Documentation

March 29, 2023

1 ADA Base - Documentation

1.1 Dataset Information

Welcome to the documentation for our multimodal dataset, which is designed to support research in the field of multimodal estimation of cognitive load.

The dataset contains multimodal data sources, including the following signals:

- `SKT`: Skin Temperature (SKT) signal from the tip of the finger
- `ECG`: Electrocardiography (ECG) signal from the chest - 2 leads
- `RSP`: Respiration signal from the chest
- `EMG`: Electromyography (EMG) signal from the trapezius
- `EDA`: Electrodermal activity (EDA) signal from the palm
- `EYE`: Eye-tracking data from an eye-tracker device (Tobii Pro Fusion) including the xyz coordinates of the gaze and pupil dilation

The dataset is stored in the Hierarchical Data Format 5 (HDF5) format, which is a file format designed to store and organize large amounts of data.

Each subject has one HDF5 file, which contains several groups:

- The `SIGNALS` group contains the raw signals captured during the experiments, including the `SKT`, `ECG`, `RSP`, `EMG`, `EDA`, and `EYE` signals.
- The `PERFORMANCE` group contains the performance metrics from the tests.
- The `SUBJECTIVE` group contains the NASA-TLX scores.

Sampling Rates

- `SKT` (100 Hz)
- `ECG` (500 Hz)
- `RSP` (250 Hz)
- `EMG` (1000 Hz)
- `EDA` (500 Hz)
- `EYE` (250 Hz)

Data
Please note that the dataframe uses `NaN` as a filler since the signals have different sampling rates. To obtain the original signal, use the `.dropna()` method on a specific column. Additionally, the Eye tracker signal may contain `INFs`. These are placeholders used to fill values that are missing in the original data, such as during blinks.

Annotation
The study has two tasks: n-Back and k-drive. The annotation is coded using the variables `STUDY`, `PHASE`, and `LEVEL`, which can be found in the `SIGNALS` group of the hdf5 file.

The `STUDY` variable is used to denote the task being performed, such as `n-back`, `k-drive`, or `questionnaire`.
The `PHASE` variable is used to denote the phase of the study, such as `train`, `test`, or `baseline`.
The `LEVEL` variable is used to encode the level within a combination of `STUDY` and `PHASE`.

For example, if `STUDY` is `n-back`, `PHASE` is `test`, and `LEVEL` is `4`, then the level would be denoted as `N-Back level 4`.
If `STUDY` is `n-back`, `PHASE` is `baseline`, and `LEVEL` is `2`, then the level would be denoted as `N-Back baseline 2`.

For both tasks, a baseline 1 is just 5 minutes of staring at a black screen. In `n-back`,baseline 2 involves receiving the same visual stimulus as the real test but with no task. In `k-drive`, baseline 2 and 3 involve visual stimuli of driving with random clicking and using the tablet by creating a random playlist, respectively.

Furthermore, the values from `PERFORMANCE` and `SUBJECTIVE` can be used to create alternative annotations.

Please refer to our publication for more detailed information about the dataset, including the experimental design and data collection procedures:

> Oppelt, M.P.; Foltyn, A.; Deuschel, J.; Lang, N.R.; Holzer, N.; Eskofier, B.M.; Yang, S.H. ADABase: A Multimodal Dataset for Cognitive Load Estimation. Sensors 2023, 23, 340. https://doi.org/10.3390/s23010340

1.2 Data Usage

[120]:
```
import h5py
import os
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import HTML
%config InlineBackend.figure_format = "retina"

# Suppress the warning
warnings.filterwarnings('ignore')
```

[121]:
```
# Sampling Rates
SKT_SR =  100
ECG_SR = 500
RSP_SR = 250
EMG_SR = 1000
EDA_SR = 500
```

```
EYE_SR = 250
```

1.2.1 Data Preparation

File Paths

```
[122]: SUBJECT_IDX = 0 # You can change this

       # Path to folder containing hdf5 files
       data_folder = "/path/to/the/dataset"

       # Get list of file names
       file_names = os.listdir(data_folder)
       file_names = list(filter(lambda x: x.endswith(".h5py"), file_names))

       # Select one file
       file_name = file_names[SUBJECT_IDX]
       file_path = os.path.join(data_folder , file_name)
```

Loading the data - We can load the full data frames in memory.

```
[123]: df_signals = pd.read_hdf(file_path, "SIGNALS", mode="r")
       df_performance = pd.read_hdf(file_path, "PERFORMANCE", mode="r")
       df_subjective = pd.read_hdf(file_path, "SUBJECTIVE", mode="r")
```

