eyetools: an R package for simplified analysis of eye data

Tom Beesley and Matthew Ivory

Lancaster University

Author Note

Tom Beesley https://orcid.org/0000-0003-2836-2743

Correspondence concerning this article should be addressed to Tom Beesley, Lancaster University, Department of Psychology, Lancaster University, UK, LA1 4YD, UK, Email: t.beesley@lancaster.ac.uk

Abstract

This tutorial is designed for scientists either experienced in, or interested in, using eye tracking who are wishing to construct reproducible analysis pipelines in R. We begin by framing the necessity for the development of software packages for this purpose before introducing the eyetools R package. This is followed by demonstrations of cleaning and repairing raw data ready for event-related processing. Event-related processing is enabled through functions to detect fixations, saccades, as well as time spent in Areas of Interest. Finally, plotting functions are shown. This is followed by a brief example of how to apply inferential statistics towards the processed data. In offering this walkthough of starting with (relatively) raw eye gaze data before cleaning and processing the data to condense large amounts of data into event-based datasets, we offer eyetools as a toolkit for enabling easier, standardised processing pipelines. eyetools provides a solution to improving reproducible research within eye tracking fields. Finally, we discuss the…

Keywords: eye-tracking; fixations; saccades; areas-of-interest

eyetools: an R package for simplified analysis of eye data

Eye tracking is now an established and widely used technique in the behavioural sciences. Perhaps the scientific discipline with the most invested interest in eye-data is Psychology, where eye-tracking systems are now commonplace in almost all university departments. Beyond academic institutions, eye-tracking continues to be a useful tool in understanding consumer behaviour, user-interface design, and in various forms of entertainment.

By recording the movement of an individual's gaze during research studies, researchers can quantify where and how long individual's look at particular regions of space (usually with a focus on stimuli presented on a 2D screen, but also within 3D space). Eye tracking provides a rich stream of continuous data and therefore can offer powerful insights into real-time cognitive processing. Such data allow researchers to inspect the interplay of cognitive processes such as attention, memory, and decision making, with high temporal precision (Duchowski & Duchowski, 2017).

While there are abundant uses and benefits of collecting eye-movement data in psychology experiments, the continual stream of recording can lead to an overwhelming amount of raw data: modern eye-trackers can record data at 1000 Hz and above, which results in 3.6 million rows of data per hour. The provision of suitable computational software for data reduction and processing is an important part of eye-tracking research. The companies behind eye-tracking devices offer licensed software that will perform many of the necessary steps for eye-data analysis. However, there are several disadvantages to using such proprietary software in a research context. Firstly, the software will typically have an ongoing (annual) license cost for continual use. Secondly, the algorithms driving the operations within such software are not readily available for inspection. Both of these important constraints mean that the use of proprietary analysis software will lead to a failure to meet the basic open-science principle of analysis reproduction, for example as set out by the UK Reproducibility Network: "We expect researchers to… make their research methods, software, outputs and data open, and available at the earliest possible point…The reproducibility of both research methods and research results …is critical to research in certain contexts, particularly in the experimental sciences with a quantitative focus…"

In the current article we introduce a new toolkit for eye-data processing and analysis called "*eyetools*", which takes the form of an R package. R packages (like R itself) are free to use without licence and are therefore available for any user across the world. The package provides a (growing) number of functions that provide an efficient and effective means to conduct basic eye-data analysis. *eyetools* is built with academic researchers in the psychological sciences in mind, though there is no reason why the package would not be effective more generally. The functions within the package reflect steps in a comprehensive analysis workflow, taking the user from initial handling of raw eye data, to summarising data for each period of a procedure, to the visualisation of the data in plots. We hope that the functions are simple enough to mean that the package is easy to use for researchers who are unfamiliar with working with eye data. It should also appeal to researchers accustomed to working with eye data in other environments who wish to transfer to working in R.

One benefit of adopting the R environment for research purposes is that there is an active community of users who are continuously developing and improving functions that help to streamline data processing and analysis. With this, if a certain processing step or analytical method is presently unavailable, one can either contribute these functions oneself, or requests can be made to other members of the community to provide these functions. The continued development of the *eyetools* package means that it does not necessarily represent a static entity, and instead is a constantly growing collection of processes that help to ease the burden of data analysis. As more processes are identified within a research pipeline that represent a "reproducible" and "repeatable" component, these can be standardised and incorporated into packages such as *eyetools*. It is important to stress that *eyetools* is not a completed project, and that contributions and suggestions from the wider community will only help to improve it.

eyetools is, of course, not the only package in R that allows users to work with eye data. A recent assessment of available packages on CRAN identified six other packages that offer relevant functions for the analysis of eye data. **eyeTrackr**, **eyelinker**, and **eyelinkReader**, all offer functionality for data only from experiments that have used ‘EyeLink’ trackers (S-R Research). In contrast, eyetools provides functions that are hardware-agnostic, relying on a format of data that can be achieved from any data source. The **eyeRead** package is designed for the analysis of eye data from reading exercises. The **emov** package offers a limited set of functions and is primarily designed for fixation detection, using the same dispersion algorithm used in eyetools. Finally, **eyetrackingR** is perhaps the most comprehensive alternative package available on CRAN. eyetrackingR offers a large suite of functionality and, like eyetools, can be applied across the entire pipeline. It has functions for cleaning data and various plotting functions, including analysis over time. It does not feature algorithms regarding the detection of events such as saccades or fixations. This limits the ability to conduct more bespoke and dataset specific processing or analysis steps. In comparison, eyetools enables easier data processing up to data analyses, making core tasks in the eye data processing easier and standardised. Table 1 highlights the key differences between the available packages.

