Huberty, Scott
seh33@uw.edu

Eye-tracking data support in MNE-Python

Including functions for preprocessing, visualization, and analysis

About Me

Name: Scott Huberty
Resume: https://scotterik.com/CV
Email: seh33@uw.edu
Github: https://github.com/scott-huberty
Time Zone: EST

University: McGill University
Program: Integrated Program in Neuroscience
Degree: PhD
Expected Graduation: December 2023

Bio: I am a doctoral candidate in Neuroscience at McGill University. I am interested in conducting reproducible research that utilizes electrophysiology and eye-tracking.

MNE-Python Code Contributions

March 2023: #11152: Added support for eye-tracking data, and a reader for *Eyelink* devices.
July 2022 (MNE-BIDS): #10323: Fixed bug where *stim* channels were being written to the *electrodes.tsv* sidecar file.
July 2022: #10898: Corrected the methods for setting channel names and locations when reading EGI MFF files.
May 2022: #10676: Added informative instructions for pip installing *extra* dependencies if user hit import errors for packages like Edflib-Python, Scikit-learn, and Pandas.
March 2022: #10304: Bug fix for setting *meas_date* in *RawMFF* objects

Huberty, Scott
seh33@uw.edu

Project information

Sub-org: MNE-Python

Project Abstract:
Eye-tracking devices are widely used in psychology and neuroscience labs. However, the Python community has yet to coalesce around a well-supported package for analyzing eye-tracking data. This project proposes to continue the work started in 2022 by myself and colleagues to support reading, visualizing, and analyzing eye-tracking signals in MNE-Python.

Detailed Description:
Eye-tracking devices are widely used in neuroscience research. Studies of visual perception and attention in particular are increasingly integrating eye-tracking and EEG/MEG devices, in order to gain a better understanding of these processes and their neural correlates. However, the Python community has yet to coalesce around a well-supported package for analyzing eye-tracking signals. Furthermore, there are no Python tools available for co-analyzing eye-tracking and EEG/MEG/fNIRS data.
This project proposes to continue the work started in 2022 by myself and colleagues to support reading, visualizing, and analyzing eye-tracking signals in MNE-Python. We have already laid the groundwork for reading eye-tracking data into MNE-Python (PR #11152). Therefore, this GSoC project can focus on developing the preprocessing, visualization, and analytical methods that are often required for eye-tracking research.
There are several key advantages to integrating eye-tracking support into MNE-Python. Firstly, eye-tracking data provides important information about the cognitive processes underlying perception and behavior, including visual attention, decision-making, and memory. By incorporating eye-tracking data into MNE-Python, researchers would be able to analyze both neural and eye movement data within the same software environment and gain a more complete understanding of these processes and their neural correlates.
Secondly, MNE-Python already has a well-designed infrastructure for handling physiological data. There are several functions for signal-processing and statistical analyses that can be applied to eye-tracking data.
Finally, MNE-Python has a strong track record of implementing the best software-development practices, including continuous integration and test-driven development, and MNE has a large community of developers and users. This will ensure that this new feature will continue to be used and maintained well after this project is finished.

The bulk of the proposed work will fall the following categories:

Huberty, Scott
seh33@uw.edu

I/O

There is meta-information about the eye-tracking recording that is not currently being read into MNE but which may be needed for later preprocessing or visualization functions. Such information includes the calibration information, the sampling coordinate system ("gaze-on-screen" or "eye-in-head"), and the sampling coordinate unit (i.e. pixels for "gaze-on-screen" data, radians for "eye-in-head" data). If there is a suitable key in the MNE *info* structure to store this information, it should be stored by MNE.

Preprocessing

There are several commonly used preprocessing routines in eye-tracking research, for example interpolating eye-position samples during "blinks", pupil unit conversion, and pupil deconvolution. These preprocessing routines can be developed as methods of eye-tracking *Classes* in MNE-Python, and/or as stand-alone functions in the MNE-Python preprocessing module.

Visualization

The now unmaintained *pyeparse* repository provided a number of visualization functions for eye-tracking data that can be ported and refactored into MNE. This includes functions to plot the eye-tracking session calibration, and a heatmap plot of positional data on the display screen. These visualization functions will help researchers to explore their data and share their findings.

Tutorials / Documentation

Any new functionality should be documented, and informative tutorials should be added to mne/tutorials/preprocessing. In particular, a tutorial should be built that walks through the process of aligning eye-tracking and EEG/MEG signals from an integrated recording.

Weekly Timeline

Below is my proposed timeline for the project. The project is planned such that the foundational work will be completed first, so that it will be available to use in later development. Further, this project plan is meant to allow for *incremental* integration into the MNE-Python codebase. Each sub-goal is contained within a 1 or 2 week timeline, and will encompass development, documentation, and unit-testing, so that a *Pull Request* can be opened to merge the sub-goal into MNE-Python's main branch after passing continuous integration and review.

