eyetools: an R package for open-source 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

Eye data analysis has become an integral part of data analysis in the psychological sciences. Here we introduce eyetools, an R package that provides an open source means of conducting eye data analysis. eyetools is aimed at researchers who may not have experience programming their own eye data analysis routines. The package provides a number of functions that facilitate key steps in the pipeline of eye data analysis, from basic data processing, extraction of fixations and saccades, to trial level statistics, and data visualisation. In this article we introduce these critical features of the package and provide a step-by-step guide which will enable researchers to find open source solutions to their eye data analysis. We discuss the ways in which we believe eyetools might be expanded in the future and how it may provide a platform for collaborative work on eye data analysis.

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

eyetools: an R package for open-source analysis of eye data

Eye-tracking is now an established and widely used tool that provides powerful measurements of human behaviour. across a number of applications. Its application across academic research is far-reaching, with major importance to the fields of computer science and AI, economics, psychology, and even having influence on art and design. Beyond purely academic application, eye-tracking is widely used in commercial fields where it can provide insights into consumer behaviour, helping to shape product design and marketing.

By recording the movement of an individual's gaze during research studies, users 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 which can offer powerful insights into real-time cognitive processing. Undoubtedly, Psychology is perhaps the scientific discipline which has seen the most substantial adoption of eye-data research, where eye-tracking systems are now commonplace in centres of academic research. Such data allow researchers to inspect the interplay of cognitive processes such as attention, memory, and decision making, with high temporal precision (Beesley et al., 2019).

While there are abundant uses and benefits of collecting eye-movement data, the collection of such data has many implications for the eventual analysis work that is undertaken by the researcher. 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 continual nature of the data also leads to a wealth of choice in the manner in which it is analysed. As such, 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 common 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 license cost for continual use. Secondly, the algorithms driving the operations within such software are not readily available for inspection or adaptation for new purposes. These significant 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 users 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 pipeline, 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. Since the pipeline is contained within the one package, it is not reliant on external software, which enables easy reproducibility of any analyses. Importantly, the functions are simple to use, ensuring that the package will be beneficial for researchers who are unfamiliar with eye data analysis. It should also appeal to researchers accustomed to working with eye data in other environments who wish to transfer to working in R.

eyetools is, of course, not the only package in R that allows users to work with eye data. A recent survey of CRAN (The Comprehensive R Archive Network) 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 source. The ***eyeRead*** package is designed for the specific 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 method employed in *eyetools*. ***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 bespoke analysis steps and it means that analysis needs to be conducted on raw data. This is disadvantageous both in terms of computing time and in the open sharing of data (event data are an order of magnitude smaller in size than raw data). Finally, *saccadr* is a package that can be used to extract saccades from raw data, and has functionality to convert binocular data into monocular data (in a similar manner to that used in *eyetools*).

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

* for Tobii data only, ** for text reading experiments only

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. Our hope is that the package will enable researchers who haven't previously engaged with eye-tracking to do so in an open-source and reproducible manner. We first describe the basic installation process and then the preparation of data into an *eyetools*-friendly format. We then describe the algorithms that can be used to extract the key characteristics of fixations and saccades. We then show the algorithms that can extract critical trial level patterns in behaviour, such as time on areas of interest and sequences of eye movements, as well as means to visualise the data.

Installing *eyetools*

eyetools is available on CRAN and can be installed with the command `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 with the command `library(eyetools)`.

Preparing data for *eyetools*

Since there is a wide range of eye tracking hardware available for researchers to use, *eyetools* currently offers only a limited number of functions for 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. This function takes the default raw data format and converts it into a simplified raw data format suitable for *eyetools*.

The *eyetools* package has been developed primarily with the analysis of experimental psychology data 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 to facilitate analysis, a “pID” column is required (even if data from only a single participant is used). This means that the user can avoid having to generate additional programming steps to analyse and combine the data from multiple participants. 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 automatically, and so it is necessary to first select the data for the period or periods that are of interest for analysis. Analysis on each 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 apart; 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. For the majority of functions that process raw data, *eyetools* expects raw data to have the following columns:

- x = horizontal spatial coordinate of the estimated eye position
- y = vertical spatial coordinate of the estimated eye position
- time = timestamp of the recording
- trial = the index of the current trial in the data
- pID = the unique identifier for the data from each participant

The first four columns should be set as type numeric, while “pID” can be numeric, character, or factor. The order of the columns is not important. Missing values in the x and y columns of the raw data must be expressed as “NA”.