Table 1

*A comparison of the functionality available in the available CRAN packages for eye data. * for Tobii data only, ** for text reading experiments only*

Package	Hardware-agnostic	Data Importing	Data processing	Identifies events	Plotting	Inferential Analysis
eyetools	✓	✓*	✓	✓	✓	
eyeTrackr		✓	✓	✓		
eyelinker		✓				
eyelinkReader		✓		✓	✓	
eyeRead	✓		✓	✓**		
emov	✓		✓	✓		
eyetrackingR	✓		✓		✓	✓

In this tutorial we demonstrate the pipeline of analysis functions within *eyetools*. The package has been designed to be simple to use by someone with basic knowledge of data handling and analysis in R. It should appeal to researchers who are working with raw eye data for the first time, as well as those accustomed to working with eye data in other environments who wish to transfer to working in R.

This tutorial is separated into five distinct sections. In the first section, we briefly describe the basic methodology of collecting eye data in general, and in regard to the specific dataset we use to illustrate all the functionality of the *eyetools* package. The second section covers the process for getting data from an eye tracker into an *eyetools*-friendly format. The third section introduces the foundational functions of the *eyetools* package, from repairing and smoothing eye data, to calculating fixations and saccades, and detecting time spent in Areas of Interest (AOIs). The fourth section takes the processed data, and applies basic analysis techniques commonplace in eye data research. In the fifth and final section, we reflect on the benefits of the *eyetools* package, including contributions to open science practices, reproducibility, and providing clarity to eye data analysis.

Installing eyetools

eyetools is available on CRAN and can be installed using: `install.packages("eyetools")` . Instructions for installing development versions can be found at the package repository: https://github.com/tombeesley/eyetools/ . Once installed, the package can be loaded into R: `library(eyetools)`

Preparing data for eyetools

Since there is a wide range of eye tracking hardware available for researchers to use, *eyetools* does not currently offer much in the way of converting raw data from specific hardware. The `hdf5_to_dataframe()` function is designed to work with output from PsychoPy experiments connected to modern Tobii hardware, and will take this format and convert it into the simplified raw data format required for *eyetools*.

The *eyetools* package has been developed primarily with the analysis of psychology experiments in mind. To this end, many of the functions expect a “trial” variable in the data, such that the algorithms will operate over multiple trials and produce output that retains this trial information. Similarly, data in psychology experiments tends to come from multiple participants, and so a participant ID column can be used (though isn't necessary), which allows many functions to be run automatically across multiple participants. It is also necessary to select the relevant "periods" of data within the recording. It is quite typical in psychology experiments for there to be multiple periods within a trial, e.g., fixation; stimulus presentation; response feedback; inter-trial-interval. *eyetools* does not interpret these changes, and so it is necessary to first select the data for the period or periods that are of interest for analysis. Analysis on each separate period would be conducted separately using the functions in *eyetools*.

The starting point for the analysis pipeline is the preparation of the raw eye data, which will consist of recorded samples from the eye-tracker, with each row in the data reflecting a single time-stamped recording. If the eye-tracker is set at 1000Hz, then consecutive recordings will be 1 millisecond of time; at 300Hz, the recordings are 3.33 milliseconds apart. The only requirement for the time column is that the values reflect a consistent and increasing set of values. There is no need to specify the sampling rate, since *eyetools* functions will calculate this automatically.

eyetools expects raw data to have the columns: x, y, time, trial, and participant_ID as shown in Table 2, and missing values in the x and y columns of the raw data should be expressed as "NA".

Table 2

Descriptions of the expected columns in eyetools-formatted data

Column name	Description
x	horizontal spatial coordinate of the estimated eye position
y	vertical spatial coordinate of the estimated eye position
time	timestamp of the recording
trial	an index of the current trial in the data
participant_ID	an index of the current participant in the data (optional)

For many methods of recording, the eye-data will be produced in binocular format. In such cases, *eyetools* has a built in function for combining the data: `combine_eyes()`. This function takes a raw data with coordinates for each eye (i.e., left_x, right_x, left_y, right_y), and converts the data into single x and y coordinates. By deafult, the function does this by taking the average of the coordinates from the two eyes, but it is also possible to select data from the eye with the fewest missing samples. This returns a flattened list of participant data that has x and y variables in place of the left_* and right_* variables.

```r
data_combined <- combine_eyes(HCL,
                              method = "average")
```

