Python Documentation

version

April 14, 2020

Contents

Welcome to moseq2-viz's documentation!

moseq2-viz

moseq2-viz.moseq2_viz package

Subpackages

moseq2-viz.moseq2_viz.helpers package

Helpers - Wrappers Module

`moseq2_viz.helpers.wrappers.`**`add_group_wrapper`** (index_file, config_data)
Given a pre-specified key and value, the index file will be updated with the respective found keys and values.

Parameters:
- **index_file (str)** (*path to index file*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)

Returns:

Return type: None

`moseq2_viz.helpers.wrappers.`**`copy_h5_metadata_to_yaml_wrapper`** (input_dir, h5_metadata_path)
Copy h5 metadata dictionary contents into the respective file's yaml file.

Parameters:
- **input_dir (str)** (*path to directory that contains h5 files.*)
- **h5_metadata_path (str)** (*path to data within h5 file to update yaml with.*)

Returns:

Return type: None

`moseq2_viz.helpers.wrappers.`**`make_crowd_movies_wrapper`** (index_file, model_path, config_data, output_dir)
Wrapper function to create crowd movie videos and write them to individual files depicting respective syllable labels.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_path (str)** (*path to trained model.*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)
- **output_dir (str)** (*directory to store crowd movies in.*)

Returns:

Return type: None

`moseq2_viz.helpers.wrappers.`**`plot_scalar_summary_wrapper`** (index_file, output_file, gui=False)
Wrapper function that plots scalar summary graphs.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*path to save graphs.*)
- **gui (bool)** (*indicate whether GUI is plotting the graphs*)

Returns: **scalar_df (pandas DataFrame)** (*df containing scalar data per session uuid.*) *(Only accessible through GUI API)*

`moseq2_viz.helpers.wrappers.`**`plot_syllable_durations_wrapper`** (index_file, model_fit, groups, count, max_syllable, output_file, ylim=None, gui=False)
Wrapper function that plots syllable durations.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to trained model.*)
- **groups (tuple)** (*list of groups to separately graph data for.*)
- **count (str)** (*method to compute usages 'usage' or 'frames'.*)
- **max_syllable (int)** (*maximum number of syllables to plot.*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **ylim (float)** (*y-axis limit in the outputted graph.*)
- **gui (bool)** (*indicate whether GUI is plotting the graphs.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_syllable_usages_wrapper`** (index_file, model_fit, max_syllable, sort, count, group, output_file, gui=False)

Wrapper function to plot syllable usages.

Parameters:
- **index_file (str)** (*path to index file.*)
- **model_fit (str)** (*path to trained model file.*)
- **max_syllable (int)** (*maximum number of syllables to plot.*)
- **sort (bool)** (*sort syllables by usage.*)
- **count (str)** (*method to compute usages 'usage' or 'frames'.*)
- **group (tuple)** (*tuple of groups to separately model usages.*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **gui (bool)** (*indicate whether GUI is plotting the graphs.*)

Returns: **plt (pyplot figure)**

Return type: graph to show in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_transition_graph_wrapper`** (index_file, model_fit, config_data, output_file, gui=False)

Wrapper function to plot transition graphs.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to trained model.*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **gui (bool)** (*indicate whether GUI is plotting the graphs.*)

Returns: **plt (pyplot figure)**

Return type: graph to show in Jupyter Notebook.

moseq2-viz.moseq2_viz.info package

Info - Utilities Module

`moseq2_viz.info.util.`**`entropy`** (labels, truncate_syllable=40, smoothing=1.0, relabel_by='usage')

Computes syllable usage entropy, base 2.

Parameters:
- **labels (np.ndarray)** (*array of predicted syllable labels*)
- **truncate_syllable (int)** (*truncate list of relabeled syllables*)
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **relabel_by (str)** (*mode to relabel predicted labels.*)

Returns: **ent (list)**

Return type: list of entropy values for each syllable label.

`moseq2_viz.info.util.`**`entropy_rate`** (labels, truncate_syllable=40, normalize='bigram', smoothing=1.0, tm_smoothing=1.0, relabel_by='usage')

Computes entropy rate, base 2 using provided syllable labels. If syllable labels have not been re-labeled by usage, this function will do so.

Parameters:
- **labels (list or np.ndarray)** (*a list of label arrays, where each entry in the list*) – is an array of labels for one subject.
- **truncate_syllable (int)** (*the number of labels to keep for this calculation*)
- **normalize (str)** (*the type of transition matrix normalization to perform. Options*) – are: 'bigram', 'rows', or 'columns'.
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **tm_smoothing (float)** (*a constant added to label transtition counts before*) – normalization.
- **relabel_by (str)** (*how to re-order labels. Options are: 'usage' and 'frames'.*)

Returns: **ent (list)**

Return type: list of entropy rates per syllable label

moseq2-viz.moseq2_viz.io package

Viz - IO Video Module

`moseq2_viz.io.video.`**`write_crowd_movies`** (sorted_index, config_data, filename_format, vid_parameters, clean_params, ordering, labels, label_uuids, max_syllable, max_examples, output_dir)

Creates syllable slices for crowd movies and writes them to files.

