Instructions for REMs analysis using manual peak detection

Scripts required:

manual_peak_detection.py

rems_analyse.py

rems_functions.py

Overview

What is manual eye movement detection?

Eye movements in a signal can either be identified by a person marking the peaks in the data corresponding to eye movements (manual detection), or by an peak detection algorithm (automatic detection).

Why are we doing manual eye movement detection?

1. To give you a better understanding of eye movement characteristics during REM sleep
2. To compare manual detection outputs to automatic detection outputs which will (hopefully!) validate the automatic detection method. This means that we have proof the automatic detection detects eye movements to a comparable extent as a person does.

What do I need to perform manual eye movement detection?

- A Python IDE (e.g. PyCharm)
- The following data files from a participant:
  - Hypnogram file (.csv format)
  - EEG data file (.edf format)
  - EOG data file (.edf format)
- The following python scripts downloaded into your PycharmProject folder:
  - manual_peak_detection.py - you will 'interact' with this file by changing the paths at the start of the file
  - rems_analyse.py - you do not need to 'interact' with this file in any way
  - rems_functions.py - you do not need to 'interact' with this file in any way

Running SECTION 1

Section 1 imports modules, specifies file paths, aligns the sleep hypnogram to the EOG data & extracts REM periods

1. Open the manual_peak_detection.py file in PyCharm

2. Make sure you have all the modules imported (**lines 9-24**).
   - To import modules, open the PyCharm terminal & type ‘pip install [name_of_module]’ & press enter

3. Specify the paths of the files you will be analysing (**lines 28-33**)
   - Remember to check the spelling of your file names in PyCharm
   - Remember to include the file type in the string (e.g. ‘scoring_vid.csv’)

4. Copy **lines 9-38**, paste into the Python Console & press enter

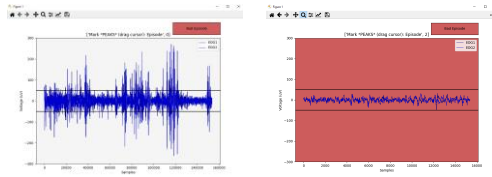
```
1  """
2  script to manually identify eye movements during REM sleep & analyse
3  author: @aggpr141
4  last updated: 27/11/21
5  """
6  # ----------------------------------------------------------------------------------------------------
7  # SECTION 1
8
9  # import modules - pip install these!
10 import os
11 import sys
12 from pathlib import Path
13 import mne
14 import numpy as np
15 import pandas
16 from matplotlib import pyplot as plt
17 import scipy
18 import neurokit2 as nk
19 import statistics
20 import matplotlib.widgets as mwidgets
21 from rems_analyse import *
22 from rems_functions import *
23 from matplotlib.widgets import Button
24 import warnings
25
26 warnings.filterwarnings("ignore")
27
28 # specify paths
29 path = Path('Y:/22qEEG/E007-2-2-1')  # define path to participant folder
30 os.chdir(path)  # change working directory to path
31 hypnogram = 'scoring_outputs/E007-2-2-1_scoring_info_vid.csv'  # define hypnogram path
32 EEG = 'exported_data/E007-2-2-1_sleep_EEG_PREP.edf'  # define EEG data path (for hypnogram alignment)
33 EOG = 'exported_data/E007-2-2-1_PSG.edf'  # define EOG data path
34 sampling_freq = 256  # define sampling frequency for EOG data
35
36 # preprocess data & extract rem periods for each eog channel
37 rem_episodes_e1, rem_episodes_e2, rem_timings = extract_rem_episodes(EEG, EOG, hypnogram)
38
```

Running SECTION 2

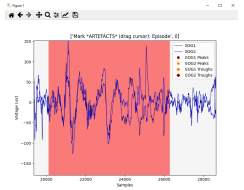
Section 2 opens several plots of data so you can 1. manually identify eye movements 2. mark bad episodes 3. mark artefacts. The script then automatically extracts eye movement, cluster & microstate characteristics

1. Copy **lines 42-108**, paste into the Python Console & press enter

2. For each episode, a graph will open with the episode's data plotted. You have 2 choices:
   - Manually define the eye movements
   - Mark the entire episode as a 'bad' episode



3. If you mark the episode as 'bad' then that episode is ignored for all future analyses
4. If you manually define the eye movements, a new graph opens & you can mark the artefact regions of the episode



