frame pixel covariances.*)
- **init_frames (int)** (*number of frames to include in the init calulation*)
- **init_method (str)** (*mode in which to process inputs*)
- **init_strel (cv2.structuringElement)** (*structuring Element to compute mask.*)

Returns: **model_parameters (dict)**

Return type: mean and covariance estimates for each frame

moseq2_extract.helpers package

Helpers - Data Module

Data selection, writing, and loading utilities. Contains helper functions to aid mostly in handling/storing data during extraction. Remainder of functions are used in the data aggregation process.

`moseq2_extract.helpers.data.`**`build_index_dict`**(`files_to_use`)

Given a list of files and respective metadatas to include in an index file, creates a dictionary that will be saved later as the index file. It will contain all the inputted file paths with their respective uuids, group names, and metadata.

Note: This is a direct helper function for generate_index_wrapper().

You can expect the following structure from file_tup elements:

('path_to_extracted_h5', 'path_to_extracted_yaml', {file_status_dict})

Parameters: **files_to_use (list)** (*list of paths to extracted h5 files.*)

Returns: **output_dict (dict)**

Return type: index-file dictionary containing all aggregated extractions.

`moseq2_extract.helpers.data.`**`build_manifest`**`(loaded, format, snake_case=True)`

aggregate_results() Helper Function. Builds a manifest file used to contain extraction result metadata from h5 and yaml files.

Parameters:

- **loaded (list of dicts)** (*list of dicts containing loaded h5 data.*)
- **format (str)** (*filename format indicating the new name for the metadata files in the aggregate_results dir.*)
- **snake_case (bool)** (*whether to save the files using snake_case*)

Returns: **manifest (dict)**

Return type: dictionary of extraction metadata.

`moseq2_extract.helpers.data.`**`check_completion_status`**`(status_filename)`

Reads a results_00.yaml (status file) and checks whether the session has been fully extracted. Returns True if yes, and False if not and if the file doesn't exist.

Parameters: **status_filename (str)** (*path to results_00.yaml containing extraction status*)

Returns: **complete (bool)**

Return type: If True, data has been extracted to completion.

`moseq2_extract.helpers.data.`**`copy_manifest_results`**`(manifest, output_dir)`

Copies all consolidated manifest results to their respective output files.

Parameters:

- **manifest (dict)** (*manifest dictionary containing all extraction h5 metadata to save*)
- **output_dir (str)** (*path to directory where extraction results will be aggregated.*)

Returns:

Return type: None

`moseq2_extract.helpers.data.`**`create_extract_h5`**`(h5_file, acquisition_metadata, config_data, status_dict, scalars_attrs, nframes, roi, bground_im, first_frame, timestamps, **kwargs)`

This is a helper function for extract_wrapper(); handles writing the following metadata to an open results_00.h5 file: Acquisition metadata, extraction metadata, computed scalars, timestamps, and original frames/frames_mask.

Parameters:

- **h5_file (h5py.File object)** (*opened h5 file object to write to.*)
- **acquisition_metadata (dict)** (*Dictionary containing extracted session acquisition metadata.*)
- **config_data (dict)** (*dictionary object containing all required extraction parameters. (auto generated)*)
- **status_dict (dict)** (*dictionary that helps indicate if the session has been extracted fully.*)
- **scalars_attrs (dict)** (*dict of computed scalar attributes and descriptions to save.*)
- **nframes (int)** (*number of frames being recorded*)
- **roi (2d np.ndarray)** (*Computed 2D ROI Image.*)
- **bground_im (2d np.ndarray)** (*Computed 2D Background Image.*)
- **first_frame (2d np.ndarray)** (*Computed 2D First Frame Image.*)
- **timestamps (np.array)** (*Array of session timestamps.*)
- **extract (moseq2_extract.cli.extract function)** (*Used to preseve CLI state parameters in extraction h5.*)

Returns:

Return type: None

`moseq2_extract.helpers.data.`**`handle_extract_metadata`**(`input_file`, `dirname`)

Extracts metadata from input depth files, either raw or compressed. Locates metadata JSON file, and timestamps.txt file, then loads them into variables to be used to extract_wrapper.

