Written by: Jonathan Jacobs
Latest version: 23 Jan 2022

OVERVIEW

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

The function "datstat" (a GUI for "rd") lets you read from a variety of data formats, including plain-text ASCII, or binary-formatted data files. OMtools can import and save EDF files, (using the function "edf2bin", which requires the edf2asc executable function from SR-Support's web site). You can also create modules to import your own file formats. Guidelines can be found in the 'rd' directory.

OMtools supports the creation of experiments using SR-Research's EyeLink trackers, allowing you to record data, set tracker parameters, present multiple types of stimuli, and communicate with external devices. This requires the installation of SR's development kit (sr-support.com), Pyschophysics Toolbox (psychtoolbox.org), and the OMrecord and EL_suppt toolboxes.

COMPATIBILITY

OMtools should run on Mac, Linux and Windows systems running at least MATLAB 2016b or later. Primary development has been for the Mac platform, with less testing on the others.

DOWNLOAD

OMtools is available from the software and models page of the OMlab.org website:
http://omlab.org/software/software.html

Direct link (stable): https://app.box.com/s/1zj33g7337ttwe223nrmayd46ic77a0s
Direct link (dev): https://app.box.com/s/w1s4og778cve8g95vdu6ys9o1tvekq8s
Direct link (both): https://app.box.com/s/n68e0sq0i63p3r1st2a3cs8afcv0somg

EL_suppt (stable): https://app.box.com/s/2zwtf8qrjz9mgc0f0nmokqwc78d04gp6
EL_suppt (dev): https://app.box.com/s/0gdb1mwqu3e68iufb9wiinu7mz52ll69

OMrecord (stable): https://app.box.com/s/fa3qa3gp52ysyo0xoo7obv07onh8rokj
OMrecord (dev): https://app.box.com/s/8iq04fdd6udsvhi3bu6f7wobvyalf2zs

There are two branches, "stable" and "dev". The stable branch contains the current working release. The dev branch is mostly the same, but includes updates made after the stable release. These changes may include new functionality, or just minor cosmetic or documentation improvements. These have not undergone as much testing, but have been working in limited release. There is a changelog document for each branch.

INSTALLATION

The OMtools folder contains a script, imaginatively named "install_omtools.m", that you can 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 preference folder (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 will next properly set and save all necessary paths, and give you the opportunity to have OMtools available every time MATLAB launches.

After the installer has finished, it will run "omstart.m" 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."

If you have trouble using the installer, you can manually copy the OMtools directory into your personal MATLAB directory (usually located in the "Documents" folder of your home folder), and then run "omstart.m" (located in the "omtoolsdirs" folder) by dragging it into the MATLAB command window.

Each time OMtools is started, it will check the online repository for the latest stable and development versions.

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 by using "adjustbias" files , which are human-readable (and modifiable) plain-text files that store all this information for a single file or group of related files. Adjustbias 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 and add them to the base workspace. OMtools understands the following default channels: Left and right horizontal, vertical, and torsional eye position, horizontal and vertical target and head position (names: lh, lv, lt, rh, rv, rt, st, sv, hh, hv), and can be expanded to include your own custom channels.

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 your data is 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.

VERGENCE CALIBRATION

There is now the rudimentary ability to calibrate multi-distance targets, using the GUI function "vrgcal". Currently (Jan 2022) this is limited to straight-ahead targets, but will soon/eventually be expanded to work with off-angle targets as well. See "Vergence cal instructions" in the OMtools documentation folder for more detail. Note that you should perform a uniplanar calibration before performing the vergence calibration.

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 file (named "xxx#_events.txt" where xxx# is the subject ID and record number) containing all Events recorded by the tracker, and a messages file ("_msgs.txt") containing all messages sent to the tracker. It will also create a results file ("_results.mat") containing 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. "aaa1_1.bin", "aaa1_2.bin", etc.), with appropriate entries made in the bias file.

OTHER STUFF WORTH KNOWING

Help is available for most functions, including mouseover tooltips in the GUIs. I have also attempted to include detailed comments about the philosophy and implementation in the code.

Zoomtool:
An interactive data exploration tool.

Utils:
A collection of functions for examining and modifying data, including: differentiation to calculate velocity and acceleration; scaling and offsetting; filtering; basic curve fitting; detecting/deleting blinks and dropouts; converting the timebase between sample and seconds; controlling MATLAB windows; components for loading and parsing text files; and performing enhanced numerical and string/character operations.

Analysis:
Functions (including GUI apps) for performing higher-level operations on data: converting between distances and angles; calculating visual acuities; detecting foveation; examining peak-velocity/duration vs amplitude; calculating nystagmus acuity (the NAFX); and more.

Graphing:
Functions for creating and modifying data plots, including: text, line, surface and axis properties (including size, color, content and location); annotations; copying and pasting graph objects; extracting x,y,z values from objects; changing axis limits, styles, and viewing angle; finding specific windows, including detecting the frontmost graph; changing window order (send front window to the back); stripping or adding color; and an expanded basic colors palette.

Labels:
Tools for placing text labels on graphs, axes, and titles; drawing foveal extents or other radial information (in time and x-y plots). Many of these are simple shortcuts for specific sets of values for particular plot types.

Eyeballs3D:
Graphical interface for creating movies of animated eye-movement data, including time-based and phase-plane plots.

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…