PuPL Manual

Isaac Kinley

July 2019

Contents

1 Getting started

PuPL is an Octave-compatible library of Matlab functions for processing eye tracking data and a user interface that runs these functions with a click.

1.1 Initializing

Change Matlab's working directory to the PuPL folder (which should contain `pupl.m`, `LICENSE.md`, etc.). You can do this either using the "Current Folder" panel or from the Command Window using the `cd` function.

Run `pupl init` in the Command Window to add the source code to Matlab's path, initialize the global `eyeData` variable, initialize the user interface, and load whichever add-ons are in the add-ons/ folder. Any of these last 3 steps can be skipped by adding `noGlobals`, `noUI`, and/or `noAddOns`, respectively, as command-line arguments to `pupl init`.

1.2 Importing raw data

Under `File > Import`, you will see options for importing raw data. Note: if you import raw data from a BIDS-formatted folder, event logs will not also be imported.

1.3 Manipulating data

Loaded data will appear on the user interface. Unselected data is ignored by functions run using the user interface. To delete data from Matlab's workspace, go to `Edit > Remove datasets`. Data is also accessible in the Command Window through the global variable `eyeData`. If you process data in the command window and want to update the user interface, run `pupl redraw`.

1.4 Importing raw event logs

After you have loaded some data, you will be able to attach event logs to it. Go to `Trials > Event logs > Import` to see your options for importing raw event logs.

1.5 Synchronizing event logs and eye tracker data

When raw eye tracker data is loaded, event onsets are recomputed such that an event coinciding with the first data sample will be assigned an onset of zero. This means that timestamps in event logs cannot be directly copied to eye data. Furthermore, the clock used to measure timestamps in the event log and the clock used to measure timestamps in the eye data may drift relative to one another. To solve this problem, we use linear regression to find a mapping between timestamps in the two records. To begin, go to `Trials > Event logs > Synchronize with eye data`. A window will open for you to specify a correspondence between event types in the two records (for example, there may be sync markers sent by stimulus presentation software to both the eye tracker and an event log). After this, you will be able to select the events from the event logs that should be copied to the eye data, whether they should be copied under different names, and whether the pre-existing event records in the eye data should be deleted.

2 Visualizing data

2.1 Plotting continuous datastreams

It is a good idea to visualize data throughout processing to see how it is being altered. Under `Plot > Plot continuous` there are tools for plotting continuous datastreams. Use the arrow and page keys to move the window of data displayed.

2.2 Plotting gaze coordinates

To see how gaze positions are distributed for each recording, go to `Plot > Gaze scatterplot`.

2.3 Plotting pupil size

To see how pupil size measurements are distributed for each recording, go to `Plot > Pupil diameter histogram`

2.4 Visualizing pupil foreshortening error

The pupil foreshortens as it turns away from the eye tracker, being measured as smaller than it really is. E.g. if the eye tracker is placed below a computer screen, the pupil will tend to be measured as smaller when looking at the top of the screen and either side. This will appear as a (roughly) linear trend in a plot of pupil size vs gaze y coordinate and a (roughly) quadratic trend in a plot of pupil size vs gaze x coordinate. Under `Plot > Pupil foreshortening error`, there are tools to examine whether there is a systematic relationship between gaze location and measured pupil size. The tools explained in 3.10 can correct for such a relationship.

2.4.1 Plotting pupil size as a function of single gaze dimension

To plot pupil size as a function of either gaze x or gaze y coordinate, go to `Plot > Pupil foreshortening error > Pupil size vs gaze [x/y]`.

2.4.2 Plotting pupil foreshortening error surface

To visualize pupil size across the whole gaze field, go to `Plot > Pupil foreshortening error > Error surface`. This divides the gaze field into a grid and computes the mean pupil size for points falling within a square centered on each node.

2.5 Plotting individual trials

Once trials have been extracted, they can be plotted using `Plot > Plot trials`. Use the arrow keys to view the next/previous trial.

2.6 Plotting trial sets

Once trials have been merged into trial sets (as explained in 5.6), they can be plotted using `Plot > Plot trial sets`. The meanings of the line colours are the same as in the continuous pupil diameter plots. The shaded error bars represent one standard error of the mean.

3 Processing raw data

All functions for preprocessing raw data can be found under `Preprocess`. To undo all processing, go to `Tools > Revert to unpreprocessed data` (this also deletes the processing history).

3.1 A note on specifying durations

You may use units and arithmetic to specify lengths of time. Valid units are as follows:

`ms`: milliseconds
`s`: seconds
`m`: minutes
`dp/d`: datapoints

For example, `1s + 2 dp - 100 ms - 2d` translates to one second plus two datapoints minus 100 milliseconds minus two datapoints.