While *eyetools* works on monocular data in the main, many eyetrackers will output binocular data. In such cases, since the primary aim of our analyses is the estimation of the spatial coordinates of gaze, the function `combine_eyes()` should be used to combine the data to form a set of monocular data. This function takes 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 default, the function does this by taking an average of the coordinates from the two eyes of each timestamp, but it is also possible to select data from the eye with the lowest proportion of missing samples. This function returns data that has x and y variables in place of the left_* and right_* variables.

```r
head(HCL,4) # first 4 rows of the built-in data
```

```
# A tibble: 4 x 7
  pID    time left_x left_y right_x right_y trial
  <chr> <dbl>  <dbl>  <dbl>   <dbl>   <dbl> <dbl>
1 118       0   909.   826.   1003.    808.     1
2 118       3   912.   829.   1001.    812.     1
3 118       7   912.   826.   1010.    813.     1
4 118      10   908.   824.   1006.    807.     1
```

```r
data <- combine_eyes(HCL) # create monocular data


head(data, 4) # first 4 rows of the monocular data
```

```
  pID time trial        x        y
1 118    0     1 955.8583 816.5646
2 118    3     1 956.5178 820.6221
3 118    7     1 960.7383 819.7616
4 118   10     1 956.9727 815.3331
```

Repairing missing data and smoothing data

Despite the best efforts of the researcher, there are occasional failures in the accurate recording of the eye position during data collection (e.g., blinks). This results in missing data within the stream of samples, which must be represented in eyetools as NA values for the x and y coordinates. To mitigate the impact of missing data on further analysis, the `interpolate()` function can estimate the missing gaze data, based upon the eye coordinates before and after the missing data, and perform a repair. The default method of linear interpolation ("approx") replaces missing values with a line of constant slope and evenly spaced coordinates that bridge between the existing data (alternatively a cubic “spline” method can be used to apply a curved path between the existing datapoints).

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

When using `interpolate()`, a report can be requested on the proportion of missing data that 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 alone can be accessed in the following way:

```r
interpolate(data,
            method = "approx",
            report = TRUE)[[2]]
```

```
  pID missing_perc_before missing_perc_after
1 118          0.02314313        0.001157156
2 119          0.01214128        0.000000000
```

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 150 ms.

Once interpolation has been performed, a common step is to smooth the eye data to minimise the effect of measurement error on the data. The function `smoother()` reduces the noise in the data by applying a moving average 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,
                 plot = TRUE)
```

Working with *eyetools*

Having explained these rudimentary steps of getting the data ready for the main analysis, we will now describe the core functions available in the latest version of *eyetools*. For illustration, *eyetools* has a built in data set that meets the required format. The data set consists of data from two participants from a few trials of a human causal learning study (Beesley et al., 2015). The nature of this experiment is largely unimportant for the current purposes, but for clarity, the data were collected from the decision period of the procedure, where two rectangular cue stimuli were presented in the top half of the screen, one on the left side and one on the right side. Two smaller response options were presented centrally in the lower half of the screen, one above the other. Participants simply had to look at the cues and choose a response. The raw eye data can be accessed by calling `HCL`, the "behavioural data" (trial events, reaction times, responses, etc) by calling `HCL_behavioural`, and the associated "areas of interest" (described later) can be called with `HCL_AOIs`.

Counterbalanced designs

Many psychology experiments will counterbalance the position of important stimuli on the screen. In the example data, there are two stimuli, with one of these appearing on the left side of the screen and the other on the right. In the design of the experiment, one of these stimuli can be considered a "target" and the other a "distractor", and the experiment counterbalances whether these are positioned in a left/right or a right/left arrangement across trials. In order to provide a meaningful analysis of the eye position over all trials, it is necessary to standardise the data, such that the resulting analyses reflect meaningful eye gaze on each type of stimulus (target or distractor).

eyetools has a built in function, `conditional_transform()`, which allows us to *transform* the x and/or y values of the stimuli so as to take into account a counterbalancing variable. This function currently performs a single-dimensional transformation, across either the horizontal or vertical midline. It can be used on raw data or fixation data; we simply need to add 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.

In the example code, 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 the value “2”). By default the function expects a resultion of 1920x1080, but custom resolutions can be specified. The resulting transformation means that the data is normalised such that the target stimulus is always positioned on the left side of the screen.

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


# perform a transformation of the data across the x coordinate midline
# for all trials with value 2 in the column cue_order
data <- conditional_transform(data,
                              flip = "x",
                              cond_column = "cue_order",
                              cond_values = "2")
```