Parameters:
- **sorted_index (dict)** (*dictionary of sorted index data.*)
- **config_data (dict)** (*dictionary of visualization parameters.*)
- **filename_format (str)** (*string format that denotes the saved crowd movie file names.*)
- **vid_parameters (dict)** (*dictionary of video writing parameters*)
- **clean_params (dict)** (*dictionary of image filtering parameters*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)
- **labels (numpy ndarray)** (*list of syllable usages*)
- **label_uuids (list)** (*list of session uuids each series of labels belongs to.*)
- **max_syllable (int)** (*maximum number of syllables to create movies for*)
- **max_examples (int)** (*maximum number of mice to include in a crowd movie*)
- **output_dir (str)** (*path directory where all the movies are written.*)

Returns:

Return type: None

`moseq2_viz.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', text=None, text_scale=1, text_thickness=2, pipe=None, close_pipe=True, progress_bar=True)

Writes out a false-colored mp4 video. [Duplicate from moseq2-extract]

Parameters:

- **filename (str)**
- **frames (3D numpy array)** (*num_frames * r * c*)
- **threads (int)** (*number of threads to write file*)
- **fps (int)** (*frames per second*)
- **pixel_format (str)** (*ffmpeg image formatting flag.*)
- **codec (str)** (*ffmpeg image encoding flag.*)
- **slices (int)** (*number of slices per thread.*)
- **slicecrc (int)** (*check integrity of slices.*)
- **frame_size (tuple)** (*image dimensions*)
- **depth_min (int)** (*minimum mouse distance from bucket floor*)
- **depth_max (int)** (*maximum mouse distance from bucket floor*)
- **get_cmd (bool)** (*return ffmpeg command instead of executing the command in python.*)
- **cmap (str)** (*color map selection.*)
- **text (range(num_frames)** (*display frame number in output video.*)
- **text_scale (int)** (*text size.*)
- **text_thickness (int)** (*text thickness.*)
- **pipe (subProcess.Pipe object)** (*if not None, indicates that there are more frames to be written.*)
- **close_pipe (bool)** (*indicates whether video is done writing, and to close pipe to file-stream.*)
- **progress_bar (bool)** (*display progress bar.*)

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

Return type: if there are more slices/chunks to write to, otherwise None.

moseq2-viz.moseq2_viz.model package

Model - Dist Module

`moseq2_viz.model.dist.`**`get_behavioral_distance`** (index, model_file, whiten='all', distances=['ar[init]', 'scalars'], max_syllable=None, resample_idx=- 1, dist_options={}, sort_labels_by_usage=True, count='usage')

`moseq2_viz.model.dist.`**`get_behavioral_distance_ar`** (ar_mat, init_point=None, sim_points=10, max_syllable=40, dist='correlation', parallel=False)

`moseq2_viz.model.dist.`**`get_init_points`** (pca_scores, model_labels, max_syllable=40, nlags=3, npcs=10)

`moseq2_viz.model.dist.`**`reformat_dtw_distances`** (full_mat, nsyllables, rescale=True)

Model - Label Utilities Module

`moseq2_viz.model.label_util.`**`syll_duration`** (labels: numpy.ndarray) → numpy.ndarray

Computes the duration of each syllable.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **durations (np.ndarray)**

Return type: array of syllable durations.

moseq2_viz.model.label_util.**syll_id** (labels: numpy.ndarray) → numpy.ndarray
Returns the syllable label at each syllable transition.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)
Returns: **labels[onsets] (np.ndarray)**
Return type: an array of compressed labels.

moseq2_viz.model.label_util.**syll_onset** (labels: numpy.ndarray) → numpy.ndarray
Finds indices of syllable onsets.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)
Returns: **indices (np.ndarray)**
Return type: an array of indices denoting the beginning of each syllables.

moseq2_viz.model.label_util.**to_df** (labels, uuid) → pandas.core.frame.DataFrame
Convert labels numpy.ndarray to pandas.DataFrame

Parameters:
- **labels (np.ndarray)** (*array of syllable labels for a mouse.*)
- **uuid (list)** (*list of session uuids representing each series of labels.*)

Returns: **df (pd.DataFrame)**
Return type: DataFrame of syllables, durations, onsets, and session uuids.

Model - Utilities Module

moseq2_viz.model.util.**calculate_label_durations** (label_arr: Union[dict, numpy.ndarray]) → Union[dict, numpy.ndarray]
Calculates syllable label durations.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)
Returns: **np.diffs(inds) (np.ndarray)**
Return type: list of durations for each syllable in respective label order.

moseq2_viz.model.util.**calculate_syllable_usage** (labels: Union[dict, pandas.core.frame.DataFrame])
Calculates a dictionary of uuid to syllable usage key-values pairs.

Parameters: **label_arr (dict or pd.DataFrame)** (*list or DataFrame of predicted syllable labels.*)
Returns: **(dict)**
Return type: dictionary of syllable usage probabilities.

moseq2_viz.model.util.**compress_label_sequence** (label_arr: Union[dict, numpy.ndarray]) → numpy.ndarray
Removes repeating values from a label sequence. It assumes the first label is '-5', which is unused for behavioral analysis, and removes it.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)
Returns: **label_arr[inds] (dict or np.ndarray)**
Return type: the compressed version of the label arrays.

moseq2_viz.model.util.**find_label_transitions** (label_arr: Union[dict, numpy.ndarray]) → numpy.ndarray
Finds indices where a label transitions into another label. This function is cached to increase performance because it is called frequently.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)
Returns: **inds (np.ndarray)**
Return type: Array of syllable transition indices for each session uuid.

moseq2_viz.model.util.**get_mouse_syllable_slices** (syllable: int, labels: numpy.ndarray) → Iterator[slice]
Return a generator containing slices of *syllable* indices for a mouse.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)

Returns: **slices (list)**

Return type: list of syllable label slices; e.g. [slice(3, 6, None), slice(9, 12, None)]

`moseq2_viz.model.util.`**`get_syllable_slices`**

Get the indices that correspond to a specific syllable for each animal in a modeling run.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **label_uuids (list)** (*list of uuid keys corresponding to each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **trim_nans (bool)** (*flag to use the pca scores file for removing time points that contain NaNs.*)
- **Only use if you have not already trimmed NaNs previously (i.e. in `scalars_to_dataframe`).**

Returns: **syllable_slices (list)** (a list of indices for *syllable* in the *labels* array. Each item in the list) *is a tuple of (slice, uuid, h5_file).*

`moseq2_viz.model.util.`**`get_syllable_statistics`** (data, fill_value=- 5, max_syllable=100, count='usage')

Compute the syllable statistics from a set of model labels

Parameters:
- **data (list of np.array of ints)** (*labels loaded from a model fit.*)
- **fill_value (int)** (*lagged label values in the labels array to remove.*)
- **max_syllable (int)** (*maximum syllable to consider.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames).*)

Returns: **usages (defaultdict)** (*default dictionary of usages*) **durations (defaultdict)** (*default dictionary of durations*)

`moseq2_viz.model.util.`**`get_transition_matrix`** (labels, max_syllable=100, normalize='bigram', smoothing=0.0, combine=False, disable_output=False) → list

Compute the transition matrix from a set of model labels.

