

Loading the data

Setting up

To begin you need three paths saved as variables:

1. the path to the metadata `.csv` file
2. the full path (including folder) the ftp location (eg: `ftp://myethoscopecom/results/`)
3. the path of a local folder to save downloaded `.db` files (if your `.db` files are already downloaded on the same machine running ethoscipy, then this will be the path to the folder containing them)

```
import ethoscipy as etho

# Replace with your own file/sever paths

meta_loc = 'USER/experiment_folder/metadata.csv'
remote = 'ftp://ftpsever/auto_generated_data/ethoscope_results'
local = 'USER/ethoscope_results'

# This will download the data remotely via FTP onto the local machine
# If your ethoscipy is running via ethoscope-lab on the same machine
# where the ethoscope data are, then this step is not necessary
etho.download_from_remote_dir(meta_loc, remote, local)
```

download_from_remote_dir - Function reference

download_from_remote_dir(meta, remote_dir, local_dir):

This function is used to import data from the ethoscope node platform to your local directory for later use. The ethoscope files must be saved on a remote FTP server and saved as `.db` files, see the Ethoscope manual for how to setup a node correctly. <https://www.notion.so/giorgiogilestro/Ethoscope-User-Manual-a9739373ae9f4840aa45b277f2f0e3a7>

Args:

meta (**str**): The path to a csv file containing columns with machine_name, date, and time if multiple files on the same

day

remote_dir (**str**): The url containing the location of the ftp server up to the folder contain the machine id's, server must

not have a username or password (anonymous login) e.g.

'ftp://YOUR_SERVER//auto_generated_data//ethoscope_results'

local_dir (**str**): The path of the local directory to save `.db` files to, files will be saved using the structure of the ftp server

e.g. 'C:\\Users\\YOUR_NAME\\Documents\\ethoscope_databases'

returns:

None

This only needs to be run if like the Gilestro lab you have all your data saved to a remote ftp sever. If not you can skip straight to the the next part.

Create a modified metadata DataFrame

This function creates a modified metadata DataFrame with the paths of the saved .db files and generates a unique id for each experimental individual. This function only works for a file structure locally saved to whatever computer you are running and is saved in a nested director structure as created by the ethoscopes, i.e.

'LOCAL_DIR/ethoscope_results/00190f0080e54d9c906f304a8222fa8c/ETHOSCOPE_001/2022-08-23_03-33-59/DATABASE.db'

For this function you only need the path to the metadata file and the path the the first higher level of your database directories, as seen in the example below. **Do not** provide a path directly to the folder with your known .db file in it, the function searches all the saved data directories and selects the ones that match the metadata file.

```
meta_loc = 'USER/experiment_folder/metadata.csv'
local = 'USER/ethoscope_results' # remember to just provide the path up to the directory where the individual
ethoscope files are saved

meta = etho.link_meta_index(meta_loc, local)
```

link_meta_index - Function reference

link_meta_index(metadata, local_dir):

A function to alter the provided metadata file with the path locations of downloaded .db files from the Ethoscope experimental system. The function will check all unique machines against the original ftp server for any errors. Errors will be omitted from the returned metadata table without warning.

Args:

metadata (**str**): The path to a file containing the metadata information of each ROI to be downloaded, must include

'ETHOSCOPE_NAME', 'date' in yyyy-mm-dd format or others (see validate_datetime), and 'region_id'

local_dir (**str**): The path to the top level parent directory where saved database files are located.

returns:

A pandas DataFrame containing the csv file information and corresponding path for each entry in the csv

Load and modify the ethoscope data

The load function takes the raw ethoscope data from its .db format and modifies it into a workable pandas DataFrame format, changing the time (seconds) to be in reference to a given hour (usually lights on). Min and max times can be provided to filter the data to only recordings between those hours. **With 0 being in relation to the start of the experiment not the reference hour.**

```
data = etho.load_ethoscope(meta, min_time = 24, max_time = 48, reference_hour = 9.0)
```

```
# you can cache the each specimen as the data is loaded for faster load times when run again, just add a file  
path to a folder of choice, the first time it will save, the second it will search the folder and load straight from  
there
```

```
# However this can take up a lot of memory and it's recommended to save the whole loaded dataset at the end  
and to load from this each time. See the end of this page
```

```
data = etho.load_ethoscope(meta, min_time = 24, max_time = 48, reference_hour = 9.0, cache =  
'path/ethoscope_cache/')
```

load_ethoscope - Function reference

load_ethoscope(metadata, min_time = 0 , max_time = float('inf'), reference_hour = None, cache = None, FUN = None, verbose = True):

The users function to iterate through the dataframe generated by link_meta_index() and load the corresponding database files and analyse them according to the inputted function.