Working with eyetools

Counterbalanced designs

Many psychology experiments will position stimuli on the screen in a counterbalanced fashion. For example, in the example data we are using, there are two stimuli, with one of these appearing on the left of the screen and he other on the right. In the design of the experiment, one of these stimuli is a “target” and one is a “distractor”, and the experiment counterbalances whether these are positioned on the left or right across trials. In order to provide a meaningful analysis of the data it is necessary to standardise the data across trials so that the resulting analyses can reflect meaningful eye gaze on relevant stimuli.

eyetools has a built in function which allows us to transform the x (or y) values of the stimuli to take into account a counterbalancing variable: `conditional_transform()`. This function currently allows for a single-dimensional flip across either the horizontal or vertical midline. It can be used on raw data or fixation data; we simply need to append a column to the data to reflect the counterbalancing variable. The result of the function is a set of data in which the x (and/or y) position is consistent across counterbalanced conditions (e.g., in our example, we can transform the data so that the target cue is always on the left). This transformation is especially useful for future visualisations and calculation of time on areas of interest. Note that `conditional_transform()` is another function that does not discriminate between multi-participant and single-participant data and so no participant_ID parameter is required.

In our example data, the stimuli were presented on either the left or the right side of the screen. Here we have merged the eye data with a set of "trial_events" data that describe the events on each trial. We can apply `conditional_transform()` and specify the relevant column (cue_order) that controls the counterbalancing, and the relevant value that signals a switch of position (here "2"). The resulting transformation of the data will mean that the target stimulus is now always positioned on the same side of the screen.

```
data_merged <- merge(data_combined, HCL_behavioural) # merges with the
↪  common variables pNum and trial

data_counterbalanced <-
  conditional_transform(data_merged,
                        flip = "x", #flip across x midline
                        cond_column = "cue_order", # counterbalance column
                        cond_values = "2",# values to flip
                        message = FALSE) # suppress output message
```

Repairing missing data and smoothing data

Despite researcher's best efforts and hopes, participants are likely to blink during data collection, resulting in observations where there are NA values for the x and y coordinates. To mitigate this issue, the `interpolate()` function estimates the gaze path taken, based upon the eye coordinates before and after the missing data. There are two methods for estimating the path, linear interpolation ("approx", the default setting) and cubic spline ("spline"). The default method of linear interpolation replaces missing values with a line of constant slope and evenly spaced coordinates reaching the existing data. The cubic spline method applies piecewise cubic functions to enable a curve to be calculated as opposed to a line between points.

```r
data <- interpolate(data_counterbalanced,
                    method = "approx",
                    participant_ID = "pNum")
```

When using `interpolate()`, a report can be requested so that a researcher can measure how much missing data has been replaced. This parameter changes the output format of the function, and returns a list of both the data and the report. The report can be easily accessed using the following code:

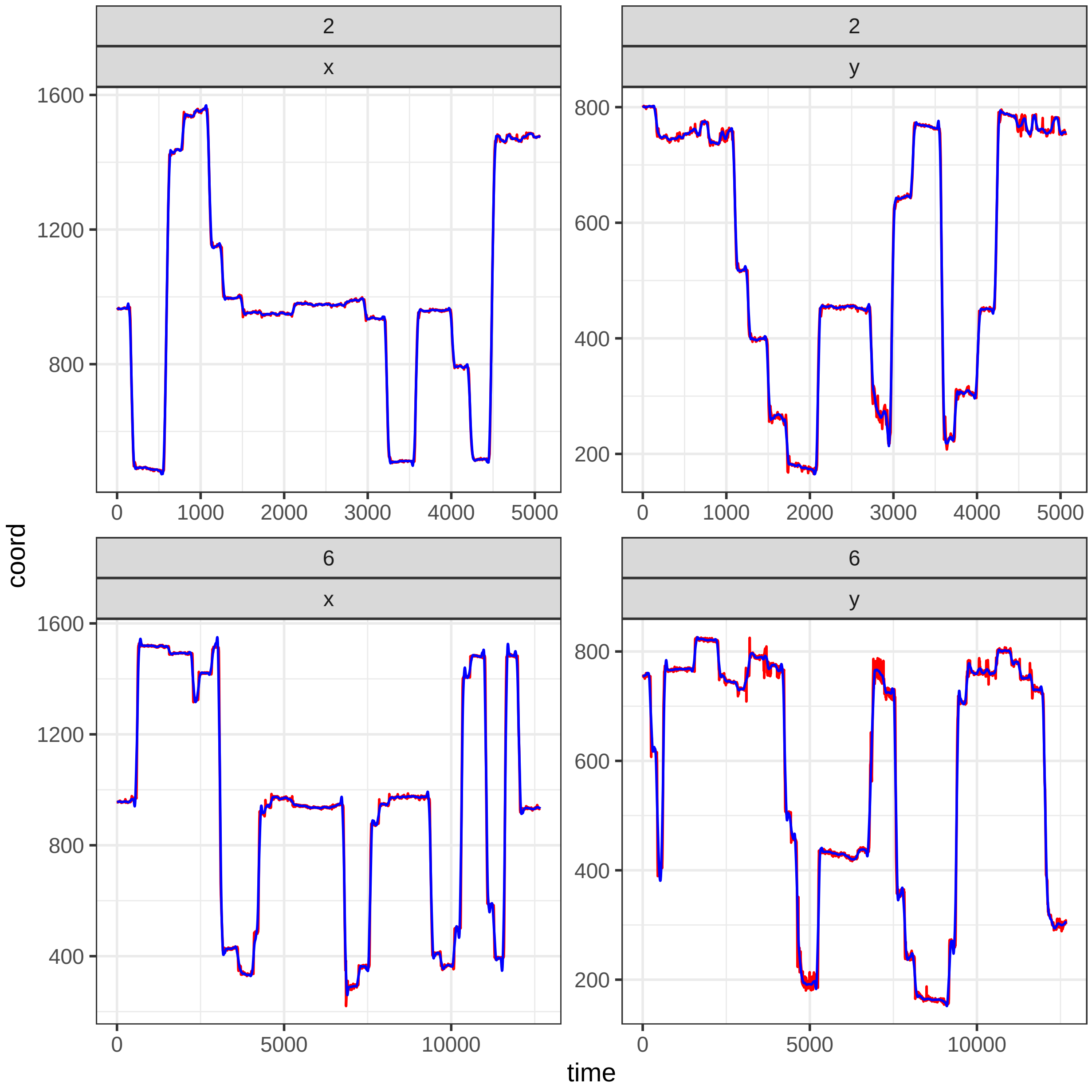
```r
interpolate(data_counterbalanced,
            method = "approx",
            participant_ID = "pNum",
            report = TRUE)[[2]]
```

```
  pNum missing_perc_before missing_perc_after
1  118          0.02314313        0.013885875
2  119          0.01214128        0.005402579
```