Fixations

Once the data has been repaired and smoothed, a core step in eye data analysis is to identify fixations. Broadly, a fixation is defined as a period in which the eye stops moving and is held in a specific location for a significant period of time [typically longer than 100 ms; Salvucci and Goldberg (2000)]. The period in which the eyes are moving between fixations reflects a “saccade”. While the eyes move during these brief (typically less than 50 ms) periods of movement, significant perceptual suppression occurs and there is minimal information processing (Duren & Sanders, 1995; Irwin et al., 1995; Sanders & Houtmans, 1985). Therefore for many cognitive psychologists, the periods of fixation are particularly important and reflect the most relevant periods of information processing in a task.

The raw data can be transformed into these meaningful eye data characteristics. 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 (from just 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. Not only is this more computationally efficient, but it also means the data are now in a far more practical format for storage in online data repositories.

There are two fixation algorithms offered in the *eyetools* package, both based on methods presented by Salvucci and Goldberg (2000). The first, `fixation_dispersion()` seeks periods of low variability in the spatial component of the data; the algorithm looks for sufficient periods of time in which the gaze position remains within a tolerated maximum range of dispersion. Once this range is exceeded, this is deemed to be the end of a possible period of fixation. If the total time of this fixation period is longer than the minimum required (set by the `min_dur` parameter), then this fixation is stored as an entry in the returned object.

The second algorithm, `fixation_VTI()` , employs a velocity-threshold approach to identifying fixations, based on the algorithm described in Salvucci and Goldberg (2000). Since points of fixation occur when the eye is not in consistent motion, the algorithm computes the Euclidean distance between points and then determines the velocity of the eye. Periods in which this velocity is consistently below the velocity threshold (for which the default is 100 degrees of visual angle per second) are identified as a potential period of fixation. The algorithm then applies a dispersion check to ensure that the eye maintains a relatively stable position across this period. Fixations must be of a minimum length for classification (by default 150 ms).

Here we can see the example data passed to the `fixation_dispersion()` algorithm and the resulting fixations that are returned.

```
fixations <-
  fixation_dispersion(data,
                      min_dur = 150, # Min duration in ms
                      disp_tol = 100, # Max dispersion tolerance in pixels
                      NA_tol = 0.25, # proportion of NAs tolerated
                      progress = FALSE) # toggle progress bar



head(fixations, 4)
```

```
  pID trial fix_n start  end duration    x   y prop_NA min_dur disp_tol
1 118     1     1     0  173      173  959 811       0     150      100
2 118     1     2   197  397      200  961 590       0     150      100
3 118     1     3   400  653      253  958 490       0     150      100
4 118     1     4   803 1083      280 1372 839       0     150      100
```

Saccades

Between periods of fixation, the velocity of the eye increases rapidly as it makes a saccade towards the next point of fixation. The `saccade_VTI()` function will extract saccades using the velocity threshold algorithm described above. The resulting output provides details of each saccade, such as the timing of the saccade onset, duration, and the origin and terminus coordinates. As with the fixation algorithms, default parameters have been chosen, but they can be adapted to fit the requirements of the researcher.

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

```
head(saccades, 4)
```

```
  pID trial sac_n start  end duration origin_x origin_y terminal_x terminal_y
1 118     1     1  2180 2240       60 833.2688 296.7871   487.3967   705.9158
2 118     1     2  2710 2750       40 614.5028 605.7001   862.3837   408.3421
3 118     1     3  3673 3726       53 885.6256 253.4150   558.1883   655.7776
4 118     1     4  4213 4233       20 460.3286 722.8386   577.2034   617.8567
  mean_velocity peak_velocity
1      225.0736      331.8455
2      200.3353      263.8863
3      243.7927      340.3059
4      195.6512      251.7763
```