Parameters:

- **input_file (str)** (*path to input file to extract*)
- **dirname (str)** (*path to directory where extraction files reside.*)

Returns: **acquisition_metadata (dict)** (*key-value pairs of JSON contents*) **timestamps (1D array)** (*list of loaded timestamps*) **alternate_correct (bool)** (*indicator for whether an alternate timestamp file was used*) **tar (bool)** (*indicator for whether the file is compressed.*)

`moseq2_extract.helpers.data.`**`load_extraction_meta_from_h5s`**(`to_load`, `snake_case=True`)

aggregate_results() Helper Function to load extraction metadata from h5 files.

Parameters:

- **to_load (list)** (*list of paths to h5 files.*)
- **snake_case (bool)** (*whether to save the files using snake_case*)

Returns: **loaded (list)**

Return type: list of loaded h5 dicts.

Helpers - Extract Module

Extraction-helper utilities. These functions are primarily called from inside the extract_wrapper() function.

`moseq2_extract.helpers.extract.`**`process_extract_batches`**(`input_file`, `config_data`, `bground_im`, `roi`, `frame_batches`, `first_frame_idx`, `str_els`, `output_mov_path`, `scalars=None`, `h5_file=None`, `**kwargs`)

Given an open h5 file, which is used to store extraction results, and some pre-computed input session data points such as the background, ROI, etc. Compute extracted frames and save them to h5 files and avi files. Called from extract_wrapper()

Parameters:
- **h5file (h5py.File)** (*opened h5 file to write extracted batches to*)
- **input_file (str)** (*path to depth file*)
- **config_data (dict)** (*dictionary containing extraction parameters (autogenerated)*)
- **bground_im (2d numpy array)** (*r x c, background image*)
- **roi (2d numpy array)** (*r x c, roi image*)
- **scalars (list)** (*list of keys to scalar attribute values*)
- **frame_batches (list)** (*list of batches of frames to serially process.*)
- **first_frame_idx (int)** (*index of starting frame.*)
- **str_els (dict)** (*dictionary containing OpenCV StructuringElements*)
- **output_mov_path (str)** (*path and filename of the output movie generated by the extraction*)

Returns: **config_data (dict)**

Return type: dictionary containing updated extraction validation parameter values

`moseq2_extract.helpers.extract.`**`run_local_extract`**(`to_extract`, `config_file`, `skip_extracted=False`)

Runs the extract command on given list of sessions to extract on a local platform. This function is meant for the GUI interface to utilize the moseq2-batch extract functionality.

Parameters:
- **to_extract (list)** (*list of paths to files to extract*)
- **config_file (str)** (*path to configuration file containing pre-configured extract and ROI*)
- **skip_extracted (bool)** (*Whether to skip already extracted session.*)

Returns:

Return type: None

`moseq2_extract.helpers.extract.`**`write_extracted_chunk_to_h5`**(`h5_file`, `results`, `config_data`, `scalars`, `frame_range`, `offset`)

Write extracted frames, frame masks, and scalars to an open h5 file.

Parameters:
- **h5_file (H5py.File)** (*open results_00 h5 file to save data in.*)
- **results (dict)** (*extraction results dict.*)
- **config_data (dict)** (*dictionary containing extraction parameters (autogenerated)*)
- **scalars (list)** (*list of keys to scalar attribute values*)
- **frame_range (range object)** (*current chunk frame range*)
- **offset (int)** (*frame offset*)

Helpers - Wrappers Module

Wrapper functions for all functionality afforded by MoSeq2-Extract. These functions perform all the data processing from start to finish, and are shared between the CLI and GUI.

`moseq2_extract.helpers.wrappers.`**`aggregate_extract_results_wrapper`**(`input_dir`, `format`, `output_dir`, `mouse_threshold=0.0`)

Copies all the h5, yaml and avi files generated from all successful extractions to a new directory to hold all the necessary data to continue down the MoSeq pipeline. Then generates an index file in the base directory/input_dir.

Parameters:
- **input_dir (str)** (*path to base directory containing all session folders*)
- **format (str)** (*string format for metadata to use as the new aggregated filename*)
- **output_dir (str)** (*name of the directory to create and store all results in*)
- **mouse_threshold (float)** (*threshold value of mean frame depth to include session frames*)

Returns: **indexpath (str)**

Return type: path to generated index file including all aggregated session information.

`moseq2_extract.helpers.wrappers.`**`convert_raw_to_avi_wrapper`**(`input_file`, `output_file`, `chunk_size`, `fps`, `delete`, `threads`)

Wrapper function used to convert/compress a raw depth file into

an avi file (with depth values) that is 8x smaller.