Loading parts the data - Alternatively, we can load only a subset of the data.

```
[162]: df_signals_ecg = pd.read_hdf(file_path, "SIGNALS", mode="r", columns=["STUDY",␣
        ↪"LEVEL", "PHASE", 'RAW_ECG_I'])
       df_signals_ecg.shape
```

```
[162]: (9148613, 4)
```

Column Names

```
[19]: # Get the column names as a list
      list(df_signals.columns)
```

```
[19]: ['TS',
       'STUDY',
       'PHASE',
       'LEVEL',
       'RAW_ECG_I',
       'RAW_ECG_II',
       'RAW_SKT',
       'RAW_RSP',
       'RAW_EMG',
       'RAW_EDA',
       'LEFT_GAZE_POINT_VALIDITY',
       'LEFT_PUPIL_DIAMETER',
       'LEFT_PUPIL_VALIDITY',
       'LEFT_GAZE_ORIGIN_VALIDITY',
       'RIGHT_GAZE_POINT_VALIDITY',
       'RIGHT_PUPIL_DIAMETER',
       'RIGHT_PUPIL_VALIDITY',
       'RIGHT_GAZE_ORIGIN_VALIDITY',
       'LEFT_GAZE_POINT_ON_DISPLAY_AREA_X',
       'LEFT_GAZE_POINT_ON_DISPLAY_AREA_Y',
       'RIGHT_GAZE_POINT_ON_DISPLAY_AREA_X',
       'RIGHT_GAZE_POINT_ON_DISPLAY_AREA_Y',
       'LEFT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_X',
       'LEFT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_Y',
       'LEFT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_Z',
       'RIGHT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_X',
       'RIGHT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_Y',
       'RIGHT_GAZE_POINT_IN_USER_COORDINATE_SYSTEM_Z',
       'LEFT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_X',
       'LEFT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_Y',
       'LEFT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_Z',
       'RIGHT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_X',
       'RIGHT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_Y',
       'RIGHT_GAZE_ORIGIN_IN_USER_COORDINATE_SYSTEM_Z',
       'LEFT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_X',
       'LEFT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_Y',
       'LEFT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_Z',
       'RIGHT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_X',
       'RIGHT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_Y',
       'RIGHT_GAZE_ORIGIN_IN_TRACKBOX_COORDINATE_SYSTEM_Z',
       'AU01',
       'AU02',
       'AU04',
       'AU05',
       'AU06',
       'AU07',
       'AU09',
       'AU10',
       'AU11',
       'AU12',
       'AU14',
       'AU15',
       'AU17',
       'AU20',
       'AU23',
       'AU24',
       'AU25',
       'AU26',
       'AU28',
       'AU43',
       'Subject',
       'label']
```

```
[20]: # Get the column names as a list
      list(df_performance.columns)
```

```
[20]: ['STUDY',
       'PHASE',
       'LEVEL',
       'AUDITIVE F1',
       'AUDITIVE MEAN REACTION TIME',
       'AUDITIVE PRECISION',
       'AUDITIVE RECALL',
       'VISUAL F1',
       'VISUAL MEAN REACTION TIME',
       'VISUAL PRECISION',
       'VISUAL RECALL',
       'F1',
       'PRECISION',
       'REACTION TIME',
       'RECALL',
       'SONGS RECALL']
```

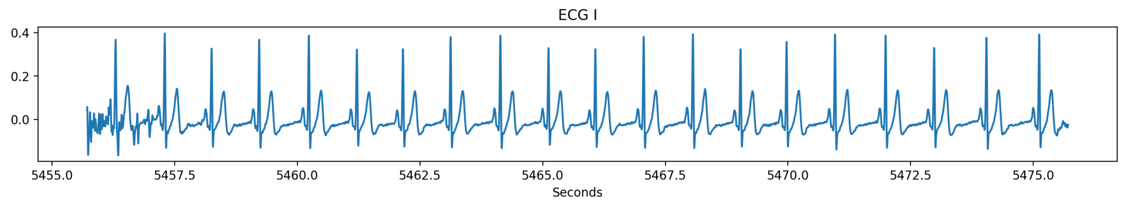
```
[21]: # Get the column names as a list
      list(df_subjective.columns)
```

```
[21]: ['STUDY',
       'PHASE',
       'LEVEL',
       'EFFORT',
       'FRUSTRATION',
       'MENTAL',
       'PERFORMANCE',
       'PHYSICAL',
       'TEMPORAL',
       'WEIGHT EFFORT',
       'WEIGHT FRUSTRATION',
       'WEIGHT MENTAL',
       'WEIGHT PERFORMANCE',
       'WEIGHT PHYSICAL',
       'WEIGHT TEMPORAL']
```

Getting the original signal - We have to drop the `NaN`s in the signal. - The eye tracker has still `inf` values after dropping the `NaN`s. These are the missing parts of the original signal (e.g. blinks).