The users function to iterate through the dataframe generated by link_meta_index() and load the corresponding database files

Args:

metadata (**pd.DataFrame**): The metadata dataframe as returned from link_meta_index function

min_time (**int, optional**): The minimum time you want to load data from with 0 being the experiment start (in hours), for

all experiments. Default is 0.

max_time (**int, optional**): Same as above, but for the maximum time you want to load to. Default is infinity.

reference_hour (**int, optional**): The hour at which lights on occurs when the experiment is begun, or when you want

the timestamps to equal 0. None equals the start of the experiment. Default is None.

cache (**str, optional**): The local path to find and store cached versions of each ROI per database. Directory tree

structure is a mirror of ethoscope saved data. Cached files are in a pickle format. Default is None.

FUN (**function, optional**): A function to apply individual curation to each ROI, typically using package generated

functions (i.e. sleep_annotation). If using a user defined function use the package analyse functions as examples. If None the data remains as found in the database. Default is None.

verbose (**bool, optional**): If True (default) then the function prints to screen information about each ROI when loading,

if False no printing to screen occurs. Default is True.

returns:

A pandas DataFrame object containing the database data and unique ids per fly as the index

Additionally, an analysing function can be also called to modify the data as it is read. **It's recommended you always call at least max_velocity_detector or sleep_annotation function when loading the data as it generates columns that are needed for the analysis / plot generating methods.**

```
from functools import partial
```

```
data = etho.load_ethoscope(meta, reference_hour = 9.0, FUN = partial(etho.sleep_annotation,  
time_window_length = 60, min_time_immobile = 300))
```

```
# time_window_length is the amount of time each row represents. The ethoscope can record multiple times per  
second, so you can go as low as 10 seconds for this.
```

```
# The default for time_window_length is 10 seconds
```

```
# min_time_immobile is your sleep criteria, 300 is 5 mins the general rule of sleep for flies, see Hendricks et al.,  
2000.
```

Ethoscopy has 2 general functions that can be called whilst loading:

- **max_velocity_detector**: Aggregates variables per the given time window, finding their means. Sleep_annotation uses this function before finding sleep bouts, so use this when you don't need to know the sleep bouts.
- **sleep_annotation**: Aggregates per time window and generates a new boolean column of sleep, as given by the time immobile argument.

max_velocity_detector - Function reference

max_velocity_detector(data, time_window_length, velocity_correction_coef = 3e-3, masking_duration = 6, optional_columns = 'has_interacted'):

Max_velocity_detector is the default movement classification for real-time ethoscope experiments. It is benchmarked against human-generated ground truth.

Args:

data (pd.DataFrame): A dataframe containing behavioural variables of a single animal (no id)
time_window_length (int): The period of time the data is binned and sampled to, i.e. if 60 the timestep will per row will be 60 seconds.

velocity_correction_coef (float, optional): A coefficient to correct the velocity data (change for different length tubes).

For 'small' tubes (20 per ethoscope) = 3e-3, for 'long' tubes (10 per ethoscope) = 15e-4. Default is 3e-3.

masking_duration (int, optional): The number of seconds during which any movement is ignored (velocity is set to 0)

after a stimulus is delivered (a.k.a. interaction). If using the AGO set to 0. Default is 6.

optional_columns (str, optional): The columns other than ['t', 'x', 'velocity'] that you want included post analysis. Default

is 'has_interacted'.

returns:

A pandas dataframe object with columns such as 't', 'moving', 'max_velocity', 'mean_velocity' and 'beam_cross'

sleep_annotation - Function reference

sleep_annotation(data, time_window_length = 10, min_time_immobile = 300, motion_detector_FUN = max_velocity_detector, masking_duration = 6, velocity_correction_coef = 3e-3):

This function first uses a motion classifier to decide whether an animal is moving during a given time window. Then, it defines sleep as contiguous immobility for a minimum duration.

