## OVERVIEW

OMtools is a toolbox for the reading, viewing, analysis and presentation of eye-movement data.

Use the function "datstat" (GUI for "rd") to read ober2, plain-text ASCII, or binary-formatted data files. It can import EDF data, (using "edf2bin", which requires the edf2asc function from SR-Support's web site). You can also create modules to import your own files. Guidelines can be found in the 'rd' directory.

BONUS: I have added support to create experiments using SR-Research's EyeLink trackers, allowing you to set tracker parameters, present stimuli and record data. This requires the installation of the Pyschophysics Toolbox (www.psychtoolbox.org). If you are interested, contact me and I will send you the base tool folder (EL_suppt), and examples of experiments created using it.

## INSTALLATION

The OMtools folder contains a script, imaginatively named "install_omtools.m", that you must drag and drop into the command window to execute. It will first check for previously existing copies of OMtools, giving you the option to disable them before installing the most recent version. A copy of the current OMtools and zoomtool preference folders (if any) will be copied to the new installation.

OMtools will be installed to the MATLAB folder in your home directory's Documents folder. This is the preferred location for non-MathWorks toolboxes.

The installer offers two ways to install OMtools: ~~using the pre-packaged MATLAB Add-ons toolbox file, OMtools.mltbx~~, or the conventional method of copying the toolbox folder to a desired location and adding it to your MATLAB paths.

Both methods have the same functionality; MATLAB's add-ons feature automatically takes care of package management, including setting paths, and uninstallation. It is pretty much "set it and forget it". The choice is yours. In either case, omtools_installer.m will properly set and save all necessary paths.

After the installer has finished, it will type "omstart" at the command line to test the status of the install. It should report the locations of the toolbox and preference folders (saved outside the main toolbox so that upgrades don't overwrite your custom settings). If all is well, it will report "OMtools is ready to use."

## READING DATA

OMtools allows you to offset and scale data when it is read in, without modifying the original data file, which remains untouched. It can also read data files that do not have any interpretation information (sample frequency, channel names) in their headers. It does this through the use of "adjustbias" files, plain-text files that store

all this information for a single file or group of related files. Adjust bias files are generated using "biasgen", either directly from the command line, or when attempting to read data that does not have one. Biasgen will prompt you to enter the information necessary to create the file.

OMtools can read data in a variety of formats, including common types as plaintext, raw binary, ober2 (still a bunch out there) and Eyelink EDF, after importing using the "edf2bin" function (which will automatically create an adjust bias file for you).

Data files are read using "datstat" (a graphical frontend for the "rd" function) which brings up a file selection dialog, allowing you to easily navigate to the folder where your data files are stored. It keeps track of the location of the last successfully loaded file (which might not be your current directory) and uses it as the starting point for the next file to load. It then looks for an adjust bias file that matches the file being loaded. If it can not find one, it will prompt you to either search for, or create one (with default offset and scale values -- 0 and 1 respectively – that will not modify your data).

As the data loads, datstat/rd will determine the number of channels, their names, the sampling frequency, and number of samples. These are stored into an eye-movement data structure (type 'help emData.m' to learn about the structure) that will be saved into the base workspace as the name of the file you loaded. Datstat will also separate the individual channels (e.g. lh,rh,lv,rv,st,sv) and add them to the base workspace.

You might be done at this point. Plot your data and see if you are happy with its calibration. If it's coil data, you probably will be, though you might need to shift 0°. Hurray for methods that can be pre-calibrated without need for a subject!

But if it's IR- or video-based (methods that must be calibrated specifically for each subject), you might need to shift to match the subject's precise 0°, and perform scaling. Is this strictly necessary? Possibly not – you'll have to use your own judgment. Even trackers such as the Eyelink that perform their own internal calibrations may be off slightly. Sometimes more than slightly, especially when recording from a subject who has difficulty complying or holding still.

## CALIBRATION

If you want to make your own adjustments, run "cal". See "Cal instructions" in the OMtools documentation folder for more detail.

After you calibrate the data, you need to modify the adjust bias file to incorporate the new offset and scaling factors. Cal outputs these values to the command window, formatted for easy copy and paste. Once you update the file, clear and re-read it to verify that it looks as expected.

## A BIT MORE ABOUT EDF2BIN

Edf2bin calls SR-Research's "edf2asc" executable, which is available from https://www.sr-support.com/forum/downloads/eyelink-display-software (free account required).

In addition to a .bin file, and an adjust bias file, edf2bin will generate an _events.txt file (all Events recorded) and an _msgs.txt file (all Messages sent). It will also create an _results.mat file that contains a record of saccades, blinks and fixations detected by the tracker.

If there are multiple trials in the EDF file, they will be saved separately, with an appended number (e.g. aaa_1.bin), with appropriate entries made in the bias file.

## OTHER STUFF WORTH KNOWING

zoomtool

utils

analysis

graphing

*Mea Culpa*

This represents an accretion of 20+ years of code, some quite primitive* and some that's as up-to-date as next week. I have *tried* to clean and modernize the most critical functions (e.g., all data loading) to make them more easily understood and expandable. However, there is still a lot of embarrassingly klunky old code that I have left in place to serve either as cautionary tale or the starting point for modernization (e.g., pre-data-structures scripts for concatenating data from multiple recordings for analysis).

* before MATLAB had modern features like object handles and data structures, I had to create functional equivalents using linked lists and other kludges, and had to keep most data and variables in the base workspace and rely heavily on global variables. Good times, good times…