3.2 A note on specifying relative quantities

You can specify relative quantities by writing any valid Matlab expression using `$` to refer to the data that. E.g. `max($)` would return the max. You can also use shortcuts to compute common statistics as follows:

`‘m`: mean
`‘d`: median
`‘s`: standard deviation
`‘v`: variance
`‘i`: interquartile range
(Note that the backticks in the above are to avoid conflicts with other Matlab expressions)
`%`: percentile (e.g. `15%` computes the 15th percentile)

E.g. `‘m - 2‘s` would compute the mean minus two standard deviations.

3.3 Converting between pupil area and diameter measurements

Some eye trackers record pupil area, while others record pupil diameter. To convert between these, go to `Process > Convert pupil size`.

3.4 Centering and scaling data

To center (e.g. by median or mean) and/or scale (e.g. by standard deviation, mean, or median) pupil data, go to `Process > Normalize data`.

3.5 Trimming data

3.5.1 Trimming extreme pupil diameter measurements

Go to `Process > Trim data > Trim extreme dilation values` to remove pupil size measurements that are too high or low. In the window that opens, you can specify cutoff points as explained in 3.2. Note: the gaze measurements corresponding to rejected pupil size measurements will also be removed. If you plan to run the same pipeline on multiple participants, it is best to specify relative cutoffs since different participants will likely have different average pupil size measurements.

3.5.2 Trimming extreme gaze measurements

Go to `Process > Trim data > Trim extreme gaze values` to remove gaze measurements that are too extreme (e.g. off-screen or too far from a fixation cross). You can specify cutoff points as explained in 3.2. Note: the pupil size measurements corresponding to rejected gaze measurements will also be removed. If you plan to run the same pipeline on multiple participants, it is best to create absolute gaze cutoffs since screen x and y values are likely to have the same meaning across participants.

3.6 Trimming isolated samples

Go to `Process > Trim data > Trim isolated samples` to remove little blips of data that are unlikely to be meaningful or accurate measurements. See below to understand how to specify the max length of these islands of isolated data.

3.7 Trimming blink-adjacent samples

The pupil is partially obstructed shortly before and after the eye fully closes during blinks. This makes samples recorded near blinks unreliable. To trim these, go to `Process > Trim blink-adjacent samples`. See 3.1 for an explanation of how to specify durations in the dialog boxes that appear.

3.8 Filtering data

To apply a moving mean or median filter to pupil size or gaze data, go to `Process > Moving average filter`.

3.9 Saccades and fixations

3.9.1 Identifying

Algorithms available for identifying saccades and fixations can be found under `Process > Identify saccades and fixations`. Note: data points themselves are not labelled as saccades or fixations, only the periods of time between them. The justification for this is as follows: if gaze samples 1, 2, and 3 are all

at the same coordinates and gaze samples 4, 5, and 6 are at a far-away coordinate, which point ought to be labelled as a saccade? Clearly the participant was fixating at points 1, 2, and 3 and then again at points 4, 5, and 6, and the saccade took place between points 3 and 4.

3.9.2 Mapping to fixations

After identifying fixation periods, the data during these periods can be reassigned to their centroids (spatial means) using `Process > Map gaze data to fixation centroids`.

3.10 Pupil foreshortening error correction

For an explanation of pupil foreshortening error, see 2.4. It can be corrected using the options under `Process > Pupil foreshortening error correction`.

3.10.1 Geometric PFE correction

The pupil foreshortening error can be corrected using straightforward geometry if the coordinates of the experimental setup are provided. First, gaze coordinates need to be converted from units of pixels to units of millimeters; go to `Process > Pupil foreshortening error correction > Convert gaze units from pixels to millimeters` to do this. Then, you can add the coordinates of the experimental setup under `Process > Pupil foreshortening error correction > Add coordinates of experimental setup`. A dialog box will appear explaining the coordinate system to use when providing these coordinates. Once you have done so, go to `Process > Pupil foreshortening error correction > Geometric PFE correction` to apply the geometric correction.

3.10.2 Detrending PFE correction

Options to correct for a linear relationship between pupil size and gaze y position and a quadratic relationship between pupil size and gaze x position can be found under `Process > Pupil foreshortening error correction`. Note that at the time of writing these tools have not been thoroughly tested and use a first approximation of the PFE to avoid having to measure the experimental setup.

3.11 Interpolating missing data

To linearly interpolate missing pupil size and gaze data, go to `Process > Interpolate missing data`. Dialog boxes will open asking for the maximum length of time and the maximum distance to interpolate over (see 3.1 and 3.2 for explanations of how to provide input for these, respectively). Note: it is a good idea to be conservative when specifying the maximum gaze distance to interpolate over since, if saccades take place during long periods of missing data, linear interpolation will make these look like periods of smooth pursuit.