Area of interest (AOI) analysis

A critical component in many analyses of eye gaze is the assessment of time spent in regions of space. *eyetools* has a number of functions for assessing the time spent in Areas of Interest (AOIs), as well as the sequence in which the eye enters and exits these areas. AOIs will typically reflect regions of space in which critical stimuli appear. AOIs are defined in *eyetools* using a dataframe object, where each row reflects a unique AOI, where values code for 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). This object can be created using the function `create_AOI_df()`:

```
# set areas of interest
AOI_areas <- create_AOI_df(3)


# populate this dataframe with AOI dimensions
# (x, y, width/radius, height)
```

```
AOI_areas[1,] <- c(460, 840, 400, 300) # Left rectangualar AOI
AOI_areas[2,] <- c(1460, 840, 200, NA) # Right circular AOI
AOI_areas[3,] <- c(960, 840, 200, 400) # Centre rectangualar AOI
```

Time spent in AOIs can also be calculated from either fixations or raw data using the `AOI_time()` function. This calculates the time spent in each AOI per trial. The resulting output can be expressed in the form of absolute time, or, by passing a vector of times to the "trial_time" parameter, can be expressed as proportional time.

```
data_AOI_time <-
  AOI_time(data = fixations,
           data_type = "fix",
           AOIs = HCL_AOIs,
           AOI_names = c("target", "distractor", "outcomes"),
           as_prop = TRUE,
           trial_time = HCL_behavioural$RT)

head(data_AOI_time, 9)
```

```
  pID trial        AOI      time
1 118     1     target 0.1043446
2 118     1 distractor 0.2488674
3 118     1   outcomes 0.4041589
4 118     2     target 0.1380248
5 118     2 distractor 0.1702220
6 118     2   outcomes 0.4816758
7 118     3     target 0.1391737
8 118     3 distractor 0.1080352
9 118     3   outcomes 0.5397965
```

We can see from the resulting data that the function provides time on each AOI for each trial. Used in combination with the `conditional_transform()` function, `AOI_time()` provides a very efficient way to assess time on critical regions of space. Since the data is in long format, it can be easily processed further with common techniques in R:

```
library(dplyr)


data_AOI_time %>%
  group_by(AOI) %>%
  summarise(mean_time = mean(time))
```

```
# A tibble: 3 x 2
  AOI        mean_time
  <fct>          <dbl>
1 target         0.201
2 distractor     0.237
3 outcomes       0.365
```

The `AOI_time_binned()` function can assess the duration of time spent in AOIs, divided into sequential time bins. Since fixations will naturally overlap these segments in many circumstances, this function operates only on raw data. Here we are assessing time in the three AOIs for periods of 1000 ms in length, and limiting this analysis to the first 8000 ms.

```
data_AOI_time_binned <-
  AOI_time_binned(data,
                  AOIs = HCL_AOIs,
                  AOI_names = c("target", "distractor", "outcomes"),
                  bin_length = 1000, # in milliseconds
                  max_time = 8000) # in milliseconds
```

```r
head(data_AOI_time_binned, 10)
```

```
   pID trial bin_n target distractor outcomes
1  118     1     1      0        217      337
2  118     1     2      0        187      757
3  118     1     3    460          0      430
4  118     1     4    270          0      680
5  118     1     5    220          0      593
6  118     1     6      0        630      337
7  118     1     7      0        797        0
8  118     1     8      0        370        0
9  118     2     1    617          0        0
10 118     2     2    167          0      800
```

It is also possible to determine the sequence of entries into AOIs using the `AOI_seq()` function. This function currently works only with fixation data. For a given trial, the sequence of fixations is assessed against the AOIs provided, where consecutive fixations within the same AOI are combined into one "entry period". The result of this function is a sequence of AOI entries per trial for each participant, providing data on the sampling order of AOIs. The resulting output provides start and end times and duration of each entry.

```r
data_AOI_entry <-
  AOI_seq(fixations,
          AOIs = HCL_AOIs,
          AOI_names = c("target", "distractor", "outcomes"))

head(data_AOI_entry, 9)
```

```
  pID trial        AOI start  end duration entry_n
1 118     1   outcomes   400  653      253       1
2 118     1 distractor   803 1083      280       2
3 118     1   outcomes  1233 2120      887       3
4 118     1     target  2260 2666      406       4
5 118     1   outcomes  2760 3646      886       5
6 118     1     target  3753 4116      363       6
7 118     1   outcomes  4286 5323     1037       7
8 118     1 distractor  5403 6772     1369       8
9 118     1 distractor  7652 9272     1620       9
```