As shown, not all missing data has been replaced, since there are certain periods in which the missing data span a period longer than the default setting of the “maxgap” parameter, which is X ms. This default setting is based on a typical duration for a blink [ref].

Once missing data has been fixed, a common step is to smooth the eye data to remove particularly jerky eye movements. The function `smoother()` reduces the noise in the data by applying a moving averaging function. The degree of smoothing can be specified, and a plot can be generated (using data from a randomly selected trial) to observe how well the smoothed data fits the raw data.

```r
data <- smoother(data,
                 span = .02,
                 participant_ID = "pNum",
                 plot = TRUE)
```



Fixations

Once the data has been repaired and smoothed, a core step in eye data analysis is to identify fixations (Salvucci & Goldberg, 2000). Broadly, a fixation is defined as a period in which gaze stops in a specific location for a given amount of time. The period in which the eyes are moving between fixations reflects a "saccade". As such, raw data can be transformed into these meaningful eye data characteristics. These different properties of eye-data have important implications for behavioural research (see X for a review). Beyond their importance for understanding psychological processes, transforming the data into fixations and saccades leads to greater computational efficiency. For example, the built in HCL data in eyetools is 479 kb, which contains 31,041 rows of data (12 trials of data). After processing the data for fixations, the resulting data is 269 rows and can be saved as 3.8 kb (less than 1% the size of the raw data).

In the *eyetools* package, there are two fixation algorithms offered; the first algorithm, `fixation_dispersion()` employs a dispersion-based approach that uses spatial and temporal data to determine fixations. By using a maximum dispersion range, the algorithm looks for sufficient periods of time that the eye gaze remains within this range and once this range is exceed, this is termed as a fixation. The second algorithm, `fixation_VTI()` takes advantage of the idea that data is either a fixation or a saccade and employs a velocity-threshold approach. It identifies data where the eye is moving at a minimum velocity and excludes this, before applying a dispersion check to ensure that the eye does not drift during the fixation period. If the range is broken, a new fixation is determined. Saccades must be of a given length to be removed, otherwise they are considered as micro-saccades [@CITATION_NEEDED_HERE?].

```r
fixations <- fixation_dispersion(data,
                                 min_dur = 150, # Minimum duration (in
                                  ↪  milliseconds) of fixations
                                 disp_tol = 100, # Maximum tolerance (in
                                  ↪  pixels) for the dispersion of values
                                 run_interp = FALSE, # the default is true,
                                  ↪  but we have already run interpolate()
                                 NA_tol = 0.25, # the proportion of NAs
                                  ↪  tolerated within a fixation window
                                 progress = FALSE, # whether to display a
                                  ↪  progress bar or not
                                 participant_ID = "pNum")
```

Additionally, in certain analyses it may be useful to extract the saccades themselves. This can be achieved using the `saccade_VTI()` function.

```
saccades <- saccade_VTI(data,
                         threshold = 150,
                         min_dur = 20,
                         participant_ID = "pNum")
```