Parameters:
- **labels (list of np.array of ints)** (*labels loaded from a model fit*)
- **max_syllable (int)** (*maximum syllable number to consider*)
- **normalize (str)** (*how to normalize transition matrix, 'bigram' or 'rows' or 'columns'*)
- **smoothing (float)** (*constant to add to transition_matrix pre-normalization to smooth counts*)
- **combine (bool)** (*compute a separate transition matrix for each element (False)*)
- **or combine across all arrays in the list (True)**
- **disable_output (bool)** (*verbosity*)

Returns: **transition_matrix (list)** – from syllable i (row) to syllable j (column)

Return type: list of 2d np.arrays that represent the transitions

`moseq2_viz.model.util.`**`labels_to_changepoints`** (labels, fs=30.0)

Compute the transition matrix from a set of model labels.

Parameters:
- **labels (list of np.array of ints)** (*labels loaded from a model fit.*)
- **fs (float)** (*sampling rate of camera.*)

Returns: **cp_dist (list of np.array of floats)**

Return type: list of block durations per element in labels list.

`moseq2_viz.model.util.`**`merge_models`** (model_dir, ext)
Merges model states by using the Hungarian Algorithm: a minimum distance state matching algorithm. User inputs a directory containing models to merge, (and the name of the latest-trained model) to match other model states to.

Parameters:
- **model_dir (str)** (*path to directory containing all the models to merge.*)
- **ext (str)** (*model extension to search for.*)

Returns: **model_data (dict)** (*a dictionary containing all the new*) *keys and state-matched labels.*

`moseq2_viz.model.util.`**`normalize_pcs`** (pca_scores: dict, method: str = 'z') → dict
Normalize PC scores. Options are: demean, zscore, ind-zscore. demean: subtract the mean from each score.

Parameters:
- **pca_scores (dict)** (*dict of uuid to PC-scores key-value pairs.*)
- **method (str)** (*the type of normalization to perform (demean, zscore, ind-zscore)*)

Returns: **norm_scores (dict)**

Return type: a dictionary of normalized PC scores.

`moseq2_viz.model.util.`**`parse_batch_modeling`** (filename)
Reads model parameter scan training results into a single dictionary.

Parameters: **filename (str)** (*path to h5 manifest file containing all the model results.*)

Returns: **results_dict (dict)** (*dictionary containing each model's training results,*) *concatenated into a single list. Maintaining the original structure as though it was a single model's results.*

`moseq2_viz.model.util.`**`parse_model_results`** (model_obj, restart_idx=0, resample_idx=- 1, map_uuid_to_keys: bool = False, sort_labels_by_usage: bool = False, count: str = 'usage') → dict
Reads model file and returns dictionary containing modeled results and some metadata.

Parameters:
- **model_obj (str or results returned from joblib.load)** (*path to the model fit or a loaded model fit*)
- **restart_idx (int)** (*Select which model restart to load. (Only change for models with multiple restarts used)*)
- **resample_idx (int)** (*Indicates the parsing method according to the shape of the labels array.*)
- **map_uuid_to_keys (bool)** (*for labels, make a dictionary where each key, value pair*)
- **contains the uuid and the labels for that session.**
- **sort_labels_by_usage (bool)** (*sort labels by their usages.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage),*)
- **or number of frames (frames).**

Returns: **output_dict (dict)**

Return type: dictionary with labels and model parameters

`moseq2_viz.model.util.`**`relabel_by_usage`** (labels, fill_value=- 5, count='usage')
Resort model labels by their usages.

Parameters:
- **labels (list of np.array of ints)** (*labels loaded from a model fit*)
- **fill_value (int)** (*value prepended to modeling results to account for nlags*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames)*)

Returns: **labels (list of np.array of ints)** (*labels resorted by usage*) **sorting (list)** (*the new label sorting. The index corresponds to the new label,*) *while the value corresponds to the old label.*

moseq2_viz.model.util.**results_to_dataframe** (model_dict, index_dict, sort=False, count='usage', normalize=True, max_syllable=40, include_meta=['SessionName', 'SubjectName', 'StartTime'])
Converts inputted model dictionary to DataFrame with user specified metadata columns.

Parameters:
- **model_dict (dict)** (*loaded model results dictionary.*)
- **index_dict (dict)** (*loaded index file dictionary*)
- **sort (bool)** (*indicate whether to relabel syllables by usage.*)
- **count (str)** (*indicate what to sort the labels by: usage, or frames*)
- **normalize (bool)** (*unused.*)
- **max_syllable (int)** (*maximum number of syllables to include in dataframe.*)
- **include_meta (list)** (*mouse metadata to include in dataframe.*)

Returns: **df (pd.DataFrame)** (*DataFrame containing model results and metadata.*) **df_dict (dict)** (*dictionary representation of the DataFrame.*)

moseq2_viz.model.util.**retrieve_pcs_from_slices** (slices, pca_scores, max_dur=60, min_dur=3, max_samples=100, npcs=10, subsampling=None, remove_offset=False, **kwargs)
Subsample Principal components from syllable slices :Parameters: * **slices (np.ndarray)** (*syllable slice or subarray to compute PCs for*)

- **pca_scores (np.ndarray)** (*PC scores for respective session.*)
- **max_dur (int)** (*maximum slice length.*)
- **min_dur (int)** (*minimum slice length.*)
- **max_samples (int)** (*maximum number of samples to slices to retrieve.*)
- **npcs (int)** (*number of pcs to use.*)
- **subsampling (int)** (*number of neighboring PCs to subsample from.*)
- **remove_offset (bool)** (*indicate whether to remove lag values.*)
- **kwargs (dict)** (*unused.*)

Returns: **syllable_matrix (np.ndarray)**

Return type: 3D matrix of subsampled PC projected syllable slices.

moseq2_viz.model.util.**simulate_ar_trajectory** (ar_mat, init_points=None, sim_points=100)
Simulate auto-regressive trajectory matrices from optionally randomly projected initalized points.