Knowing the order in which the eyes visit particular regions of space is essential for many steps in eye data analysis. For example, each trial might start with a fixation point in the centre of the screen. Below we show how the `AOI_seq` function can be used in combination with other basic R commands to efficiently detect the first AOI entry on each trial. We can see that in all but one of the 12 example trials, participants process the central fixation point first.

```r
library(dplyr)


# add a central fixation AOI region
HCL_AOIs[4,] <- c(960, 810, 200, 200)


data_AOI_entry <-
  AOI_seq(fixations,
          AOIs = HCL_AOIs,
          AOI_names = c("target", "distractor", "outcomes", "fixation"))


data_AOI_entry %>%
  group_by(pID, trial) %>%
  slice(1)
```

```
# A tibble: 12 x 7
# Groups:   pID, trial [12]
   pID   trial AOI      start   end duration entry_n
   <chr> <dbl> <chr>    <dbl> <dbl>    <dbl>   <dbl>
 1 118       1 fixation     0   173      173       1
 2 118       2 fixation     0   186      186       1
 3 118       3 outcomes   393   906      513       1
 4 118       4 fixation     0   153      153       1
 5 118       5 fixation    10   163      153       1
 6 118       6 fixation     0   177      177       1
 7 119       1 fixation     0   246      246       1
 8 119       2 fixation     0   160      160       1
 9 119       3 fixation     0   180      180       1
10 119       4 fixation     0   227      227       1
11 119       5 fixation     0   197      197       1
12 119       6 fixation     0   260      260       1
```

Visualisations

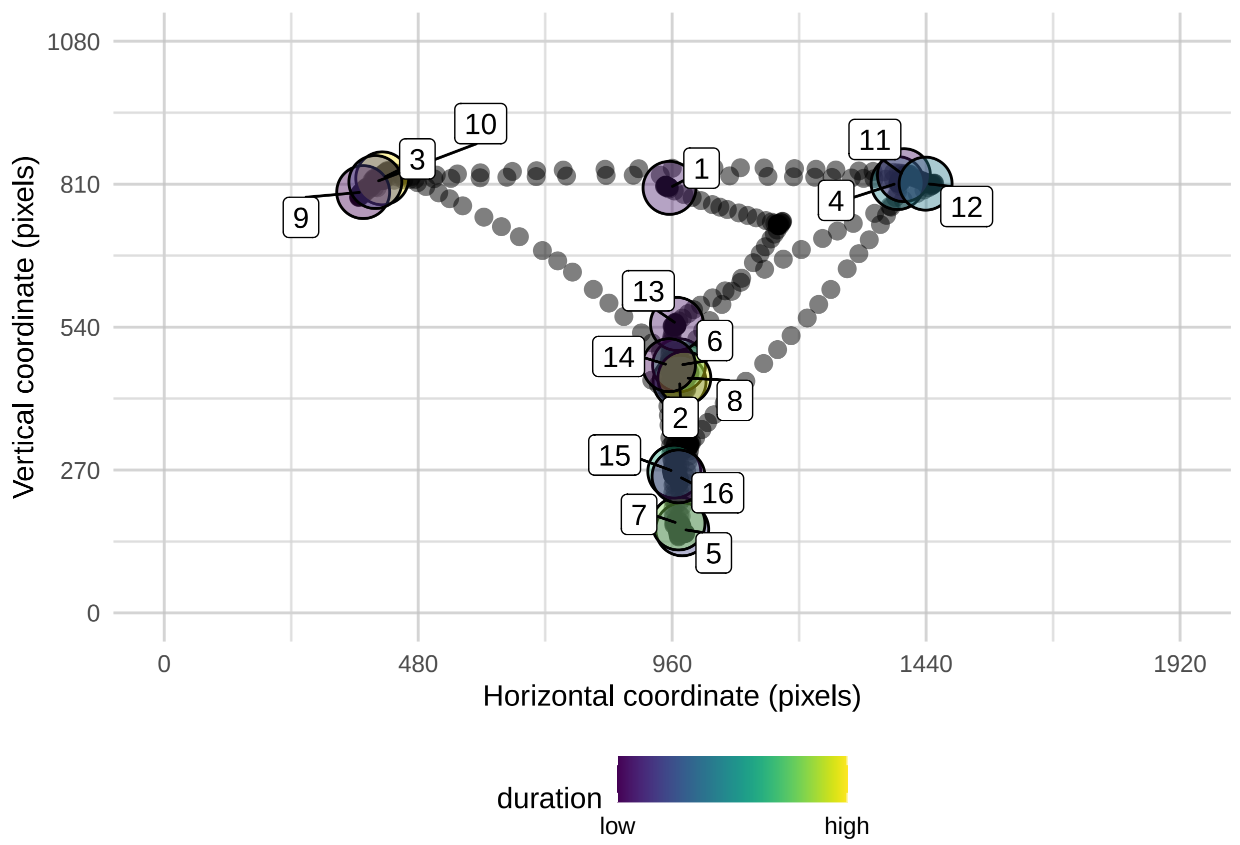
The *eyetools* package has a number of built in visualisations that allow for functional plots of the data, with minimal effort. All plots use the dominant graphical R package `ggplot`, which means that the resulting plots from these functions are ggplot objects and can therefore be customised using the full suite of options for ggplot and its extensions.