Parameters:
- **input_file (str)** (*Path to depth file to convert*)
- **output_file (str)** (*Path to avi output file*)
- **chunk_size (int)** (*Size of frame chunks to iteratively process*)
- **fps (int)** (*Frames per second.*)
- **delete (bool)** (*Delete the original depth file if True.*)
- **threads (int)** (*Number of threads used to encode video.*)

`moseq2_extract.helpers.wrappers.`**`copy_h5_metadata_to_yaml_wrapper`**(`input_dir`, `h5_metadata_path`)

Copy's user specified metadata from h5path to a yaml file.

Parameters:
- **input_dir (str)** (*path to directory containing h5 files*)
- **h5_metadata_path (str)** (*path within h5 to desired metadata to copy to yaml.*)

Returns:

Return type: None

`moseq2_extract.helpers.wrappers.`**`copy_slice_wrapper`**(`input_file`, `output_file`, `copy_slice`, `chunk_size`, `fps`, `delete`, `threads`)

Wrapper function to copy a segment of an input depth recording into a new video file.

Parameters:
- **input_file (str)** (*Path to depth file to read segment from*)
- **output_file (str)** (*Path to outputted video file with copied slice.*)
- **copy_slice (2-tuple)** (*Frame range to copy from input file.*)
- **chunk_size (int)** (*Size of frame chunks to iteratively process*)
- **fps (int)** (*Frames per second.*)
- **delete (bool)** (*Delete the original depth file if True.*)
- **threads (int)** (*Number of threads used to encode video.*)

`moseq2_extract.helpers.wrappers.`**`extract_wrapper`**(`input_file`, `output_dir`, `config_data`, `num_frames=None`, `skip=False`)

Wrapper function to run extract function for both GUI and CLI.

Parameters:
- **input_file (str)** (*path to depth file*)
- **output_dir (str)** (*path to directory to save results in.*)
- **config_data (dict)** (*dictionary containing extraction parameters.*)
- **num_frames (int)** (*number of frames to extract. All if None.*)
- **skip (bool)** (*indicates whether to skip file if already extracted*)
- **extract (function)** (*extraction function state (Only passed by CLI)*)

Returns: **output_dir (str)**

Return type: path to directory containing extraction (only if gui==True)

`moseq2_extract.helpers.wrappers.`**`get_roi_wrapper`**`(input_file, config_data, output_dir=None)`

Wrapper function to compute ROI given depth file.

Parameters:
- **input_file (str)** (*path to depth file.*)
- **config_data (dict)** (*dictionary of ROI extraction parameters.*)
- **output_dir (str)** (*path to desired directory to save results in.*)

Returns: **roi (2d array)** (*ROI image to plot in GUI*) **bground_im (2d array)** (*Background image to plot in GUI*) **first_frame (2d array)** (*First frame image to plot in GUI*)

moseq2_extract.io package

IO - Image Module

Image reading/writing functionality.

Specifically for tiff files containing backgrounds, ROIs, etc.

`moseq2_extract.io.image.`**`read_image`**`(filename, scale=True, scale_key='scale_factor')`

Load image data, possibly with scale factor…

Parameters:
- **filename (str)** (*path to file to write to.*)
- **scale (bool)** (*indicates whether to scale image*)
- **scale_key (str)** (*indicates scale factor.*)

Returns: **image (2d np array)**

Return type: loaded image

`moseq2_extract.io.image.`**`read_tiff_files`**`(input_dir)`

Reads ROI output results (Tiff files) located in the given input_directory

into array variables to be graphed in a jupyter notebook.

Parameters: **input_dir (str)** (*path to directory containing ROI files AKA tiff files.*)

Returns: **images (list)** (*list of 2d arrays read from each located tiff file.*) **filenames (list)** (*list of corresponding filenames to each read image.*)

`moseq2_extract.io.image.`**`write_image`**`(filename, image, scale=True, scale_factor=None, dtype='uint16', compress=0)`

Save image data, possibly with scale factor for easy display.

Parameters:
- **filename (str)** (*path to file to write to.*)
- **image (2d numpy array)** (*the (unscaled) 2-D image to save*)
- **scale (bool)** (flag to scale the image between the bounds of *dtype*)
- **scale_factor (int)** (*factor by which to scale image*)
- **dtype (str)** (*array data type*)
- **compress (int)** (*image compression level*)

Returns:

Return type: None

IO - Video Module

Video and video-metadata read/write functionality.