Parameters:
- **ar_mat (np.ndarray)** (*numpy array representing the autoregressive matrix of each model state.*)
- **init_points (np.ndarray)** (*pre-initialzed array of the same shape as the ar-matrices.*)
- **sim_points (int)** (*number of trajectories to simulate.*)

Returns: **sim_mat[nlags**

Return type:] simulated AR matrices excluding lagged values.

moseq2_viz.model.util.**sort_batch_results** (data, averaging=True, filenames=None, **kwargs)
Sort modeling results from batch/parameter scan.

Parameters:
- **data (np.ndarray)** (*model AR-matrices.*)
- **averaging (bool)** (*return an average of all the model AR-matrices.*)
- **filenames (list)** (*list of paths to fit models.*)
- **kwargs (dict)** (*dict of extra keyword arguments.*)

Returns: *new_matrix (np.ndarray) param_dict (dict)* **filename_index (list)** (*list of filenames associated with each model.*)

moseq2_viz.model.util.**syllable_slices_from_dict**
Reads dictionary of syllable labels, and returning a dict of syllable slices.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **filter_nans (bool)** (*replace NaN values with 0.*)

Returns: **vals (dict)**

Return type: key-value pairs of syllable slices per session uuid.

moseq2_viz.model.util.**whiten_pcs** (pca_scores, method='all', center=True)
Whiten PC scores using Cholesky whitening

Args:

pca_scores (dict): dictionary where values are pca_scores (2d np arrays) method (str): 'all' to whiten using the covariance estimated from all keys, or 'each' to whiten each separately center (bool): whether or not to center the data

Returns:

whitened_scores (dict): dictionary of whitened pc scores

Examples:

Load in pca_scores and whiten

>> from moseq2_viz.util import h5_to_dict >> from moseq2_viz.model.util import whiten_pcs >> pca_scores = h5_to_dict('pca_scores.h5', '/scores') >> whitened_scores = whiten_pcs(pca_scores, method='all')

moseq2-viz.moseq2_viz.scalars package

Scalars - Utilities Module

moseq2_viz.scalars.util.**convert_legacy_scalars** (old_features, force: bool = False, true_depth: float = 673.1) → dict
Converts scalars in the legacy format to the new format, with explicit units.

Parameters:
- **old_features (str, h5 group, or dictionary of scalars)** (*filename, h5 group,*)
- **or dictionary of scalar values.**
- **force (bool)** (*force the conversion of centroid_[xy]_px into mm.*)
- **true_depth (float)** (*true depth of the floor relative to the camera (673.1 mm by default)*)

Returns: **features (dict)**

Return type: dictionary of scalar values

moseq2_viz.scalars.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 coordinate lists.*)
- **resolution (tuple)** (*video frame size.*)
- **field_of_view (tuple)** (*camera focal lengths.*)
- **true_depth (float)** (*detected distance between depth camera and bucket floor.*)

Returns: **new_coords (list)**

Return type: list of same [x,y] coordinates in millimeters.

moseq2_viz.scalars.util.**find_and_load_feedback** (extract_path, input_path)

moseq2_viz.scalars.util.**generate_empty_feature_dict** (nframes) → dict
Generates a dict of numpy array of zeros of length nframes for each feature parameter.

Parameters: **nframes (int)** (*length of video*)
Returns: **(dict)**
Return type: dictionary feature to numpy 0 arrays of length nframes key-value pairs.

moseq2_viz.scalars.util.**get_scalar_map** (index, fill_nans=True, force_conversion=False)
Returns a dictionary of scalar values loaded from an index dictionary.

Parameters:
- **index (dict)** (*dictionary of index file contents.*)
- **fill_nans (bool)** (*indicate whether to replace NaN values with 0.*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)

Returns: **scalar_map (dict)**
Return type: dictionary of all the scalar values acquired after extraction.

moseq2_viz.scalars.util.**get_scalar_triggered_average** (scalar_map, model_labels, max_syllable=40, nlags=20, include_keys=['velocity_2d_mm', 'velocity_3d_mm', 'width_mm', 'length_mm', 'height_ave_mm', 'angle'], zscore=False)
Get averages of selected scalar keys for each syllable.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **model_labels (dict)** (*dictionary of uuid to syllable label array pairs.*)
- **max_syllable (int)** (*maximum number of syllables to use.*)
- **nlags (int)** (*number of lags to use when averaging over a series of PCs.*)
- **include_keys (list)** (*list of scalar values to load averages of.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

Returns: **syll_average (dict)**
Return type: dictionary of scalars for each syllable sequence.

moseq2_viz.scalars.util.**is_legacy** (features: dict)
Checks a dictionary of features to see if they correspond with an older version of moseq.

Parameters: **features**
Returns: **(bool)**
Return type: true if the dict is from an old dataset

moseq2_viz.scalars.util.**nanzscore** (data)
Z-score numpy array that may contain NaN values.

Parameters: **data (np.ndarray)** (*array of scalar values.*)
Returns: **data (np.ndarray)**
Return type: z-scored data.

moseq2_viz.scalars.util.**process_scalars** (scalar_map: dict, include_keys: list, zscore: bool = False) → dict
Fill NaNs and possibly zscore scalar values.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **include_keys (list)** (*scalar keys to process.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

moseq2_viz.scalars.util.**remove_nans_from_labels** (idx, labels)
Removes the frames from *labels* where *idx* has NaNs in it.