`plot_spatial()` offers a simple means to view the data produced by *eyetools*. By default this will plot all of the data that is passed to the function, but participant IDs and trial values can be specified in order to plot specific data. Here we plot the raw data from a single trial for one participant, with the detected fixations overlaid. When using fixation data, the fixations are labelled in their temporal order (by default), enabling a clear presentation of how the fixations arose.

```
plot_spatial(raw_data = data,
             fix_data = fixations,
             pID_values = 118,
             trial_values = 6)
```

Figure 1

Raw data (grey points) and fixations (coloured circles) for a single trial

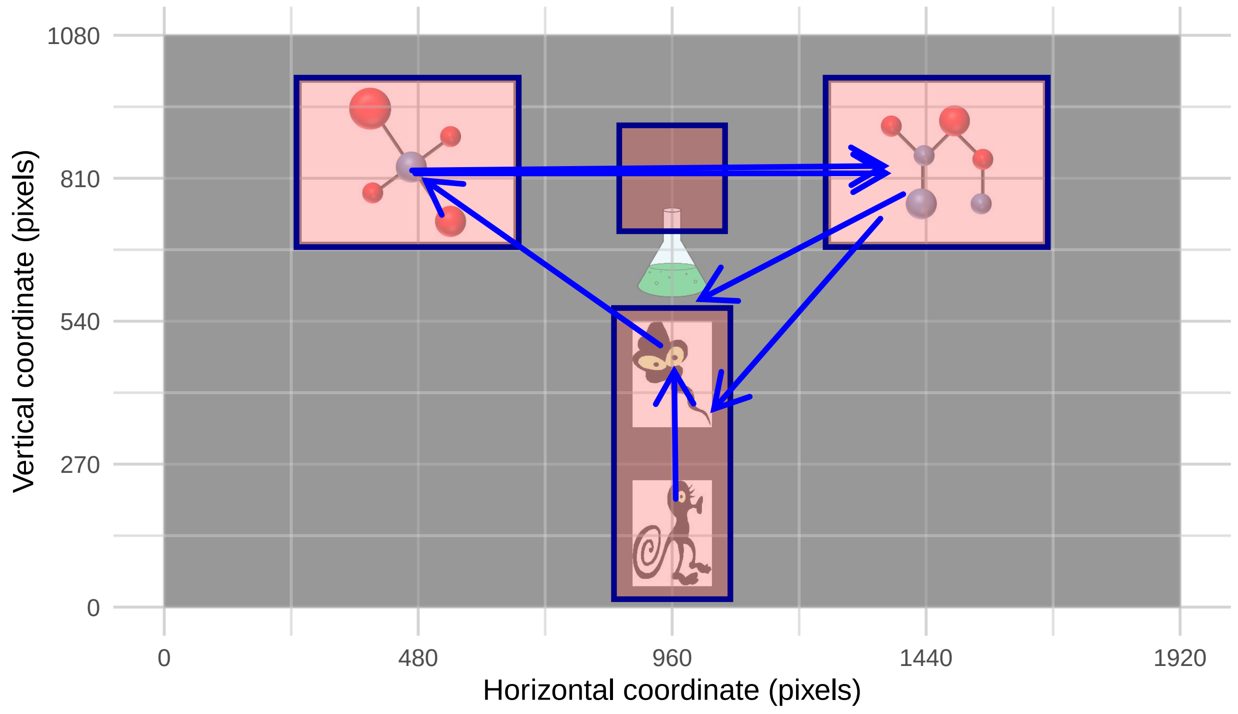


In addition to eye data, a background image can be added to the plot, which is useful for inspecting data over a representation of the experimental task. If AOIs have been defined, these can be plotted as well. Here we demonstrate the plotting of the saccades, AOIs, and a background image:

```
plot_spatial(sac_data = saccades,

             AOIs = HCL_AOIs,

             pID_values = 118,

             trial_values = 6,

             bg_image = "images/HCL_sample_image.png")
```