`moseq2_extract.io.video.`**`get_movie_info`**(`filename`, `frame_dims=(512, 424)`, `bit_depth=16`)
Returns dict of movie metadata.

Parameters:
- **filename (str)** (*path to video file*)
- **frame_dims (tuple)** (*video dimensions*)
- **bit_depth (int)** (*integer indicating data type encoding*)

Returns: **metadata (dict)**

Return type: dictionary containing video file metadata

`moseq2_extract.io.video.`**`get_raw_info`**(`filename`, `bit_depth=16`, `frame_dims=(512, 424)`)
Gets info from a raw data file with specified frame dimensions and bit depth.

Parameters:
- **filename (string)** (*name of raw data file*)
- **bit_depth (int)** (*bits per pixel (default: 16)*)
- **frame_dims (tuple)** (*wxh or hxw of each frame*)

Returns: **file_info (dict)**

Return type: dictionary containing depth file metadata

`moseq2_extract.io.video.`**`get_video_info`**(`filename`)
Get dimensions of data compressed using ffv1, along with duration via ffmpeg.

Parameters: **filename (string)** (*name of file*)

Returns: **(dict)**

Return type: dictionary containing video file metadata

`moseq2_extract.io.video.`**`load_movie_data`**(`filename`, `frames=None`, `frame_dims=(512, 424)`, `bit_depth=16`, `**kwargs`)
Parses file extension to check whether to read the data using ffmpeg (read_frames) or to read the frames directly from the file into a numpy array (read_frames_raw).

Parameters:
- **filename (str)** (*Path to file to read video from.*)
- **frames (int or list)** (*Frame indices to read in to output array.*)
- **frame_dims (tuple)** (*Video dimensions (nrows, ncols)*)
- **bit_depth (int)** (*Number of bits per pixel, corresponds to image resolution.*)
- **kwargs (dict)** (*Any additional parameters that could be required in read_frames_raw().*)

Returns: **frame_data (3D np.ndarray)**

Return type: Read video as numpy array. (nframes, nrows, ncols)

`moseq2_extract.io.video.`**`read_frames`**(`filename`, `frames=range(0, 0)`, `threads=6`, `fps=30`, `pixel_format='gray16le'`, `frame_size=None`, `slices=24`, `slicecrc=1`, `mapping=0`, `get_cmd=False`)

Reads in frames from the .nut/.avi file using a pipe from ffmpeg.

Parameters:

- **filename (str)** (*filename to get frames from*)
- **frames (list or 1d numpy array)** (*list of frames to grab*)
- **threads (int)** (*number of threads to use for decode*)
- **fps (int)** (*frame rate of camera in Hz*)
- **pixel_format (str)** (*ffmpeg pixel format of data*)
- **frame_size (str)** (*wxh frame size in pixels*)
- **slices (int)** (*number of slices to use for decode*)
- **slicecrc (int)** (*check integrity of slices*)
- **mapping (int)** (*ffmpeg channel mapping; "o:mapping"*)
- **get_cmd (bool)** (*indicates whether function should return ffmpeg command (instead of executing).*)

Returns: **video (3d numpy array)**

Return type: frames x h x w

`moseq2_extract.io.video.`**`read_frames_raw`**(`filename`, `frames=None`, `frame_dims=(512, 424)`, `bit_depth=16`, `dtype='<i2'`, `tar_object=None`)

Reads in data from raw binary file.

Parameters:

- **filename (string)** (*name of raw data file*)
- **frames (list or range)** (*frames to extract*)
- **frame_dims (tuple)** (*wxh of frames in pixels*)
- **bit_depth (int)** (*bits per pixel (default: 16)*)
- **tar_object (tarfile.TarFile)** (*TarFile object, used for loading data directly from tgz*)

Returns: **chunk (numpy ndarray)**

Return type: nframes x h x w

`moseq2_extract.io.video.`**`write_frames`**(`filename`, `frames`, `threads=6`, `fps=30`, `pixel_format='gray16le'`, `codec='ffv1'`, `close_pipe=True`, `pipe=None`, `slices=24`, `slicecrc=1`, `frame_size=None`, `get_cmd=False`)

Write frames to avi file using the ffv1 lossless encoder

Parameters:

- **filename (str)** (*path to file to write to.*)
- **frames (np.ndarray)** (*frames to write*)
- **threads (int)** (*number of threads to write video*)
- **fps (int)** (*frames per second*)
- **pixel_format (str)** (*format video color scheme*)
- **codec (str)** (*ffmpeg encoding-writer method to use*)
- **close_pipe (bool)** (*indicates to close the open pipe to video when done writing.*)
- **pipe (subProcess.Pipe)** (*pipe to currently open video file.*)
- **slices (int)** (*number of frame slices to write at a time.*)
- **slicecrc (int)** (*check integrity of slices*)
- **frame_size (tuple)** (*shape/dimensions of image.*)
- **get_cmd (bool)** (*indicates whether function should return ffmpeg command (instead of executing)*)

Returns: **pipe (subProcess.Pipe)**

Return type: indicates whether video writing is complete.

`moseq2_extract.io.video.`**`write_frames_preview`**(`filename`, `frames=array([]`, `dtype=float64)`, `threads=6`, `fps=30`, `pixel_format='rgb24'`, `codec='h264'`, `slices=24`, `slicecrc=1`, `frame_size=None`, `depth_min=0`, `depth_max=80`, `get_cmd=False`, `cmap='jet'`, `pipe=None`, `close_pipe=True`, `frame_range=None`, `progress_bar=False`)

Simple command to pipe frames to an ffv1 file. Writes out a false-colored mp4 video.

Parameters:

- **filename (str)** (*path to file to write to.*)
- **frames (np.ndarray)** (*frames to write*)
- **threads (int)** (*number of threads to write video*)
- **fps (int)** (*frames per second*)
- **pixel_format (str)** (*format video color scheme*)
- **codec (str)** (*ffmpeg encoding-writer method to use*)
- **slices (int)** (*number of frame slices to write at a time.*)
- **slicecrc (int)** (*check integrity of slices*)
- **frame_size (tuple)** (*shape/dimensions of image.*)
- **depth_min (int)** (*minimum mouse depth from floor in (mm)*)
- **depth_max (int)** (*maximum mouse depth from floor in (mm)*)
- **get_cmd (bool)** (*indicates whether function should return ffmpeg command (instead of executing)*)
- **cmap (str)** (*color map to use.*)
- **pipe (subProcess.Pipe)** (*pipe to currently open video file.*)
- **close_pipe (bool)** (*indicates to close the open pipe to video when done writing.*)
- **frame_range (range())** (*frame indices to write on video*)

Returns: **pipe (subProcess.Pipe)**

Return type: indicates whether video writing is complete.

Index

- **genindex**

Index

Symbols


-erode-iterations <erode_iterations>
moseq2-extract-extract command line option
moseq2-extract-find-roi command line option
--flip-classifier <flip_classifier>
moseq2-extract-extract command line option
assifier-smoothing <flip_classifier_smoothing>
moseq2-extract-extract command line option
--format <format>
moseq2-extract-aggregate-results command line option
--fps <fps>
moseq2-extract-convert-raw-to-avi command line option
moseq2-extract-copy-slice command line option
moseq2-extract-extract command line option
--frame-dtype <frame_dtype>
moseq2-extract-extract command line option
--frame-trim <frame_trim>
moseq2-extract-extract command line option
--input-dir <input_dir>
moseq2-extract-aggregate-results command line option
moseq2-extract-generate-index command line option
--max-height <max_height>
moseq2-extract-extract command line option
--min-height <min_height>
moseq2-extract-extract command line option
el-smoothing-clips <model_smoothing_clips>
moseq2-extract-extract command line option
ouse-threshold <mouse_threshold>
moseq2-extract-aggregate-results command line option
--noise-tolerance <noise_tolerance>
moseq2-extract-extract command line option
moseq2-extract-find-roi command line option
--num-frames <num_frames>
moseq2-extract-extract command line option
--output-dir <output_dir>
moseq2-extract-aggregate-results command line option
moseq2-extract-download-flip-file command line option
moseq2-extract-extract command line option
moseq2-extract-find-roi command line option
--output-file <output_file>
moseq2-extract-convert-raw-to-avi command line option
moseq2-extract-copy-slice command line option
moseq2-extract-extract command line option
moseq2-extract-generate-config command line option
moseq2-extract-generate-index command line option
--progress-bar
moseq2-extract-extract command line option
moseq2-extract-find-roi command line option
--skip-completed
moseq2-extract-extract command line option
--spatial-filter-size <spatial_filter_size>
moseq2-extract-ex command line opt
--tail-filter-iters <tail_filter_iters>
moseq2-extract-extra command line option
--tail-filter-shape <tail_filter_shape>
moseq2-extract-extra command line option
--tail-filter-size <tail_filter_size>
moseq2-extract-extrac command line option
--temporal-filter-size <temporal_filter_size>
moseq2-extract-c command line op
--threads <threads>
moseq2-extract-convert-raw-to-avi command line option
moseq2-extract-copy-slice command line option
moseq2-extract-extract command line option
--tracking-model-init <tracking_model_init>
moseq2-extract-ex command line opt
--tracking-model-ll-clip <tracking_model_ll_clip>
moseq2-extra command line
--tracking-model-ll-threshold <tracking_model_ll_threshold>
moseq2 comma
--tracking-model-mask-threshold <tracking_model_mask_threshold>
mos com
--tracking-model-segment <tracking_model_segment>
moseq2-extra command line
--use-cc <use_cc>
moseq2-extract-extract command line option
--use-plane-bground
moseq2-extract-extract command line option
moseq2-extract-find-roi command line option
--use-tracking-model <use_tracking_model>
moseq2-extract-ext command line optio
--version
moseq2-extract command line option
--write-movie <write_movie>
moseq2-extract-extract command line option