3.12 Averaging left and right pupil size

To average the left and right data streams, go to `Process > Merge left and right diameter streams`.

3.13 Adding BIDS information

To add subject, session, task, and acquisition information for saving data to the BIDS format, go to `BIDS > Add BIDS info`.

4 Saving data

To save data, go to `File > Save`. A separate filesave dialog will open for each active recording. The `.eyedata` extension on saved recordings is merely an alias for version 6 `.mat` files.

4.1 Batch saving data

To save all active data to the same folder using filenames corresponding to recording names, go to `File > Batch save`. This will open a single folder selection dialog.

4.2 Saving data to BIDS format

If you have data loaded and active, you can save it in a BIDS-formatted project directory.

4.2.1 Saving raw

To create and populate the `raw/` folder in your BIDS-formatted project directory, go to `BIDS > Save > Save raw from current data`.

4.2.2 Saving sourcedata

To create and populate the sourcedata/ folder in your BIDS-formatted project directory, go to `BIDS > Save > Save sourcedata of current data`. Note: this re-loads the raw data using the `getraw` method of the data structure, which can be slow depending on the format of the raw data.

4.2.3 Saving derivatives

To save processed data in a folder withing the `derivatives/` folder of your BIDS-formatted project directory, go to `BIDS > Save > Save current data as derivative`. Note: this folder will have the same internal structure and filenames as the `sourcedata/` folder and can be loaded using the function explained in 4.3.

4.3 Loading BIDS sourcedata

Under `BIDS > Load sourcedata`, you will have the option to load sourcedata from a BIDS-formatted directory. Note: if you use this option, any event logs corresponding to the data will be imported as well. You can also use this menu option to load data from a derivatives/ subfolder that was saved using the function explained in 4.2.3. Simply provide said subfolder as the sourcedata folder.

5 Event-related pupillometry

5.1 Defining compound events

Often, we want to analyze responses to specific serial combinations of events (e.g. a trial of a certain type followed by a response of a certain type). To define these "compound events", go to `Trials > Define higher-order events`. In the dialog box that appears, specify the name of the compound event you wish to define. In the next dialog box, specify the "time-locking" events (these are the events whose onset times will be used as the onset times of your compound event). Next, specify "secondary events" that, if they occur in some serial position and/or time window relative to the time-locking events, will indicate the presence of your compound event. Next, you can specify a time window centered on the time-locking events in which the secondary events must occur (e.g. Start: `-2s`, End: `0s`; see 3.1 for an explanation of how to specify this window; if none is specified,, itdefaults to the entire duration of the recording). Next, you can specify the relative serial positions at which the secondary events must occur (these can be specified using Matlab's colon operator, e.g. `-3:-1`. Next, you can specify whether compound events should be marked based on the presence or the absence of the secondary events within the specified window and serial position list. Finally, you will be asked if you want to overwrite the time-locking events. If you select "Yes", your compound event's name will replace its time-locking event's name. If you select "No", the name of your compound event will be appended to the name of its time-locking event.

5.1.1 Example

If the time-locking events are `te1` and `te2` and the secondary events are `se1` and `se2`, and the window is specified as `-3s` to `0s`, and the relative serial positions are specified as `-3:-1`, then compound events will be marked as: each instance of `te1` or `te2` where an instance of `se1` or `se2` occurred at most 3 seconds prior but not afterward, and occurred as the immediately preceeding event, the one before the immediately preceeding event, or the one before that.

5.2 Computing reaction times

To compute reaction times, go to `Trials > Compute reaction times`. You will be asked to specify which events indicate the beginning of a trial and which events indicate a response. If there is no response event between one trial onset event and the next, the reaction time defaults to NaN. Next, you will be asked which events reaction times should be written to (e.g. you may want to record reaction times with either the onset or the response, or with another event nearby) a time window within which these events must occur, and whether this window is centred on the onset events or the response events.

5.3 Defining trials

To extract trial data, go to `Trials > Fragment continuous data into trials`. Trials are defined and extracted relative to events in the eye data. For example, if you wanted to examine pupillary response to an event called `showImage`, you would select this event name in the window that appears next. The durations of trials can be specified in the next dialog window. For example, to extract data from 200 milliseconds before the occurrence of events until 2 seconds after, you would enter `-200ms` and `2s` in the next dialog window. These durations can be specified as explained in 3.1.