Figure 2

Saccades (blue arrows) and Area-Of-Interest regions (pink shapes) for a single trial, against a background image.

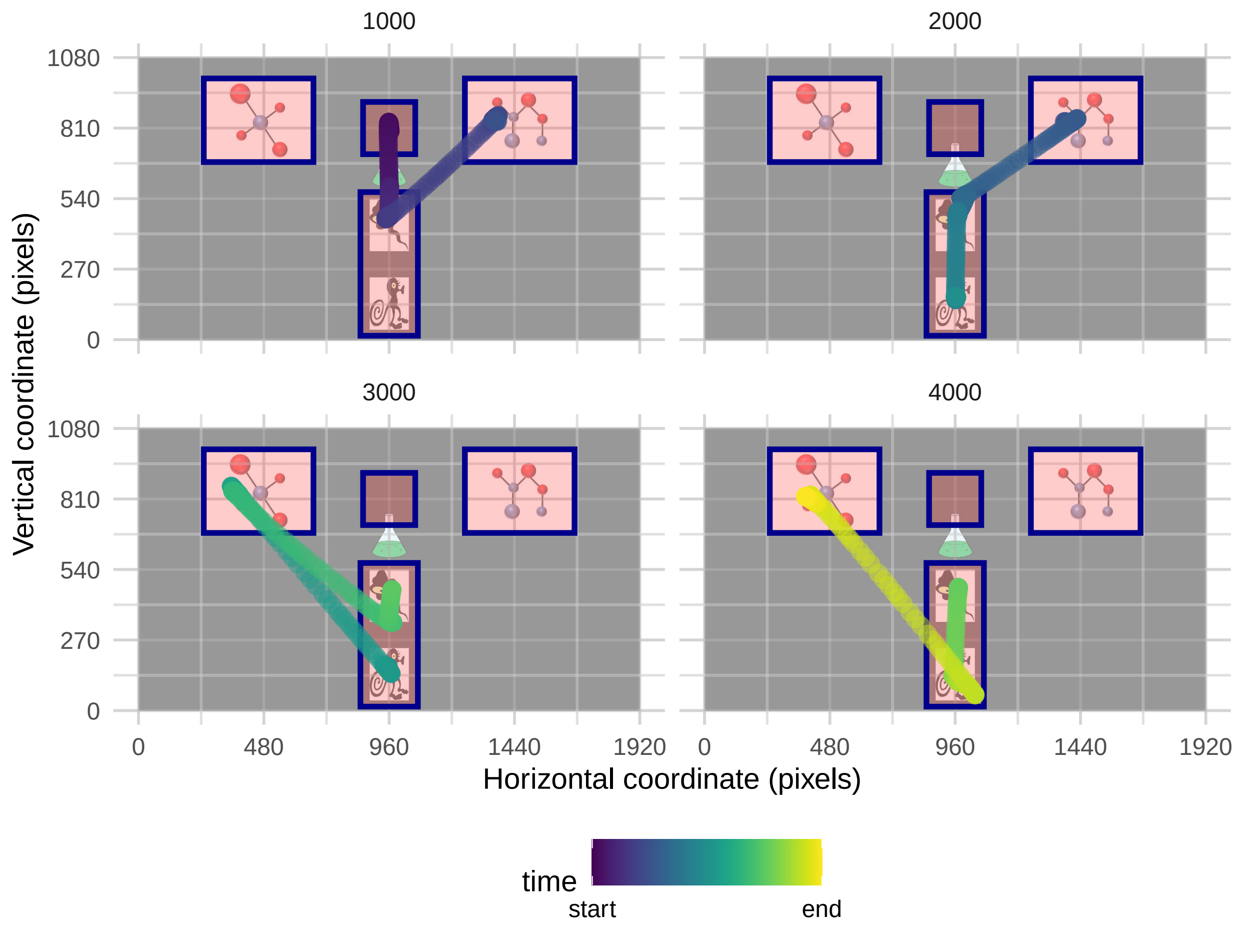


The function `plot_seq()` is useful for visualising data as a series of plots, mapping out eye movements over the course of a single trial. By default this function will plot a randomly selected trial from the raw data that is passed to the function. Otherwise, specific trials and participant values can be specified. The function requires a “bin_time” parameter, that specifies the length of each time-period within the trial. An optional parameter of “bin_range” can be specified to restrict the range of these periods that are presented. For example here we plot data in periods of 1000 ms across the first four of these periods.

```
plot_seq(data = data,
         bin_time = 1000,
         bin_range = c(1,4),
         trial_values = 1,
         pID_values = 118,
         AOIs = HCL_AOIs,