Once fixations have been calculated, they can be used in conjunction with Areas of Interest (AOIs) to determine the sequence in which the eye enters and exits these areas, as well as the time spent in these regions. When referring to AOIs, these often refer to the cues presented and the outcome object. In our example, the two cues at the top of the screen are the cues, and the outcome is at the bottom. We can define these areas in a separate dataframe object by giving the centrepoint of the AOI in x, y coordinates along with the width and height (if the AOIs are rectangular) or just the radius (if circular).

```
# set areas of interest
AOI_areas <- data.frame(matrix(nrow = 3, ncol = 4))
colnames(AOI_areas) <- c("x", "y", "width_radius", "height")

AOI_areas[1,] <- c(460, 840, 400, 300) # Left cue
AOI_areas[2,] <- c(1460, 840, 400, 300) # Right cue
AOI_areas[3,] <- c(960, 270, 300, 500) # outcomes
```

In combination with the fixation data, the AOI information can be used to determine the sequence of AOI entries using the `AOI_seq()` function. This fucntion checks whether a fixation is detected within an AOI, and if not, it is dropped from the output, and then provides a list of the sequence of AOI entries, along with start and end timestamps, and the duration.

```
data_AOI_entry <- AOI_seq(fixations, AOIs = AOI_areas,
                          AOI_names = c("predictive", "non-predictive",
                           ↪  "target"),
                          participant_ID = "pNum")
```

Time spent in AOIs can also be calculated from fixations or raw data using the `AOI_time()` function available. This calculates the time spent in each AOI in each trial, based on the data type given, in our case fixation data.

```
data_AOI_time <- AOI_time(fixations, data_type = "fix", AOIs = AOI_areas,
                          AOI_names = c("predictive", "non-predictive",
                           ↪  "target"),
                          participant_ID = "pNum")
```

If choosing to work with the raw data, there is also the option of using `AOI_time_binned()` which allows for the trials to be split into bins of a given length, and the time spent in AOIs calculated as a result.

```
data_AOI_time_binned <- AOI_time_binned(data, AOIs = AOI_areas,
                                        AOI_names = c("predictive",
                                         ↪  "non-predictive", "target"),
                                        bin_length = 100,
                                        max_time = 2000, #in milliseconds
                                        participant_ID = "pNum")
```

Visualisations made easy

When working with eye data, it can be beneficial for the researcher to familiarise themselves with the dataset. Visualising the data through graphics can help to identify meaningful patterns that are obscured when relying on statisitical analyses alone (Kabacoff, 2022). Graphics are also very effective at conveying information in a way that is easily grasped by a diverse audience. eyetools offers a selection of in-built plotting functions that work with data at most stages of processing. These plots are designed to aid in the researcher's processing and data analysis.

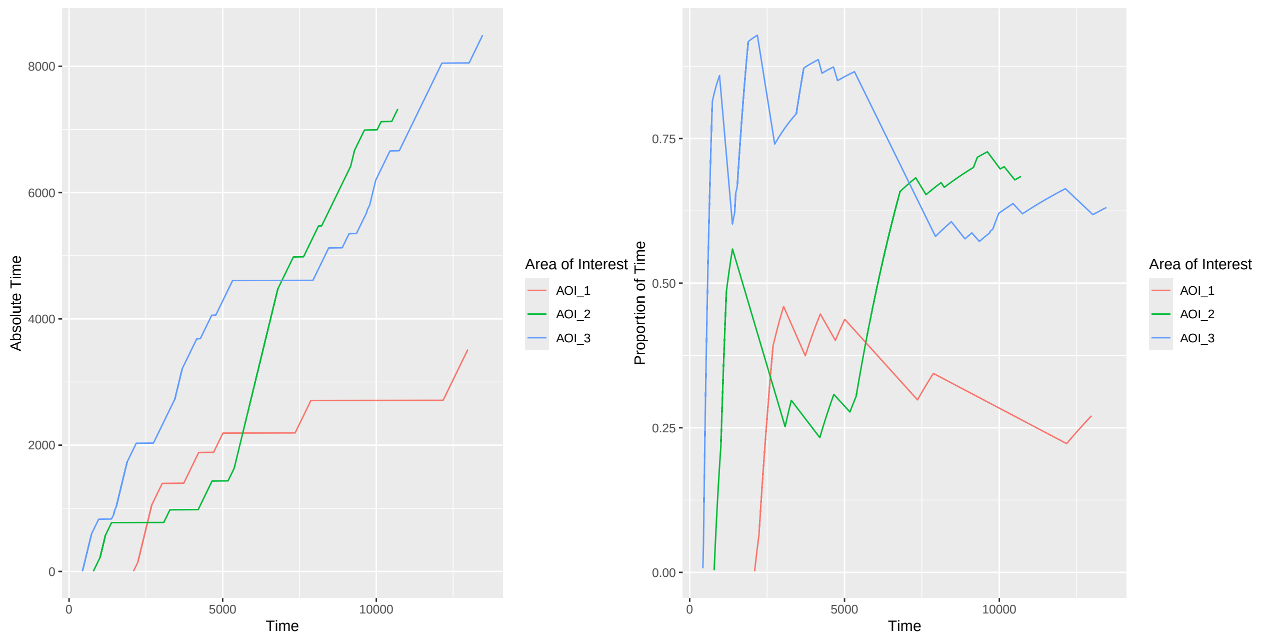
The `plot_AOI_growth()` function offers the representation of how an individual (on a single trial) spends their time looking at the different AOIs. This can be useful to see how AOIs are interacted with over time, and this can be presented as either a cumulative over time, or as a proportion of the time spent in the trial.

```
# plot absolute and then proportional
plot_AOI_growth(data = data,
                AOIs = HCL_AOIs,
                type = "abs",
                trial_number = 1)
plot_AOI_growth(data = data,
                AOIs = HCL_AOIs,
                type = "prop",
                trial_number = 1)
```