5.4 Baseline-correcting trials

Due to variability in pupil size measurements, it is necessary to baseline correct trial data, either by subtracting the baseline mean or by computing the percentage change from baseline. To do this, go to `Trials > Baseline correction`. The mapping from baseline periods to trials can be one-to-one (e.g. baselines periods are the 200 ms before each event onset), one-to-all (e.g. one single baseline period occurs at the beginning of the experiment), or one-to-some (e.g. one baseline period occurs before a block of 5 trials). If the one-to-some mapping is selected, trials are baseline-corrected using the most recent baseline period. The durations of baseline periods can be specified as explained in 3.1.

5.5 Rejecting trials

Trials can be removed from further analysis using `Trials > Trial rejection`. Trials can be rejected according to the proportion of data missing, the presence of extreme pupil size measurements, or reaction time. It is also possible to reject trials on the basis of the amount of data missing within a specific window (e.g. the baseline period) using `Trials > Trial rejection > Reject by proportion missing data within window`.

5.6 Merging trials into sets

It is useful to average together trials of multiple types to get reliable estimates of pupillary response. E.g. you may want to examine responses to both a

particular stimulus and a category of stimuli to which it belongs. To define sets of trials for later analysis, go to `Trials > Merge trials into sets`. Note: even when you do this the first time, you will be asked if you want to overwrite the pre-existing trial sets. This is because initially extracting trials initially trial sets with a one-to-one mapping to trial types.

5.7 Calculating statistics

To compute statistics on the trial data, go to `Trials > Write statistics to spreadsheet`. You can compute multiple statistics, and the time windows to be used for analysis can be specified as explained in 3.1. You can compute the statistics on each of the trials and either save all of these individually or save their average, or you can compute the trial averages and compute statistics on these.

6 Automating processing pipelines

When a processing function is run, it saves the equivalent command as a string to the `history` field of the data. All of these commands together constitute a processing pipeline. To export the processing history to a Matlab script, go to `Tools > Save current processing history as script`. To view the processing history within the command window, run `pupl history`. To run a processing pipeline on the currently active data, go to `Tools > Run processing script`.

6.1 Running pipeline on BIDS sourcedata

To run a processing pipeline on the data within a BIDS `sourcedata/` folder and save it to a folder within `derivatives/`, go to `File > BIDS > Run processing script on sourcedata`.

7 Contributing

PuPL is meant to be extensible. All folders and sub-folders in `add-ons/` are added to the path when `pupl_init` is run, and within each folder in `add-ons/`, PuPL searches for and runs a file called `init.m`, which can be used to add menus to the user interface.

7.1 Adding to the user interface

In the `init.m` file, including the line `global userInterface` will provide access to the PuPL user interface. The functions `appendtodata.m` and `updateactivedata.m` make it convenient to append to or update the `eyeData` variable and both take a single function as their input. For example, if you the following function:

```
function EYE = renameall(EYE, name)

for dataidx = 1:numel(EYE)
    EYE(dataidx).name = name;
end

end
```

it can be added to the user interface in the `init.m` file as follows:

```
function init

global userInterface

uimenu(findobj(userInterface, 'Label', '&File'),...
    'Label', 'Name all ''&foo bar''',...
    'UserData', @()isnonemptyfield(getactive, 'name'),...
    'CallBack', @(h, e)...
        updateactivedata(@()...
            renameall(getactivedata, 'foo bar')))

end
```

The `UserData` attribute of the UI menu is a function that is called each time the user interface updates. If it returns `true`, the menu item will be available. Otherwise it will be inactive.

7.2 creating an event log loader

Code to load raw event data must be a function that takes the full path to the event log as its first argument and returns a struct with a single field, `event`, which is itself a struct array with the following fields:

`type`: the name of the event, a char array
`time`: the time, in seconds, of the event's onset
`rt`: the reaction time in seconds, if applicable (if not, NaN)

7.3 Creating a raw data loader

Code to load raw data must be a function that takes the full path as its first argument and returns a struct with the following fields:

`srate`: the sample rate, in Hz
`urdiam`: a struct with the following fields:
  `left`: left eye pupil diameter, as a double-precision row vector
  `left`: right eye pupil diameter, as a double-precision row vector
`urgaze`: a struct with the following fields:
  `x`: a struct with the following fields:

`left`: left eye gaze x position, as measured from the left side of the screen in millimeters, as a double-precision row vector

`right`: right eye gaze x position, as measured from the left side of the screen in millimeters, as a double-precision row vector

`y`: a struct with the following fields:

`left`: left eye gaze y position, as measured from the top of the screen downward in millimeters, as a double-precision row vector

`right`: right eye gaze y position, as measured from the top of the screen downward in millimeters, as a double-precision row vector

`event` a struct array with the same fields as specified in 7.2

`coords`: a struct with the following fields:

`left` (left eye)

`right` (right eye)

`camera`

Each of the above will be a struct with the fields `x`, `y`, and `z`, specifying coordinates in millimeters using the coordinate system explained in 3.10.1.