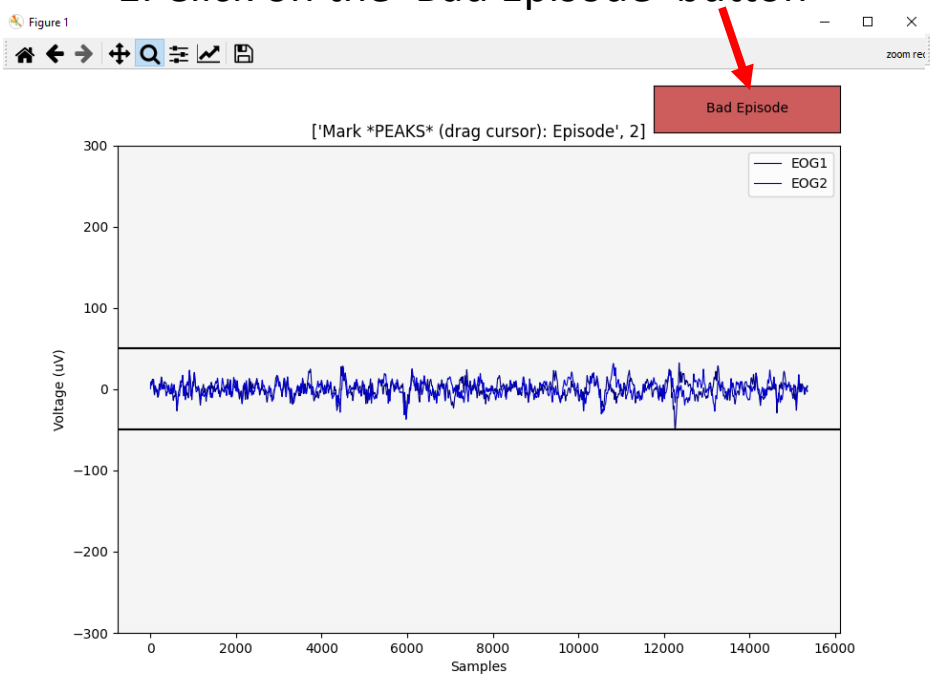
5. All REMs characterisation is then calculated automatically for the episode
6. This process repeats automatically until all the episodes have been gone through

```
# ------------------------------------------------------------------------------------------------------------------------
# SECTION 2 - manual identification of peaks, mark bad episodes, mark artefacts, calculate rem, cluster & microstate characteristics

# initialise variables to append to later
sys.stdout = open('processing_output.txt', 'w')
episode_count = 0
rems_df = pandas.DataFrame()
rems_cluster_df = pandas.DataFrame()
episode_list = []
bad_episode_list = []
good_episode_list = []
ep_list = []  # initalise episode list
pp_list = []  # initialise phasic percentage list
tp_list = []  # initialise tonic percentage list
ap_list = []  # initialise artefact percentage list
td_list = []  # initialise total duration list
ttd_list = []  # initialise total ton. duration list
tpd_list = []  # initalise total phasic duration list
tad_list = []  # initialise total art duration list
rems_microstates_df = pandas.DataFrame()

# manual peak detection for each REM episode - mark peaks or specify 'bad episode'. automatic analysis follows
for episode in range(len(rem_episodes_e1)):
    peak_range_start, peak_range_end, clean_e1, clean_e2, e1_peaks, e1_troughs, e2_peaks, e2_troughs, \
    channel_crossings, bad_episode_list, good_episode_list = manual_peak_detect(episode, rem_episodes_e1,
                                                                                 rem_episodes_e2, bad_episode_list,
                                                                                 good_episode_list, episode_count)
    episode_count += 1
    # if episode is marked 'bad' it is excluded from further analysis
    if episode in bad_episode_list:
        print('episode', episode, 'is a bad episode. no analysis done.')
    # if episode is marked 'good' you will be asked to manually mark artefacts. automatic analysis follows.
    elif episode in good_episode_list:
        # mark artefacts
        start_art, end_art = plot_episode(e1_peaks, e2_peaks, e1_troughs, e2_troughs, clean_e1, clean_e2, episode)
        e1_peaks, e1_troughs, e2_peaks, e2_troughs = remove_art_peaks(e1_peaks, e2_peaks, e1_troughs, e2_troughs,
                                                                      start_art, end_art, episode)
        # analyse peaks/troughs from each channel
        rems_df_e1p = rems_analyse(clean_e1, e1_peaks, channel_crossings, 'Left', 'Left', e1_peaks, e1_troughs,
                                   e2_peaks, e2_troughs, rem_episodes_e1, episode, invert=False)

        rems_df_e1t = rems_analyse(clean_e1, e1_troughs, channel_crossings, 'Left', 'Right', e1_peaks, e1_troughs,
                                   e2_peaks, e2_troughs, rem_episodes_e1, episode, invert=True)

        rems_df_e2p = rems_analyse(clean_e2, e2_peaks, channel_crossings, 'Right', 'Right', e1_peaks, e1_troughs,
                                   e2_peaks, e2_troughs, rem_episodes_e2, episode, invert=False)

        rems_df_e2t = rems_analyse(clean_e2, e2_troughs, channel_crossings, 'Right', 'Left', e1_peaks, e1_troughs,
                                   e2_peaks, e2_troughs, rem_episodes_e2, episode, invert=True)

        rems_df = rems_df.append([rems_df_e1p, rems_df_e1t, rems_df_e2p, rems_df_e2t])  # compile rem characteristics

        remgram, rems_clusters, tonic_percentage, phasic_percentage, art_percentage, total_duration, total_ton_dur, \
        total_phas_dur, total_art_dur = phasic_tonic_detections(e1_peaks, e2_peaks, e1_troughs, e2_troughs, clean_e1,
                                                                episode, start_art, end_art)

        rems_cluster_df = rems_cluster_df.append(rems_clusters)  # compile cluster characteristics

        ep_list, tp_list, pp_list, td_list, ttd_list, tpd_list, ap_list, tad_list = initialise_tp_micro(
            episode, tonic_percentage, phasic_percentage, ep_list, tp_list, pp_list, td_list, ttd_list, tpd_list,
            total_duration, total_ton_dur, total_phas_dur, ap_list, tad_list, art_percentage, total_art_dur)

    # compile microstate characteristics
    rems_microstates_df = rems_microstates(ep_list, tp_list, pp_list, ap_list, td_list, ttd_list, tpd_list, tad_list)
sys.stdout.close()
```