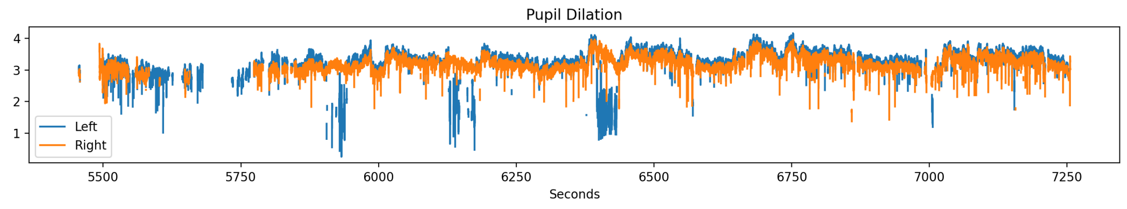
```
[163]: study_filter = df_signals["STUDY"] == "n-back"
```

[164]:
```
# Set figure size with a 16:6 aspect ratio
fig, ax = plt.subplots(figsize=(16, 2))

# Set the number of seconds to plot
seconds = 20

# Get the ECG signal data
ecg_signal = df_signals.loc[study_filter, "RAW_ECG_I"].dropna()

# Set the x-axis limits to the number of samples in the specified time range
num_samples = ECG_SR * seconds

# Plot the ECG signal
ax.plot(ecg_signal.index[:num_samples]/1000, ecg_signal[:num_samples]);
ax.set_title("ECG I");
ax.set_xlabel('Seconds');
```



[165]:
```
# Set figure size with a 16:6 aspect ratio
fig, ax = plt.subplots(figsize=(16, 2))

# Set the number of seconds to plot
start_second = 0
end_second = 60*30

# Get the EYE signal data - we replace inf with nan to get the original signal.␣
↪NaNs represent for example blinks or missing data
eye_left_signal = df_signals.loc[study_filter, "LEFT_PUPIL_DIAMETER"].dropna()
eye_right_signal = df_signals.loc[study_filter, "RIGHT_PUPIL_DIAMETER"].dropna()


#eye_left_signal = df_signals.loc[:, "LEFT_PUPIL_DIAMETER"].replace([np.inf],␣
↪np.nan)
#eye_right_signal = df_signals.loc[:, "RIGHT_PUPIL_DIAMETER"].replace([np.inf],␣
↪np.nan)

# Set the x-axis limits to the number of samples in the specified time range
num_samples_start = EYE_SR * start_second
num_samples_end = EYE_SR * end_second
```

```
# Plot the EYE signal
ax.plot(eye_left_signal.index[num_samples_start:num_samples_end]/1000,␣
 ↪eye_left_signal[num_samples_start:num_samples_end], label="Left");
ax.plot(eye_right_signal.index[num_samples_start:num_samples_end]/1000,␣
 ↪eye_right_signal[num_samples_start:num_samples_end], label="Right");
ax.set_title("Pupil Dilation");
ax.set_xlabel('Seconds')
ax.legend();
```



Annotation - We have to convert the `PHASE`, `STUDY` and `LEVEL` columns into our preferred annotation.

[167]:
```
# Map LEVEL, PHASE, STUDY to  annotations
CLASS_MAPPING = {"low": [[2, "baseline", "n-back"], [1, "test", "n-back"]],
                 "high": [[2, "test", "n-back"], [3, "test", "n-back"]]}

def get_label(study:str, phase: str, level: str):
    """Creates a label based on the specification of CLASS_MAPPING"""
    for key, label_list in CLASS_MAPPING.items():
        for spec in label_list:
            if [level, phase, study] == spec:
                return key
    return "null"
```

[168]:
```
get_label_vectorized = np.vectorize(get_label) # Create a vectorized version to␣
 ↪improve speed
df_signals["label"] = get_label_vectorized(df_signals["STUDY"],␣
 ↪df_signals["PHASE"], df_signals["LEVEL"])
```