INPUT_FILE

moseq2-extract-copy-slice command line option

--chunk-size <chunk_size>

--copy-slice <copy_slice>

--delete

--fps <fps>

--output-file <output_file>

--threads <threads>

-b

-c

-o

-t

INPUT_FILE

moseq2-extract-download-flip-file command line option

--output-dir <output_dir>

CONFIG_FILE

moseq2-extract-extract command line option

--angle-hampel-sig <angle_hampel_sig>

--angle-hampel-span <angle_hampel_span>

--bg-roi-depth-range <bg_roi_depth_range>

--bg-roi-dilate <bg_roi_dilate>

--bg-roi-erode <bg_roi_erode>

--bg-roi-fill-holes <bg_roi_fill_holes>

--bg-roi-gradient-filter <bg_roi_gradient_filter>

--bg-roi-gradient-kernel <bg_roi_gradient_kernel>

--bg-roi-gradient-threshold <bg_roi_gradient_threshold>

--bg-roi-index <bg_roi_index>

--bg-roi-shape <bg_roi_shape>

--bg-roi-weights <bg_roi_weights>

--bg-sort-roi-by-position <bg_sort_roi_by_position>

--bg-sort-roi-by-position-max-rois <bg_sort_roi_by_position_max_rois>

--cable-filter-iters <cable_filter_iters>

--cable-filter-shape <cable_filter_shape>

--cable-filter-size <cable_filter_size>

--camera-type <camera_type>

--centroid-hampel-sig <centroid_hampel_sig>

--centroid-hampel-span <centroid_hampel_span>

--chunk-overlap <chunk_overlap>

--chunk-size <chunk_size>

--compress <compress>

--compress-chunk-size <compress_chunk_size>

--compress-threads <compress_threads>

--compute-raw-scalars

--config-file <config_file>

--crop-size <crop_size>

--delete

--detected-true-depth <detected_true_depth>

--dilate-iterations <dilate_iterations>

--erode-iterations <erode_iterations>

--flip-classifier <flip_classifier>

--flip-classifier-smoothing <flip_classifier_smoothing>

--fps <fps>

--frame-dtype <frame_dtype>

--frame-trim <frame_trim>

--max-height <max_height>

--min-height <min_height>

--model-smoothing-clips <model_smoothing_clips>

--noise-tolerance <noise_tolerance>

--num-frames <num_frames>

--output-dir <output_dir>

--output-file <output_file>

--progress-bar

--skip-completed

--spatial-filter-size <spatial_filter_size>

--tail-filter-iters <tail_filter_iters>

--tail-filter-shape <tail_filter_shape>

--tail-filter-size <tail_filter_size>

--temporal-filter-size <temporal_filter_size>

--threads <threads>

--tracking-model-init <tracking_model_init>

--tracking-model-ll-clip <tracking_model_ll_clip>

--tracking-model-ll-threshold <tracking_model_ll_threshold>

--tracking-model-mask-threshold <tracking_model_mask_threshold>

--tracking-model-segment <tracking_model_segment>

--use-cc <use_cc>

--use-plane-bground

--use-tracking-model <use_tracking_model>

--write-movie <write_movie>

-b

-c

Python Module Index

m