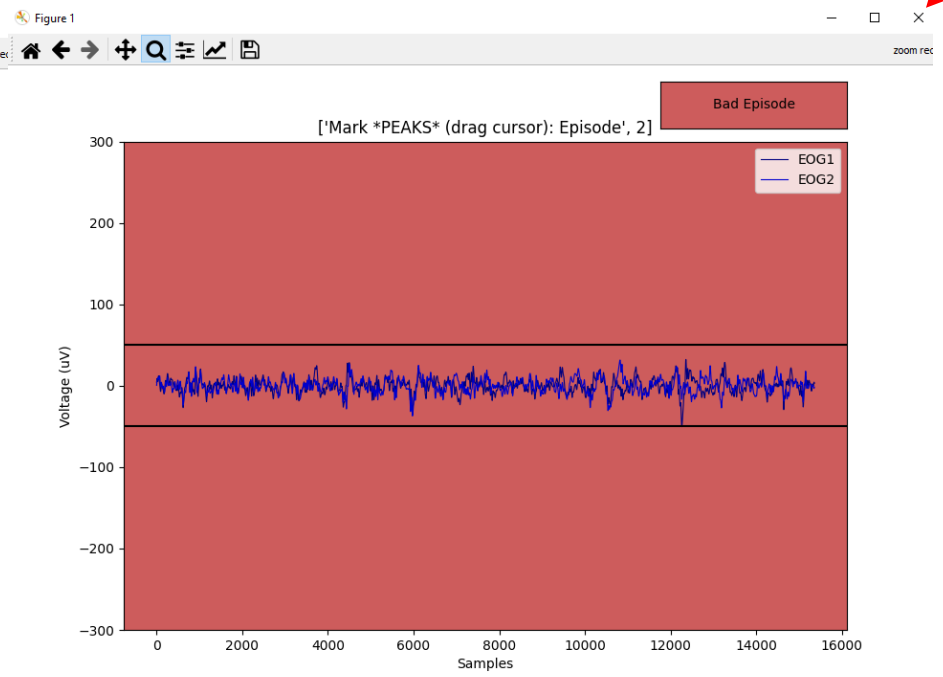
Spotlight on: Marking a Bad Episode

Bad episodes are where the majority/all of the data has an artefact (ECG bleed-through, for example) & you are unable to detect eye movements.

