Neural Engineering Project Notes:

To parse the data, there are software markers that are in a “marker stream” in the .xdf file.

Below is some example code to parse the markers in python. This can also be done in matlab.

Note that this is after the xdf stream is imported using the pyxdf library:

```
   # Extract EEG and marker data
   eeg_timestamps = np.array(eeg_stream['time_stamps'])
   eeg_data = np.array(eeg_stream['time_series']).T  # (n_channels,
n_samples)
   channel_names = get_channel_names_from_xdf(eeg_stream)

   marker_values = np.array([int(m[0]) for m in
marker_stream['time_series']])
   marker_timestamps = np.array([float(value[1]) for value in
marker_stream['time_series']])
```

The markers dataset (marker_stream[“time_series”]) will look something like this:

```
[[ 200.  1075.2926]
 [ 220.  1080.2528]
 [ 300.  1080.2648]
 [ 320.  1093.2731]
 [ 200.  1099.3313]
 [ 220.  1104.3313]
 [ 300.  1104.3492]
 [ 320.  1117.3499]
 [ 100.  1123.3976]
```

The convention is [marker value, timestamp].

Once imported, you will need to parse through the software triggers to find experimental timings. This will allow you to form your data into epochs. Below are the software trigger mappings that you can use to find synchronized events in the data.

```
# software triggers
TRIGGERS = {
   "MI_BEGIN": "200",
   "MI_END": "220",
   "MI_EARLYSTOP": "240",
   "ROBOT_BEGIN": "300",
   "ROBOT_END": "320",
   "ROBOT_EARLYSTOP": "340",
   "ROBOT_CONFIRM_STOP": "345",
   "REST_BEGIN": "100",
   "REST_END": "120",
   "REST_EARLYSTOP": "140"

}
```

The most important triggers are MI_BEGIN/MI_END and REST_BEGIN/REST_END. You can use these timestamps to determine where the trials are. Alternatively, the offline trials are hardcoded in time, so you can also use the begin times, followed by a static offset (for offline only).

Below are the configured times for each experimental window:

```
TIME_MI = 5 # time for motor imagery and rest
TIME_ROB = 13 # time allocated for robot to move
TIME_STATIONARY = 2 # time for stationary feedback after no
movement/failed movement trial
```

The time allocated for MI (and rest) is 5s, the time for robot movement is 13 seconds, and the time after a failed trial is 2s.

For the online trials, there is an early cutout condition - therefore the above timings cannot be assumed. When running your model against the test set, make sure you parse through and find the corresponding END timestamps and only parse between BEGIN and END.

Finally, EARLYSTOP and CONFIRM_STOP are there for internal reasons. EARLYSTOP is only sent when an early cutout has been activated, and CONFIRM_STOP is a communication initiated from the robot as an acknowledgement. You won't need to use these.

For topographical analysis - refer to the below lines of code as an example of how to import and load the channels. I recommend using the standard 10-20 montage included in MNE toolbox if you are using python:

```
# Extract EEG timestamps and data
eeg_timestamps = np.array(eeg_stream["time_stamps"])  # (N_samples,)
eeg_data = np.array(eeg_stream["time_series"]).T  # Shape: (n_channels,
n_samples)
channel_names = get_channel_names_from_xdf(eeg_stream)
marker_data = np.array([int(value[0]) for value in
marker_stream['time_series']])
#marker_timestamps = np.array(marker_stream['time_stamps'])
marker_timestamps = np.array([float(value[1]) for value in
marker_stream['time_series']])

#FILTER: Keep ONLY markers 100, 120, 200, 220
target_markers = [100, 120, 200, 220]
keep = np.isin(marker_data, target_markers)

removed = int(marker_data.size - keep.sum())
marker_data = marker_data[keep]
marker_timestamps = marker_timestamps[keep]

print(f"🧹 Removed {removed} markers not in {target_markers}; kept
{marker_data.size}.")

#print(marker_stream['time_series'])
#print(marker_timestamps)
print("\n EEG Channels from XDF:", channel_names)
#print(marker_stream[0])
# Load standard 10-20 montage
montage = mne.channels.make_standard_montage("standard_1020")

# Case-sensitive renaming dictionary
rename_dict = {
   "FP1": "Fp1", "FPZ": "Fpz", "FP2": "Fp2",
   "FZ": "Fz", "CZ": "Cz", "PZ": "Pz", "POZ": "POz", "OZ": "Oz"
}
```

```
# Drop non-EEG channels
non_eeg_channels = {"AUX1", "AUX2", "AUX3", "AUX7", "AUX8", "AUX9",
"TRIGGER"}
valid_eeg_channels = [ch for ch in channel_names if ch not in
non_eeg_channels]

# Filter data to keep only valid EEG channels
valid_indices = [channel_names.index(ch) for ch in valid_eeg_channels]  #
Get indices
eeg_data = eeg_data[valid_indices, :]  # Keep only valid EEG data

# Create MNE Raw Object
sfreq = config.FS
info = mne.create_info(ch_names=valid_eeg_channels, sfreq=sfreq,
ch_types="eeg")
raw = mne.io.RawArray(eeg_data, info)
```

Additional recommendations:

See the code snippet above - develop a version that only keeps the Markers 100, 120, 200, and 220. This will discard all markers that are not necessary for your analysis. An example of this is in light purple in the above snippet.

If you are applying a baseline correction to your epochs, try your analysis 2 ways:

1. Using the marker value timestamp as given, for example: [100.   1123.3976] ← timestamp is 1123.3976. When you epoch your data, this is t=0 relative to a trial. Baseline your data on [-1,0] and analyze [0,5].
2. Same as above, except baseline your data on [-2.5,-1.5], and analyze on [-1.5, 5]

Reason is due to the experimental timings:

Marker_timestamp - 1.5s corresponds with the time when the “prepare” message shows up
Marker_timestamp corresponds with the trail start (feedback w/ bar increasing or online neuromodulation).

For any questions please reach out to apaydarfar@utexas.edu