Figure 1

Examples of the absolute and proportional time plots from `plot_AOI_growth()`

(a) (b)

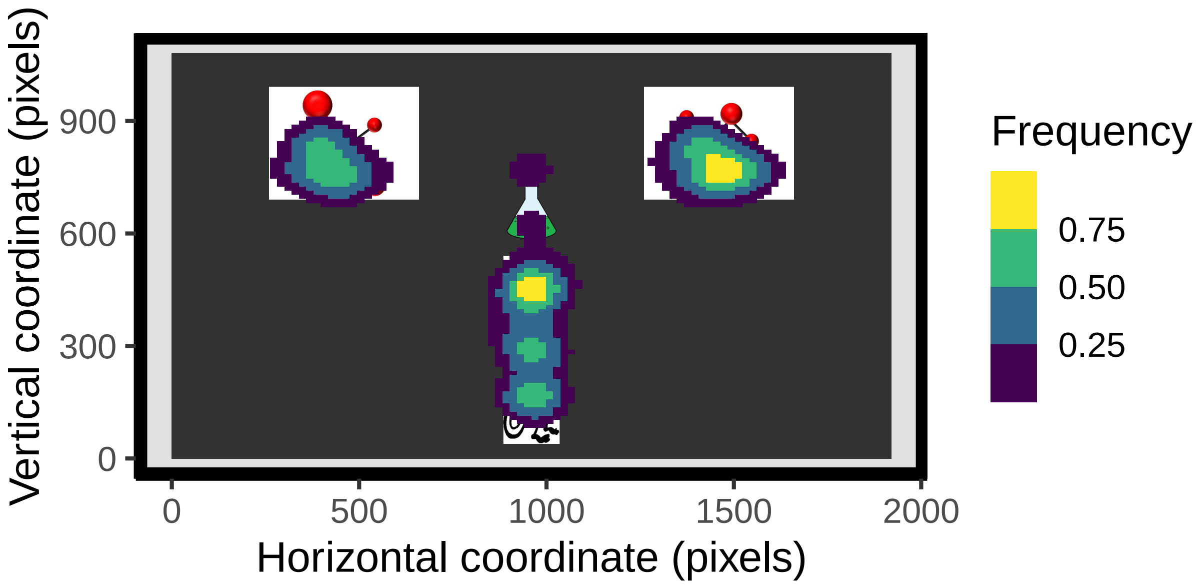


A heatmap of eye gaze positions can be generated using `plot_heatmap()` which takes raw data as an input. As a function, and unlike many of the processing steps, it does not differentiate between trials or participants and plots any coordinate data it is given. This behaviour is allowed as the heatmap offers an excellent and fast "sanity check" that participants were, on the whole, looking at the expected areas of the experiment screen during the trials. As can be seen in Figure Figure 2, we can be reassured that participants do indeed spend most of their time looking at the stimuli on screen rather than in the empty space. `plot_heatmap()` also allows for the modification of the amount of data displayed, using the `alpha_control` parameter. By decreasing `alpha_control` in Figure Figure 3, we gain more visualised information and we can still see that the majority of the data is kept within the stimuli and saccades between these areas.

```
plot_heatmap(data, bg_image = "images/HCL_sample_image.jpg")
```