1. Click on the ‘Bad Episode’ button



2. The entire graph will go red

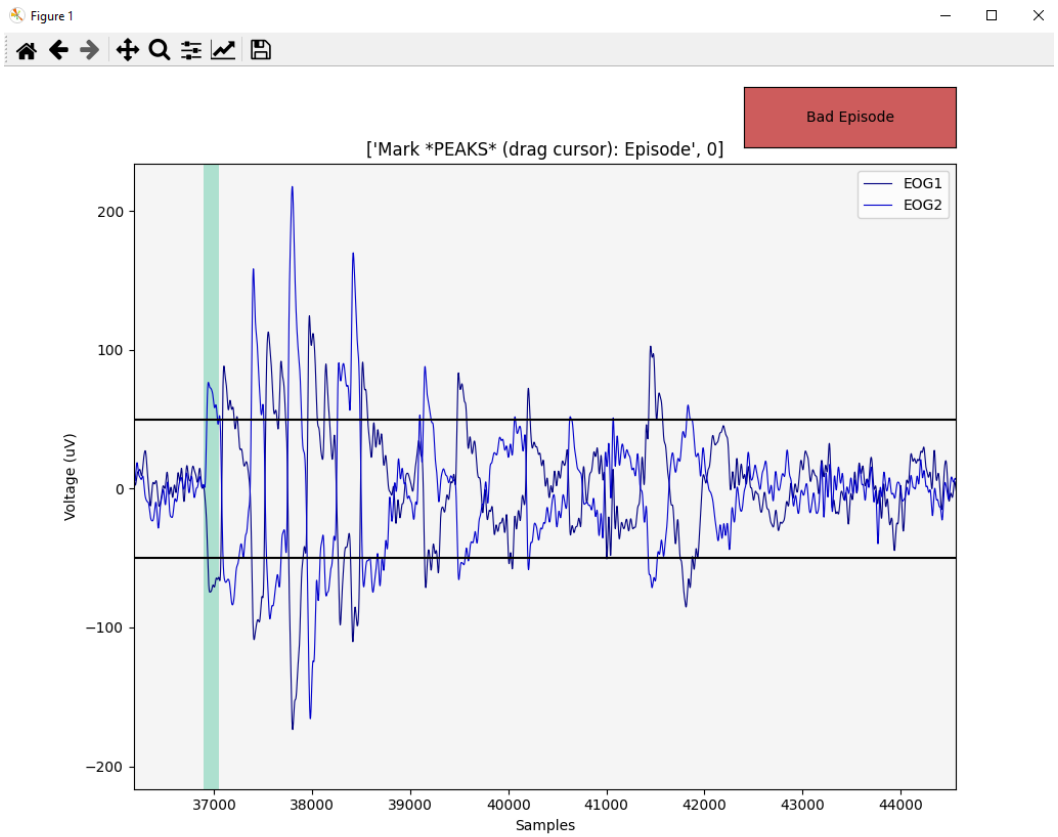


3. Exit the graph. This episode is now marked as ‘bad’ and will be ignored in all future analyses

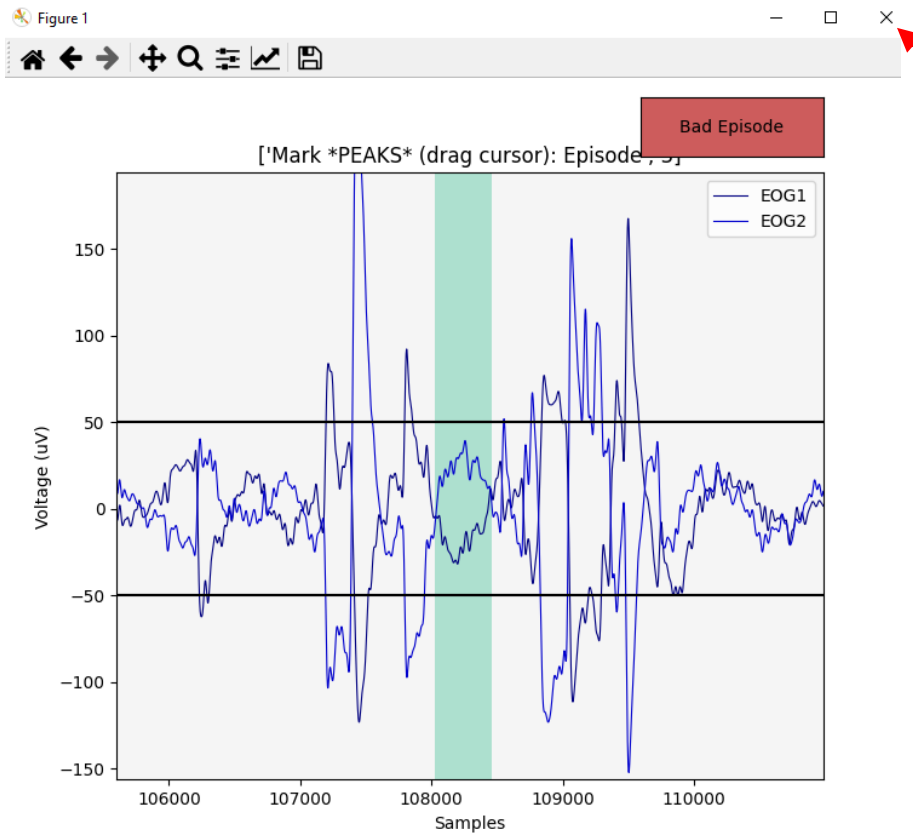
Spotlight on: Manually Defining Eye Movements

Manual identification of eye movements relies on you being confident in what an eye movement is- see Movement_V_Artefact_Identification.pdf for more guidance