Args:

data (pd.DataFrame): The dataframe containing behavioural variables from one animals.

time_window_length (int): The period of time the data is binned and sampled to.
Default is 10

min_time_immobile (int, optional): Immobility bouts longer or equal to this value are considered as asleep.

Default is 300 (i.e 5 mins)

motion_detector_FUN (function, optional): The function to curate raw ethoscope data into velocity measurements.

Default is max_velocity_detector.

masking_duration (int, optional): The number of seconds during which any movement is ignored (velocity is set to 0)

after a stimulus is delivered (a.k.a. interaction). If using the AGO set to 0. Default is 6.

velocity_correction_coef (float, optional): A coefficient to correct the velocity data (change for different length tubes).

For 'small' tubes (20 per ethoscope) = $3e-3$, for 'long' tubes (10 per ethoscope) = $15e-4$. Default is $3e-3$.

returns:

A pandas dataframe containing columns 'moving' and 'asleep'

Ethoscopy also has 2 functions for use with AGO or mAGO ethoscope module (odour delivery and mechanical stimulation):

- **stimulus_response**: Finds the interaction times and then searches a given window post interaction for movement.
- **stimulus_prior**: A modification of stimulus_response, the function finds all interaction times and their response whilst retaining all the previous variables information in a given time window. I.e. if the window is 300, all 300 seconds worth of data will be retained prior to the second when the stimulus is given.

stimulus_response - Function reference

stimulus_response(data, start_response_window = 0, response_window_length = 10, add_false = False, velocity_correction_coef = $3e-3$):

Stimulus_response finds interaction times from raw ethoscope data to detect responses in a given window. This function will only return data from around interaction times and not whole movement data from the experiment.

Args:

data (pd.DataFrame): The dataframe containing behavioural variable from many or one multiple animals

response_window (int, optional): The period of time (seconds) after the stimulus to check for a response (movement). Default is 10 seconds

add_false (bool / int, optional): If not False then an int which is the percentage of the total of which to add false

interactions, recommended is 10. This is for use with old datasets with no false interactions so you can observe spontaneous movement with a HMM. Default is False

velocity_correction_coef (**float, optional**): A coefficient to correct the velocity data (change for different length tubes).

Default is 3e-3.

returns:

A pandas dataframe object with columns such as 'interaction_t' and 'has_responded'

stimulus_prior - Function reference

stimulus_prior(data, window = 300, response_window_length = 10, velocity_correction_coef = 3e-3):

Stimulus_prior is a modification of stimulus_response. It only takes data with a populated has_interacted column.

The function will take a response window (in seconds) to find the variables recorded by the ethoscope in this window prior to an interaction taking place. Each run is given a unique ID per fly, however it is not unique to other flies. To do so, combine the fly ID with run ID after.

Args:

data (**pd.DataFrame**): A dataframe containing behavioural variable from many or one multiple animals

window (**int, optional**): The period of time (seconds) prior to the stimulus you want data retrieved for. Default is 300.

response_window_length (**int, optional**): The period of time (seconds) after the stimulus to check for a response

(movement). Default is 10 seconds.

velocity_correction_coef (**float, optional**): A coefficient to correct the velocity data (change for different length tubes).

Args:Default is 3e-3

returns:

A pandas dataframe object with columns such as 't_count' and 'has_responded'

Saving the data

Loading the ethoscope data each time can be a long process depending on the length of the experiment and number of machines. It's recommended to save the loaded/modified DataFrame as a pickle `.pkl` file. See [here](#) for more information about pandas and pickle saves. The saved behavpy object can then be loaded in instantly at the start of a new session!

```
# Save any behavpy or pandas object with the method below
```

```
import pandas as pd
```

```
df.to_pickle('path/behavpy.pkl') # replace string with your file location/path
```

```
# Load the saved pickle file like this. It will retain all the metadata information
```

```
df = pd.read_pickle('path/behavpy.pkl')
```

Revision #28

Created 22 November 2022 19:23:03 by Giorgio Gilestro

Updated 16 November 2023 13:49:03 by Laurence Blackhurst