Each sub-goal is labeled with one of the following categories:

I/O **Preprocessing** **Visualization** **Tutorial**

Prior to Week 1

A decision should be made regarding which meta-data from eye-tracking recordings will be needed for later preprocessing and visualization functions, So that this information can be read

and stored during file I/O. There is a Pull Request in development for adding an eye-tracking BIDS standard, which may be able to help inform our decision-making process regarding what meta-data to read into MNE. Before the official GSoC start date, I will correspond with my mentors and other developers to reach a decision on which meta-data from eye-tracking recordings is important to read into MNE-Python, so that I can begin this work during **week 1**

Week 1

Store necessary eye-tracking meta-data in the Info structure I/O

This week will focus on extracting the needed meta-data during I/O, and storing this information in the MNE *Info* structure. This week's work will include development, unit-testing, and documentation so that a Pull Request can be opened prior to week 2.

Week 2

Pupil unit conversion to Meters Preprocessing

In Eyelink systems, Pupil data are reported in *arbitrary units*. Since **1)** MNE prefers to internally store data according to an SI unit, and **2)** Many eye-tracking researchers prefer to convert their pupil data to millimeters, a preprocessing function to convert data to meters (the SI unit of length) should be added. This function would require information on the head distance from the display screen. This information may be available in the eye-tracking recording (For *Eyelink* systems that use a head-sticker for that purpose), or could be provided by the user as a keyword argument (for “fixed” setups that use a mounted chin-rest, such that the monitor location is stationary relative to the head).

Week 3

Refactor mne.io.raw.plot **Visualization**

The coordinate unit of eye-tracking data will differ depending on the derivative of the data that is reported from the recording (for example, estimated pixel position of the gaze on the screen, versus the eye-angle relative to the head.).

Currently, MNE assumes that the eye-position data are represented in *radians*, and sets the default plot scalings accordingly. However, often-times an eye-tracker recording will report the estimated *pixel* position of the gaze on the screen (these data range from 0 - 1000, and are considered as *arbitrary units* by MNE-Python). In this case, the eye-position channel trace will not be legible on plots, given MNE’s current default scalings. Based on discussion in #11152, it may be preferable to create a dictionary mapping between units and their default scalings, and refactor mne.io.raw.plot to use this unit-to-scaling mapping when setting the *ylim* scaling for plots.

Week 4

Build function for interpolating eye-position data during blinks Preprocessing

In data from eye-tracking recordings, periods of blinks will result in NaN values for eye-position data. It is common in eye-tracking research to interpolate the NaN values during blink periods, using information from the position data before and after the blink. A function can be added to mne.preprocessing.eyetracking to interpolate blinks.

Huberty, Scott
seh33@uw.edu

Week 5
Build Calibration and eye-position plotting functions Visualization
The aforementioned *pyeparse* repository contains functions for plotting the eye-tracking calibration, and plotting eye-position data across trials as a heatmap. This week will focus on porting these functions to MNE, documenting, and testing them.

Week 6 (mid-term evaluation). **Visualization Tutorial**
Build documentation and tests for visualization functions from week 5 Visualization
The visualization functions developed in week 5 should be documented in one of the existing eye-tracking tutorials, or in a new tutorial. Unit tests must also be developed and integrated to CI for each of the new functions.

Week 7
Build tutorial for aligning integrated eyetracking-EEG signals, and analyzing fixation related potentials (FRPS) Tutorial
Much of the motivation for supporting eye-tracking data in MNE-Python stems from the fact that many research labs are conducting integrated eye-tracking and EEG/MNE recordings. This week will focus on building a tutorial (using real data) that aligns, concatenates, and analyzes the eye-tracking and EEG signals from an integrated recording. This will require collecting an integrated EEG eye-tracking recording in my lab that can be used for the tutorial.

Week 8
Pupil deconvolution Preprocessing
Methods for estimating the deconvolved pupil response were described by McCloy and colleagues, 2016. The purpose for this deconvolution is to separate the pupil dilation changes driven by an attentional (or some other intrinsic) process of interest, from external nuisance factors that can also affect pupil size (such as luminance). The aforementioned *pyeparse* repository contains some functions for obtaining the pupil kernel which may be useful for deconvolution. This week will focus on porting, testing, and documenting that code.

Week 9
Build a tutorial that applies Pupil deconvolution to a dataset Preprocessing Tutorial
This week will focus on applying the pupil deconvolution functions to real data, and documenting this in a tutorial. This may require cutting a test recording in-lab that can be used for this purpose.

Week 10
[Planned Vacation]

Week 11
Code-Freeze

Huberty, Scott
seh33@uw.edu

Development should be completed by week 11, so that the last two weeks of this project can focus on reviewing the tests and documentation, and addressing any bug-reports that have been reported thus far.

Week 12

Wrap-up

Review code, tests, and address any bug-reports. Build report of the project accomplishments.