Marking a ‘horizontal’ eye movement
Drag the cursor over the peak



Marking a ‘circular/oblique’ eye movement
Drag the cursor from the preceding channel crossing to the following channel crossing

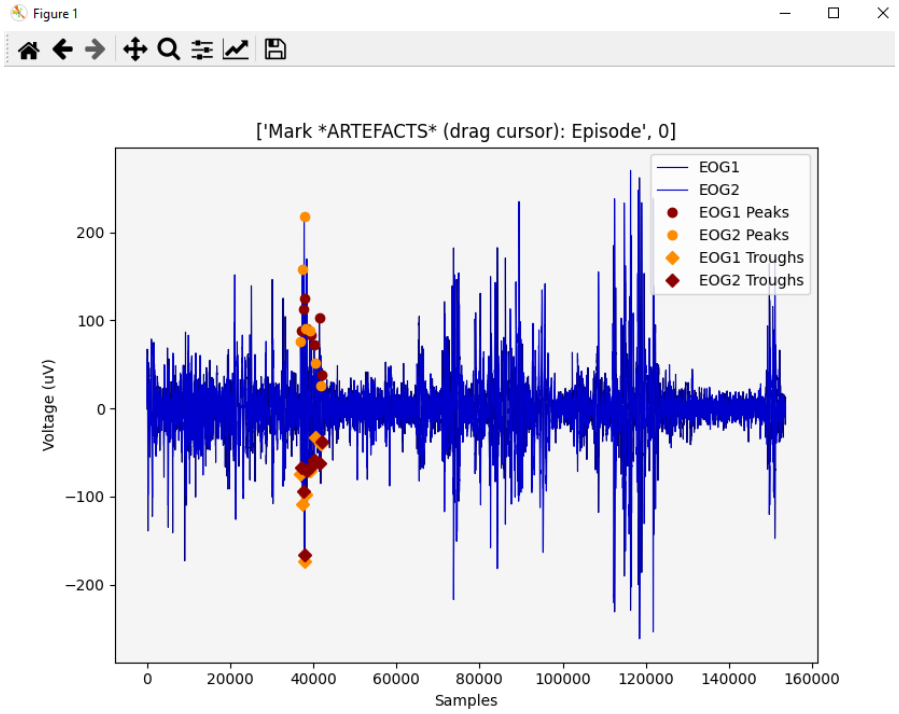


* Once you have marked all the eye movements in the episode, exit the graph and the ‘artefacts’ graph will appear so you can mark any artefacts