Parameters:
- **idx (list)** (*indices to remove NaN values at.*)
- **labels (list)** (*label list containing NaN values.*)

Returns: **(list)**

Return type: label list excluding NaN values at given indices

`moseq2_viz.scalars.util.`**`scalars_to_dataframe`** (index: dict, include_keys: list = ['SessionName', 'SubjectName', 'StartTime'], include_model=None, disable_output=False, include_pcs=False, npcs=10, include_feedback=None, force_conversion=True, include_labels=False)

Generates a dataframe containing scalar values over the course of a recording session. If a model string is included, then return only animals that were included in the model Called to sort scalar metadata information when graphing in plot-scalar-summary.

Parameters:
- **index (dict)** (a sorted_index generated by *parse_index* or *get_sorted_index*)
- **include_keys (list)** (*a list of other moseq related keys to include in the dataframe*)
- **include_model (str)** (*path to an existing moseq model*)
- **disable_output (bool)** (*indicate whether to show tqdm output.*)
- **include_pcs (bool)** (*UNUSED*)
- **npcs (int)** (*UNUSED*)
- **include_feedback (bool)** (*indicate whether to include timestamp data*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)
- **include_labels (bool)** (*UNUSED*)

Returns: **scalar_df (pandas DataFrame)**

Return type: DataFrame of loaded scalar values with their selected metadata.

`moseq2_viz.scalars.util.`**`star_valmap`** (func, d)

moseq2-viz.moseq2_viz.tests package

Subpackages

moseq2-viz.moseq2_viz.tests.integration_tests package

Integration Tests - CLI Tests

class `moseq2_viz.tests.integration_tests.test_cli.`**`TestCLI`** (methodName='runTest')

Bases: **`unittest.case.TestCase`**

`test_add_group` **()**

`test_copy_h5_metadata_to_yaml` **()**

`test_make_crowd_movies` **()**

`test_plot_scalar_summary` **()**

`test_plot_transition_graph` **()**

`test_plot_usages` **()**

Integration Tests - GUI Tests

class `moseq2_viz.tests.integration_tests.test_gui.`**`TestGUI`** (methodName='runTest')

Bases: **unittest.case.TestCase**

test_add_group_by_session ()

test_add_group_by_subject ()

test_copy_h5_metadata_to_yaml_command ()

test_get_groups_command ()

test_make_crowd_movies_command ()

test_plot_scalar_summary_command ()

test_plot_syllable_durations_command ()

test_plot_transition_graph_command ()

test_plot_usages_command ()

Integration Tests - Visualization Tests

class moseq2_viz.tests.integration_tests.test_viz.**TestViz** (methodName='runTest')
Bases: **unittest.case.TestCase**

test_clean_frames ()

test_convert_ebunch_to_graph ()

test_convert_transition_matrix_to_ebunch ()

test_duration_plot ()

test_floatRgb ()

test_graph_transition_matrix ()

test_make_crowd_matrix ()

test_position_plot ()

test_scalar_plot ()

test_usage_plot ()

moseq2_viz.tests.integration_tests.test_viz.**get_ebunch** (max_syllable=40, ret_trans=False)

moseq2_viz.tests.integration_tests.test_viz.**get_fake_movie** ()

moseq2-viz.moseq2_viz.tests.unit_tests package

Unit Tests - Info Utilities Tests Module

class moseq2_viz.tests.unit_tests.test_info_utils.**TestInfoUtils** (methodName='runTest')
Bases: **unittest.case.TestCase**

test_entropy ()

test_entropy_rate ()

Unit Tests - IO Video Tests Module

class moseq2_viz.tests.unit_tests.test_io_video.**TestIOVideo** (methodName='runTest')
Bases: **unittest.case.TestCase**

test_write_crowd_movies ()

test_write_frames_preview ()

Unit Tests - Model Dist Tests Module

class moseq2_viz.tests.unit_tests.test_model_dist.**TestModeLDists** (methodName='runTest')
Bases: **unittest.case.TestCase**

test_get_behavioral_distance ()

test_get_behavioral_distance_ar ()

test_get_init_points ()

test_reformat_dtw_distance ()

Unit Tests - Model Utilities Tests Module

class moseq2_viz.tests.unit_tests.test_model_util.**TestModelUtils** (methodName='runTest')
Bases: **unittest.case.TestCase**

test_calculate_label_durations ()

test_calculate_syllable_usage ()

test_compress_label_sequence ()

test_find_label_transitions ()

test_gen_to_arr ()

test_get_mouse_syllable_slices ()

test_get_syllable_slices ()

test_get_syllable_statistics ()

test_get_transition_martrix ()

test_get_transitions ()

test_labels_to_changepoints ()

test_merge_models ()

test_normalize_pcs ()

test_parse_batch_modeling ()

test_parse_model_results ()

test_relabel_by_usage ()

test_results_to_dataframe ()

test_retrieve_pcs_from_slices ()

test_simulate_ar_trajectory ()

test_sort_batch_results ()

test_syllable_slices_from_dict ()

moseq2_viz.tests.unit_tests.test_model_util.**make_sequence** (lbls, durs)

Unit Tests - Scalar Utilities Tests Module

class moseq2_viz.tests.unit_tests.test_scalar_utils.**TestScalarUtils** (methodName='runTest')

Bases: **unittest.case.TestCase**

test_convert_legacy_scalars ()

test_convert_pxs_to_mm ()

test_find_and_load_feedback ()

test_generate_empty_feature_dict ()

test_get_scalar_map ()

test_is_legacy ()

test_nanzscore ()

test_pca_matches_labels ()

test_process_scalars ()

test_remove_nans_from_labels ()

test_scalar_triggered_average ()

test_scalars_to_dataframe ()

test_star_valmap ()

Unit Tests - Train Label Utilities Tests Module

class moseq2_viz.tests.unit_tests.test_train_label_util.**TestTrainLabelUtils** (methodName='runTest')

Bases: **unittest.case.TestCase**

test_syll_duration ()

test_syll_id ()

test_syll_onset ()

test_to_df ()

Submodules

CLI Module

`moseq2_viz.cli.`**`new_init`** (self, *args, **kwargs)

GUI Module

`moseq2_viz.gui.`**`add_group_by_session`** (index_file, value, group, exact, lowercase, negative, output_directory=None)

Updates index file SessionName group names with user defined group names.