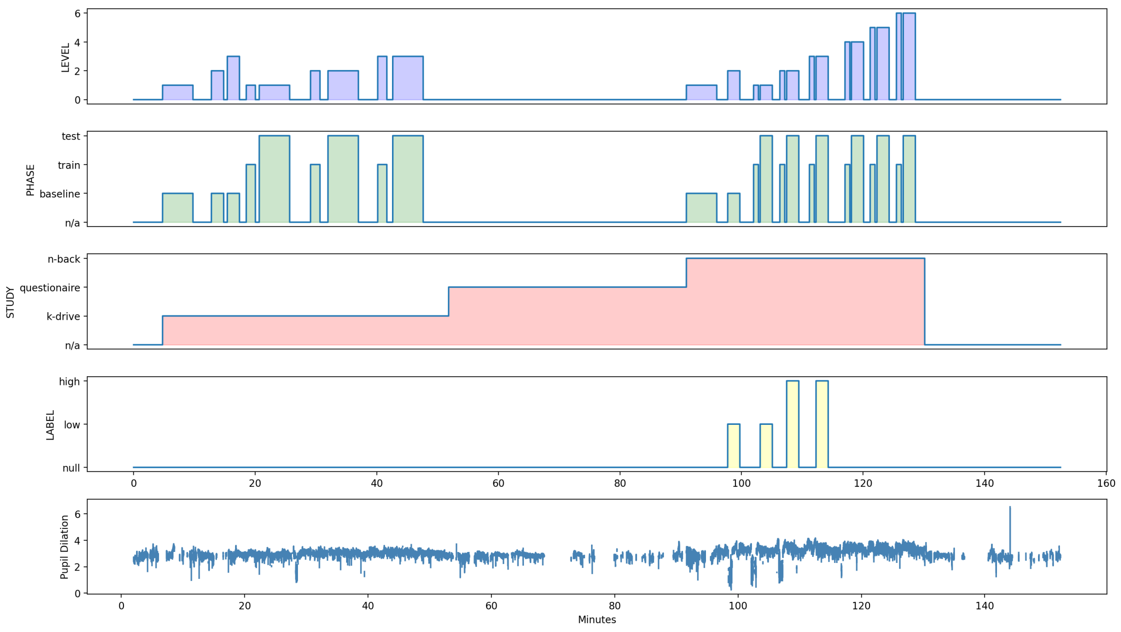
[169]:
```
df_signals["label"].unique()
```

[169]:
```
array(['null', 'low', 'high'], dtype=object)
```

Overview

[186]:
```
# Create a figure with a 16:9 ratio
fig, ax = plt.subplots(figsize=(16, 9))
```

```
# Create the first subplot
ax1 = plt.subplot(5, 1, 1)
ax1.plot(df_signals["TS"], df_signals["LEVEL"])
ax1.fill_between(df_signals["TS"], df_signals["LEVEL"], color='blue', alpha=0.2)
ax1.set_ylabel('LEVEL')
ax1.set_xticks([])

# Create the second subplot
ax2 = plt.subplot(5, 1, 2,)
ax2.plot(df_signals["TS"], df_signals["PHASE"])
ax2.fill_between(df_signals["TS"], df_signals["PHASE"], color='green', alpha=0.
 ↪2)
ax2.set_ylabel('PHASE')
ax2.set_xticks([])

# Create the third subplot
ax3 = plt.subplot(5, 1, 3, )
ax3.plot(df_signals["TS"]/(1000*60), df_signals["STUDY"])
ax3.fill_between(df_signals["TS"]/(1000*60), df_signals["STUDY"], color='red',␣
 ↪alpha=0.2)
ax3.set_ylabel('STUDY')
ax3.set_xticks([])

# Create the fourth subplot
ax4 = plt.subplot(5, 1, 4, )
ax4.plot(df_signals["TS"]/(1000*60), df_signals["label"])
ax4.fill_between(df_signals["TS"]/(1000*60), df_signals["label"],␣
 ↪color='yellow', alpha=0.2)
ax4.set_ylabel('LABEL')

# Create the fiveth subplot
eye_left_signal_full = df_signals.loc[:, "LEFT_PUPIL_DIAMETER"].dropna()
ax5 = plt.subplot(5, 1, 5, )
ax5.plot(eye_left_signal_full.index/(1000*60), eye_left_signal_full,␣
 ↪color="steelblue")
ax5.set_xlabel('Minutes')
ax5.set_ylabel('Pupil Dilation')

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.2)
plt.tight_layout()
```


LEVEL
PHASE
test
train
baseline
n/a
STUDY
n-back
questionaire
k-drive
LABEL
high
low
null
Pupil Dilation
Minutes