Figure 2

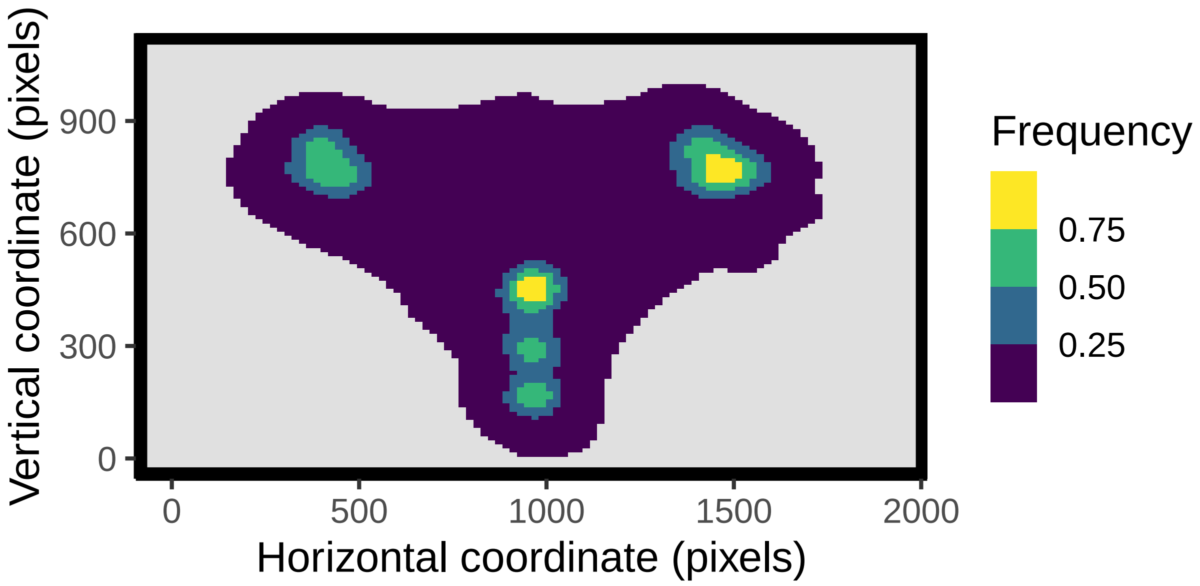
A heatmap overlaid upon a sample stimuli image demonstrating where the participants looked most over all trials



```r
plot_heatmap(data, alpha_control = .001)
```

Figure 3

A heatmap overlaid upon a sample stimuli image demonstrating where the participants looked most over all trials

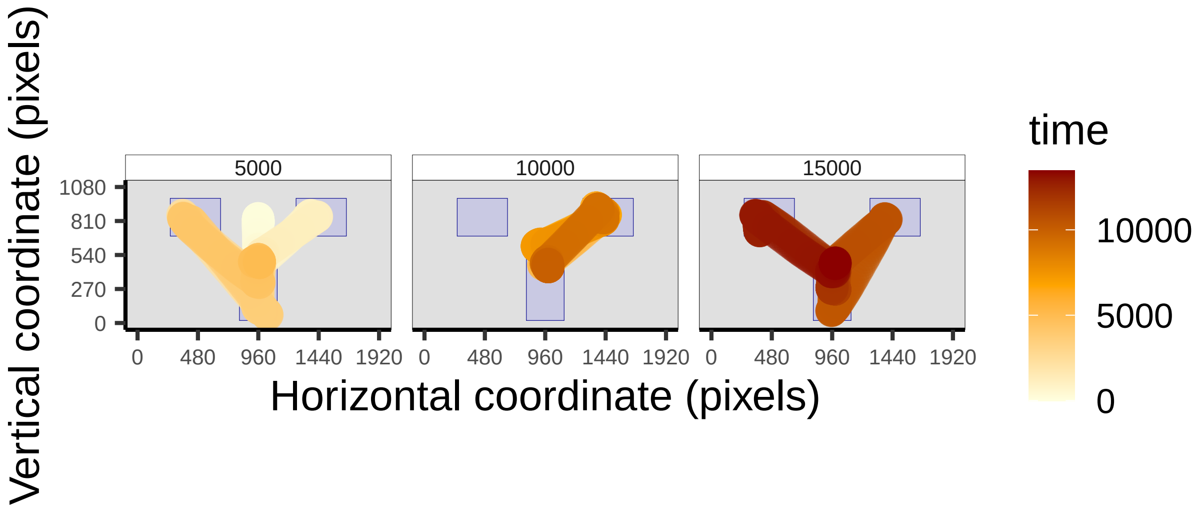


The `plot_seq()` function allows for the plotting of raw data to visualise the gaze pattern from a single trial and where the gaze fell on the screen across the entire trial. Figure 4 offers an example trial split into time bins of 5000ms. This plot shows the time dimension as a change in colour that overlaps older data. This plot serves as a useful check, similar to `plot_heatmap()`, as to where the eyes spent their time, but `plot_seq()` has the benefit of showing the time dimension compared to a simple heatmap.

```r
# plot raw data with bins
plot_seq(data = data[data$pNum == 118,],
         AOIs = HCL_AOIs,
         bin_time = 5000,
         trial_number = 1)
```

Figure 4

The output from plot_seq() with included AOIs and time binned into 5 second sections