Parameters:
- **index_file (str)** (*path to index file*)
- **value (str)** (*SessionName value to search for*)
- **group (str)** (*group name to allocate.*)
- **exact (bool)** (*indicate whether to search for exact match.*)
- **lowercase (bool)** (*indicate whether to convert all searched for names to lowercase.*)
- **negative (bool)** (*whether to update the inverse of the found selection.*)
- **output_directory (str)** (*path to alternative index file path*)

Returns:

Return type: None

`moseq2_viz.gui.`**`add_group_by_subject`** (index_file, value, group, exact, lowercase, negative, output_directory=None)

Updates index file SubjectName group names with user defined group names.

Parameters:
- **index_file (str)** (*path to index file*)
- **value (str)** (*SessionName value to search for*)
- **group (str)** (*group name to allocate.*)
- **exact (bool)** (*indicate whether to search for exact match.*)
- **lowercase (bool)** (*indicate whether to convert all searched for names to lowercase.*)
- **negative (bool)** (*whether to update the inverse of the found selection.*)
- **output_directory (str)** (*path to alternative index file path*)

Returns:

Return type: None

`moseq2_viz.gui.`**`copy_h5_metadata_to_yaml_command`** (input_dir, h5_metadata_path)

Reads h5 metadata from a specified metadata h5 path.

Parameters:
- **input_dir (str)** (*path to directory containing h5 file*)
- **h5_metadata_path (str)** (*path to metadata within h5 file*)

Returns:

Return type: None

`moseq2_viz.gui.`**`get_groups_command`** (index_file, output_directory=None)

Jupyter Notebook to print index file current metadata groupings.

Parameters:
- **index_file (str)** (*path to index file*)
- **output_directory (str)** (*path to alternative index file path*)

Returns: **(int)**

Return type: number of unique groups

moseq2_viz.gui.**make_crowd_movies_command** (index_file, model_path, output_dir, max_syllable, max_examples, output_directory=None)

Runs CLI function to write crowd movies, due to multiprocessing compatibilty issues with Jupyter notebook's scheduler.

Parameters:
- **index_file (str)** (*path to index file.*)
- **model_path (str)** (*path to fit model.*)
- **output_dir (str)** (*path to directory to save crowd movies in.*)
- **max_syllable (int)** (*number of syllables to make crowd movies for.*)
- **max_examples (int)** (*max number of mice to include in a crowd movie.*)
- **output_directory (str)** (*alternative directory prefix to save crowd movies in.*)

Returns: **(str)**

Return type: Success string.

moseq2_viz.gui.**plot_scalar_summary_command** (index_file, output_file)

Creates a scalar summary graph and a position summary graph

Parameters:
- **index_file (str)** (*path to index file*)
- **output_file (str)** (*prefix name of scalar summary images*)

Returns: **scalar_df (pandas DataFrame)**

Return type: DataFrame containing all of scalar values for debugging.

moseq2_viz.gui.**plot_syllable_durations_command** (model_fit, index_file, groups, count, max_syllable, output_file, ylim=None)

Plot average syllable durations.

Parameters:
- **model_fit (str)** (*path to fit model.*)
- **index_file (str)** (*path to index file.*)
- **groups (tuple)** (*tuple groups to separately plot.*)
- **count (str)** (*method to calculate syllable usages, either by 'frames' or 'usage'.*)
- **max_syllable (int)** (*number of syllables to plot durations for.*)
- **output_file (str)** (*name of saved image of durations plot.*)
- **ylim (int)** (*y-axis limit of graph.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

moseq2_viz.gui.**plot_transition_graph_command** (index_file, model_fit, config_file, max_syllable, group, output_file)

Creates transition graphs given groups to process.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to fit model*)
- **config_file (str)** (*path to config file*)
- **max_syllable (int)** (*maximum number of syllables to include in graph*)
- **group (tuple)** (*tuple of names of groups to graph transition graphs for*)
- **output_file (str)** (*name of the transition graph saved image*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

moseq2_viz.gui.**plot_usages_command** (index_file, model_fit, sort, count, max_syllable, group, output_file)

Graph syllable usages from fit model data.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to fit model.*)
- **sort (bool)** (*sort by usages.*)
- **count (str)** (*method to calculate syllable usages, either by 'frames' or 'usage'*)
- **max_syllable (int)** (*max number of syllables to plot.*)
- **group (tuple)** (*groups to include in usage plot. If empty, plots default average of all groups.*)
- **output_file (str)** (*name of saved usages graph.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

General Utilities Module

`moseq2_viz.util.`**`camel_to_snake`** (s)
Converts CamelCase to snake_case

Parameters: **s (str)** (*string to convert to snake case*)

Returns: **(str)**

Return type: snake_case string

`moseq2_viz.util.`**`check_video_parameters`** (index: dict) → dict
Iterates through each extraction parameter file to verify extraction parameters were the same. If they weren't this function raises a RuntimeError.

Parameters: **index (dict)** (a *sorted_index* dictionary of extraction parameters.)

Returns: **vid_parameters (dict)**

Return type: a dictionary with a subset of the used extraction parameters.

`moseq2_viz.util.`**`clean_dict`** (dct)
Casts dict values to numpy arrays

Parameters: **dct (dict)** (*dictionary with values to clean.*)

Returns: **(dict)**

Return type: dictionary with standardized value type:list

`moseq2_viz.util.`**`get_sorted_index`** (index_file: str) → dict
Just return the sorted index from an index_file path.

Parameters: **index_file (str)** (*path to index file.*)

Returns: **sorted_ind (dict)**

Return type: dictionary of loaded sorted index file contents

`moseq2_viz.util.`**`get_timestamps_from_h5`** (h5file: str)
Returns dict of timestamps from h5file.

Parameters: **h5file (str)** (*path to h5 file.*)

Returns: **(dict)**

Return type: dictionary containing timestamp data.