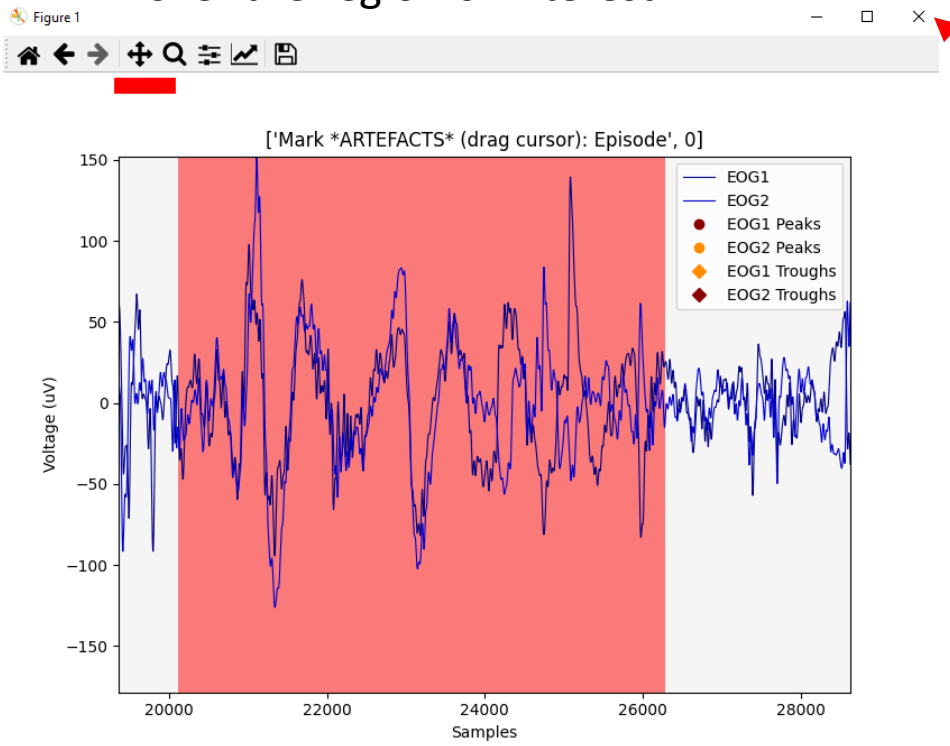
Spotlight on: Marking Artefacts

Manual identification of artefacts relies on you being confident in what an eye movement is- see Movement_V_Artefact_Identification.pdf for more guidance

The episode will be plotted & any manually identified eye movements will be plotted



Using the zoom & move functions of the GUI you can then visualise the data. Mark any artefact sections by dragging your cursor over the region of interest



* Once you have marked all the artefacts in the episode, exit the graph and the automatic analyses of the eye movements will happen

Running SECTION 3

Section 3 saves the output files at the end of the analyses, as well as a 'manual_processing_output.txt' file which details which episodes were bad & the percentage of peaks rejected due to artefacts/episode.

```
105  # ----------------------------------------------------------------------------------------------------------------------
106  # SECTION 3  - save outputs to csv files
107
108  # save individual eye movement, cluster & microstate characteristics to .csv file format into participant folder
109  rems_df.to_csv(path_or_buf='manual_rems_characteristics.csv', index_label='Peak Number in Ep')
110  rems_cluster_df.to_csv(path_or_buf='manual_rems_clusters.csv', index=False)
111  rems_microstates_df.to_csv(path_or_buf='manual_rems_microstates.csv', index=False)
112
```

1. Copy **lines 110-115**, paste into the Python Console & press enter
   - The following files will be saved to the participant's folder:
     - manual_rems_characteristics.csv - contains characteristics for each 'peak' (i.e. eye movement) during a REM sleep episode
     - manual_rems_clusters.csv - contains characteristics for each 'cluster' (i.e. group of eye movements) during a REM sleep episode
     - manual_rems_microstates.csv - contains data on macro characteristics of each episode
     - manual_processing_output.txt - contains information on the analysis status of each episode (e.g. bad episode, code bug & episode analysis skipped, analysis completed)

Questions

What if I make a mistake with my selections?

There is no ‘undo’ button on any of the graphs, so if you make a mistake you can either a) continue with the analyses or b) close the python console in PyCharm and start the entire analysis again. It is up to you which you do, if you are on the final episode it might be better to continue your analyses and make a note of your mistake. The occasional incorrectly selected movement shouldn’t impact the overall results as you will be calculating averages.

What if there is an error in the analysis pipeline?

There is a mechanism built in to the code so that if any error comes up (unforeseen calculation bugs etc) then the entire episode will be skipped and you will move onto the next REM episode. Sometimes strange errors appear due to small differences between participants and it is hard to predict. If an episode is skipped, this will be shown in the ‘manual_processing_output.txt’ file. You can then analyse that episode individually.