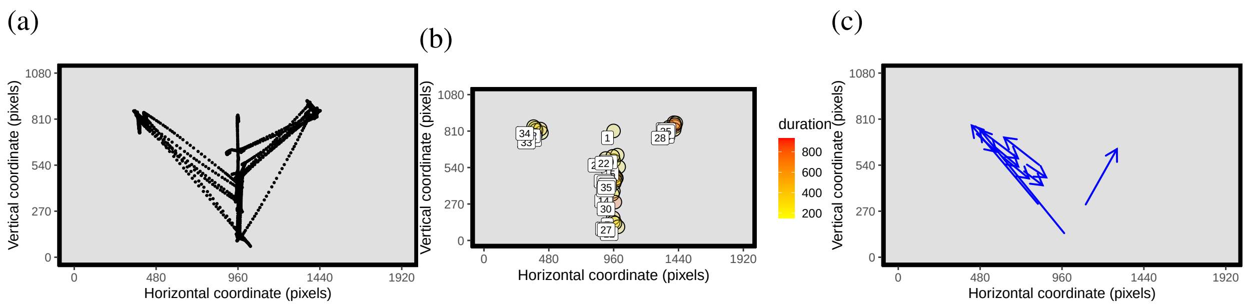


The final plotting function in eyetools is `plot_spatial()`. This can plot raw data, fixations, and saccades, either separately or in combination. `plot_spatial()` plots the location of the eye gaze of a trial, and when given raw data is very similar to the output of `plot_seq()`, when using fixation data, then an additional parameter can be used to label the fixations in their temporal order, enabling a better presentation of how fixations arise. Finally, providing saccade data allows for the length and direction of saccades to be presented.

```
plot_spatial(raw_data = data[data$pNum == 118,],
             trial_number = 1)
# plot just fixation data together
plot_spatial(fix_data = fixations[fixations$pNum == 118,],
             trial_number = 1)
#plot saccades
plot_spatial(sac_data = saccades[saccades$pNum == 118,],
             trial_number = 1)
```

Figure 5

The three types of plot that can be created using `plot_spatial()`



eyetool assumptions

As with any data processing or analysis, there are certain assumptions made when developing the eyetools package. Some of these are built into the package directly, either as errors or warnings, such as the assumption that data is ordered by participant ID (if present) and trial, and some are not built in because they would limit the flexibility of the package functionality. One built-in assumption is the handling of missing data. eyetools expects track loss to be represented as NA within the data, and so any system that provides a different convention for recording track loss needs to be changed prior to using eyetools functions.

During development, eyetools was tested using data collected from a Tobii Pro Spectrum eye tracker recording at 300Hz. Screen resolutions were constant at 1080x1920 pixels, and the timestamps were recorded in milliseconds. Whilst most of the functions were designed to work with any hardware provided the data is formatted to eyetools expectations (with the exception of `hdf5_to_df()` and `hdf5_get_event()` as these convert Tobii data), as well as not relying on specific frequencies or resolutions (either through the function behaviour, or by supplying parameters for specificity), eyetools has not been tested on a diverse set of datasets.

Some default behaviours are in-built, but are easily overrided such as parameters for resolution in the plotting functions. Similarly `saccade_VTI()` and `fixation_VTI()` were tested with 300Hz data. For these functions, as the frequency increases, the relative saccadic velocities will be lower meaning that the thresholds need to be reduced. This is important to note when working with data that is not recorded at 300Hz. To circumvent the potential issue of sample rates being an issue, by default functions that require a sample rate will deduce the frequency from the data.

Analysing eye data

@tom

Discussion

In the present tutorial, we began by identifying the current gap in available tools for working with eye data in open-science pipelines. We then provided an overview of the general data collection process required for eye tracking research, before detailing the conversion of raw eye data into a useable *eyetools* format. We then covered the entire processing pipeline using functions available in the *eyetools* package that included the repairing and normalising the data, and the detection of events such as fixations, saccades, and AOI entries. @SOMETHING_ON_THE_ANALYSIS_GOES_HERE.

From a practical perspective, this tutorial offers a step-by-step walkthrough for handling eye data using R for open-science, reproducible purposes. It provides a pipeline that can be relied upon by novices looking to work with eye data, as well as offering new functions and tools for experienced researchers. By enabling the processing and analysis of data in a single R environment it also helps to speed up data analysis.

Advantages of Open-Source Tools

eyetools offers an open-source toolset that holds no hidden nor proprietary functionality. The major benefits of open-source tools are extensive, but the main ones include the ability to explore and engage with the underlying functions to ensure that

A collaborative community - with open source tools, if an unmet need is identified, then the community can work to provide a solution.

Good Science Practices with eyetools

Creating savepoints (like having processed raw data, and then post-fixation calculation). Reduces the need to completely rework workflows if an issue is detected as savepoints can be used to ensure that computationally-intense or time-heavy processes are conducted as infrequently as possible.

Data Availability

The data required for reproducing this tutorial is available at: @URL. A condensed version of the dataset (starting with the `combine_eyes()` function) is a dataset in the *eyetools* package called HCL.

Code Availability

The code used in this tutorial is available in the reproducible manuscript file available at:(IF STORING IN GITHUB, THEN WE NEED TO CREATE A ZENODO SNAPSHOT FOR A DOI RATHER THAN JUST A GITHUB LINK)

References

Duchowski, A. T., & Duchowski, A. T. (2017). *Eye tracking methodology: Theory and practice*. Springer.

Kabacoff, R. I. (2022). *R in action: Data analysis and graphics with r and tidyverse*. Simon; Schuster.

Salvucci, D. D., & Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, 71–78.