`moseq2_viz.util.`**`h5_filepath_from_sorted`** (sorted_index_entry: dict) → str
Gets the h5 extraction file path from a sorted index entry

Parameters: **sorted_index_entry (dict)** (*get filepath from sorted index.*)

Returns: **(str)**

Return type: a str containing the extraction filepath

`moseq2_viz.util.`**`h5_to_dict`** (h5file, path: str = '/') → dict

Load h5 dict contents to a dict variable.

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. Default: /*)

Returns: **out (dict)**

Return type: dictionary of all h5 contents

`moseq2_viz.util.`**`load_changepoints`** (cpfile)

`moseq2_viz.util.`**`load_timestamps`** (timestamp_file, col=0)

Read timestamps from space delimited text file.

Parameters:
- **timestamp_file (str)** (*path to timestamp file*)
- **col (int)** (*column to load.*)

Returns: **ts (numpy array)**

Return type: loaded array of timestamps

`moseq2_viz.util.`**`np_cache`** (function)

`moseq2_viz.util.`**`parse_index`** (index_file: str) → tuple

Load an index file, and use extraction UUIDs as entries in a sorted index.

Parameters: **index_file**

Returns: **index (dict)** (*loaded index file contents in a dictionary*) **uuid_sorted (dict)** (*dictionary of a list of files and pca_score path.*)

`moseq2_viz.util.`**`read_yaml`** (yaml_path: str)

`moseq2_viz.util.`**`recursive_find_h5s`** (root_dir='/Users/aymanzeine/Desktop/moseq/moseq2-viz/docs', ext='.h5', yaml_string='{}.yaml')

Recursively find h5 files, along with yaml files with the same basename.

Parameters:
- **root_dir (str)** (*path to directory containing h5*)
- **ext (str)** (*extension to search for.*)
- **yaml_string (str)** (*yaml file format name.*)

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

`moseq2_viz.util.`**`star`**

Apply a function to a tuple of args, by expanding the tuple into each of the function's parameters. It is curried, which allows one to specify one argument at a time.

Parameters:
- **f (function)** (*a function that takes multiple arguments*)
- **args (tuple)** (: a tuple to expand into `f`)

Returns:

Return type: the output of `f`

`moseq2_viz.util.`**`strided_app`** (a, L, S)

Taking subarrays from numpy array given stride

Parameters:
- **a (np.array)** (*array to get subarrays from.*)
- **L (int)** (*window length.*)
- **S (int)** (*stride size.*)

Returns: **(np.ndarray)**

Return type: sliced subarrays

Visualization Module

`moseq2_viz.viz.`**`clean_frames`** (frames, medfilter_space=None, gaussfilter_space=None, tail_filter=None, tail_threshold=5)

Filters frames using spatial filters such as Median or Gaussian filters.

Parameters:
- **frames (3D numpy array)** (*frames to filter.*)
- **medfilter_space (list)** (*list of len()==1, must be odd. Median space filter kernel size.*)
- **gaussfilter_space (list)** (*list of len()==2. Gaussian space filter kernel size.*)
- **tail_filter (int)** (*number of iterations to filter over tail.*)
- **tail_threshold (int)** (*filtering threshold value*)

Returns: **out (3D numpy array)**

Return type: filtered numpy array.

`moseq2_viz.viz.`**`convert_ebunch_to_graph`** (ebunch)

Convert transition matrices to tranistion DAGs.

Parameters: **ebunch (list of tuples)** (*syllable transition data*)

Returns: **g (networkx.DiGraph)**

Return type: DAG object to graph

`moseq2_viz.viz.`**`convert_transition_matrix_to_ebunch`** (weights, transition_matrix, usages=None, usage_threshold=- 0.1, edge_threshold=- 0.1, indices=None, keep_orphans=False, max_syllable=None)

Parameters:
- **weights (np.ndarray)** (*syllable transition edge weights*)
- **transition_matrix (np.ndarray)** (*syllable transition matrix*)
- **usages (list)** (*list of syllable usages*)
- **usage_threshold (float)** (*threshold to include a syllable in list of orphans*)
- **edge_threshold (float)** (*threshold to consider an edge part of the graph.*)
- **indices (list)** (*indices of syllables to list as orphans*)
- **keep_orphans (bool)** (*indicate whether to graph orphan syllables*)
- **max_syllable (bool)** (*maximum numebr of syllables to include in graph*)

Returns: **ebunch (list)** (*syllable transition data.*) **orphans (list)** (*syllables with no edges.*)

`moseq2_viz.viz.`**`crowd_matrix_from_loaded_data`** (slices: Iterable[Tuple[int, int]], frames, scalars, nexamples=50, pad=30, dur_clip=1000, raw_size=512, 424, crop_size=80, 80)

This function assumes angles have already been treated for flips, if necessary. UNUSED

Parameters:
- **slices**
- **frames**
- **scalars**
- **nexamples**
- **pad**
- **dur_clip**
- **raw_size**
- **crop_size**

Returns:

Return type: None

`moseq2_viz.viz.`**`duration_plot`** (df, groups=None, headless=False, ylim=None, **kwargs)

Creates a seaborn pointplot depicting average syllable durations.

Parameters:
- **df (pandas DataFrame)** (*dataframe containing syllable duration data*)
- **groups (tuple)** (*groups to graph durations for*)
- **headless (bool)** (*drop first row of dataframe*)
- **ylim (int)** (*y-axis limit in figure*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **fig (pyplot figure)** (*figure to plot/save*) **ax (pyplot axis)** (*axis object of figure*)

`moseq2_viz.viz.`**`floatRgb`** (mag, cmin, cmax)

Return a tuple of floats between 0 and 1 for R, G, and B.

Parameters:
- **mag (float)** (*color intensity.*)
- **cmin (float)** (*minimum color value*)
- **cmax (float)** (*maximum color value*)

Returns: **red (float)** (*red value*) **green (float)** (*green value*) **blue (float)** (*blue value*)

`moseq2_viz.viz.`**`graph_transition_matrix`** (trans_mats, usages=None, groups=None, edge_threshold=0.0025, anchor=0, usage_threshold=0, node_color='w', node_edge_color='r', layout='spring', edge_width_scale=100, node_size=400, fig=None, ax=None, width_per_group=8, height=8, headless=False, font_size=12, plot_differences=True, difference_threshold=0.0005, difference_edge_width_scale=500, weights=None, usage_scale=100000.0, arrows=False, keep_orphans=False, max_syllable=None, orphan_weight=0, edge_color='k', **kwargs)

Creates transition graph plot given a transition matrix and some metadata.

Parameters:

- **trans_mats (np.ndarray)** (*syllable transition matrix*)
- **usages (list)** (*list of syllable usage probabilities*)
- **groups (list)** (*list groups to graph transition graphs for.*)
- **edge_threshold (float)** (*threshold to include edge in graph*)
- **anchor (int)** (*syllable index as the base syllable*)
- **usage_threshold (int)** (*threshold to include syllable usages*)
- **node_color (str)** (*node colors*)
- **node_edge_color (str)** (*node edge color.*)
- **layout (str)** (*layout format*)
- **edge_width_scale (int)** (*edge line width scaling factor*)
- **node_size (int)** (*node size scaling factor*)
- **fig (pyplot figure)** (*figure to plot to*)
- **ax (pyplot Axes)** (*axes object*)
- **width_per_group (int)** (*graph width scaling factor per group*)
- **height (int)** (*UNUSED.*)
- **headless (bool)** (*exclude first node.*)
- **font_size (int)** (*size of node label text.*)
- **plot_differences (bool)** (*plot difference between group transition matrices*)
- **difference_threshold (float)** (*threshold to consider 2 graph elements different*)
- **difference_edge_width_scale (float)** (*difference graph edge line width scaling factor*)
- **weights (list)** (*list of edge weights*)
- **usage_scale (float)** (*syllable usage scaling factor*)
- **arrows (bool)** (*indicate whether to plot arrows as transitions.*)
- **keep_orphans (bool)** (*plot orphans.*)
- **max_syllable (int)** (*number of syllables (nodes) to plot*)
- **orphan_weight (int)** (*scaling factor to plot orphan node sizes*)
- **edge_color (str)** (*edge color*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **fig (pyplot figure)** (*figure containing transition graphs.*) **ax (pyplot axis)** (*figure axis object.*) **pos (dict)** (*dict figure information.*)

`moseq2_viz.viz.`**`make_crowd_matrix`** (slices, nexamples=50, pad=30, raw_size=512, 424, frame_path='frames', crop_size=80, 80, dur_clip=1000, offset=50, 50, scale=1, center=False, rotate=False, min_height=10, legacy_jitter_fix=False, **kwargs)

Creates crowd movie video numpy array.

Parameters:
- **slices (numpy array)** (*video slices of specific syllable label*)
- **nexamples (int)** (*maximum number of mice to include in crowd_matrix video*)
- **pad (int)** (*number of frame padding in video*)
- **raw_size (tuple)** (*video dimensions.*)
- **frame_path (str)** (*path to in-h5 frames variable*)
- **crop_size (tuple)** (*mouse crop size*)
- **dur_clip (int)** (*maximum clip duration.*)
- **offset (tuple)** (*centroid offsets from cropped videos*)
- **scale (int)** (*mouse size scaling factor.*)
- **center (bool)** (*indicate whether mice are centered.*)
- **rotate (bool)** (*rotate mice to orient them.*)
- **min_height (int)** (*minimum max height from floor to use.*)
- **legacy_jitter_fix (bool)** (*whether to apply jitter fix for K1 camera.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **crowd_matrix (3D numpy array)**

Return type: crowd movie for a specific syllable.

`moseq2_viz.viz.`**`position_plot`** (scalar_df, centroid_vars=['centroid_x_mm', 'centroid_y_mm'], sort_vars=['SubjectName', 'uuid'], group_var='group', sz=50, headless=False, **kwargs)

Creates a position summary graph that shows all the mice's centroid path throughout the respective sessions.

Parameters:
- **scalar_df (pandas DataFrame)** (*dataframe containing all scalar data*)
- **centroid_vars (list)** (*list of scalar variables to track mouse position*)
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups df column to graph position plots for.*)
- **sz (int)** (*plot size.*)
- **headless (bool)** (*UNUSED*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **fig (pyplot figure)** (*pyplot figure object*) **ax (pyplot axis)** (*pyplot axis object*)

`moseq2_viz.viz.`**`scalar_plot`** (scalar_df, sort_vars=['group', 'uuid'], group_var='group', show_scalars=['velocity_2d_mm', 'velocity_3d_mm', 'height_ave_mm', 'width_mm', 'length_mm'], headless=False, **kwargs)

Creates scatter plot of given scalar variables representing extraction results.

Parameters:
- **scalar_df (pandas DataFrame)**
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups df column to graph position plots for.*)
- **show_scalars (list)** (*list of scalar variables to plot.*)
- **headless (bool)** (*exclude head of dataframe from plot.*)
- **kwargs (dict)** (*extra keyword variables*)

Returns: **fig (pyplot figure)** (*plotted scalar scatter plot*) **ax (pyplot axis)** (*plotted scalar axis*)

`moseq2_viz.viz.`**`usage_plot`** (usages, groups=None, headless=False, **kwargs)

Creates a syllable usage plot for the given group

Parameters:

- **usages (pandas DataFrame)** (*DataFrame containing syllable usages and other metadata*)
- **groups (tuple)** (*groups to graph usages for.*)
- **headless (bool)** (*Drop first row of usages.*)
- **kwargs (dict)** (*extra keyword arguments.*)

Returns: **fig (pyplot figure)** (*figure to plot/save*) **ax (pyplot axis)** (*axis object of figure*)

Indices and tables

- `genindex`
- `modindex`
- `search`

Index

N

P

U

W

Python Module Index

m