         bg_image = "images/HCL_sample_image.png")
```

Figure 3

The data from the same example trial, plotted across 4 consecutive bins of 1000 milliseconds



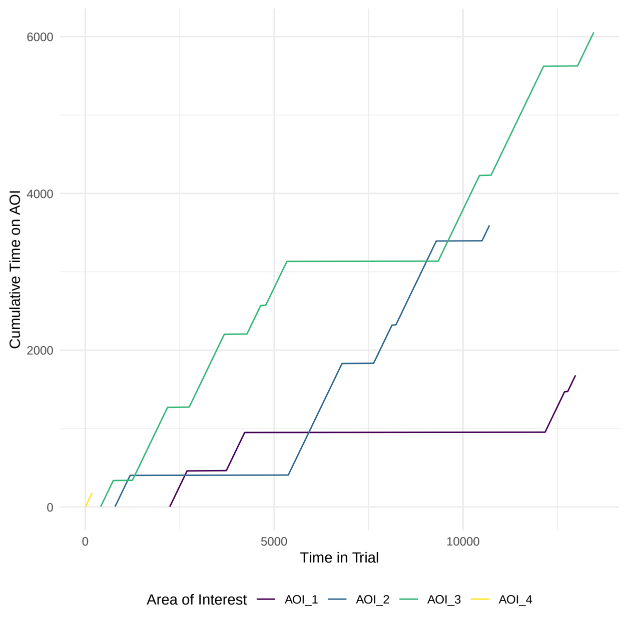
The `plot_AOI_growth()` function offers a visualisation of the progression of time spent on AOIs across a single trial. This can be useful to see how participants interact with AOIs over time, and this can be presented as either a plot of the cumulative time, or as a proportion of the time spent in the trial.

```r
# plot absolute and then proportional
plot_AOI_growth(data = data,
                AOIs = HCL_AOIs,
                type = "abs",
                pID_values = 118,
                trial_values = 1)
plot_AOI_growth(data = data,
                AOIs = HCL_AOIs,
                type = "prop",
                pID_values = 118,
                trial_values = 1)
```

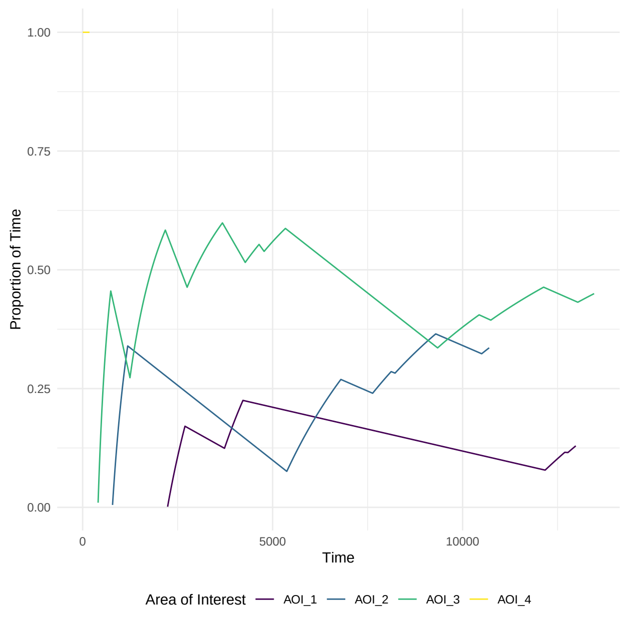
Figure 4

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

(a)



(b)



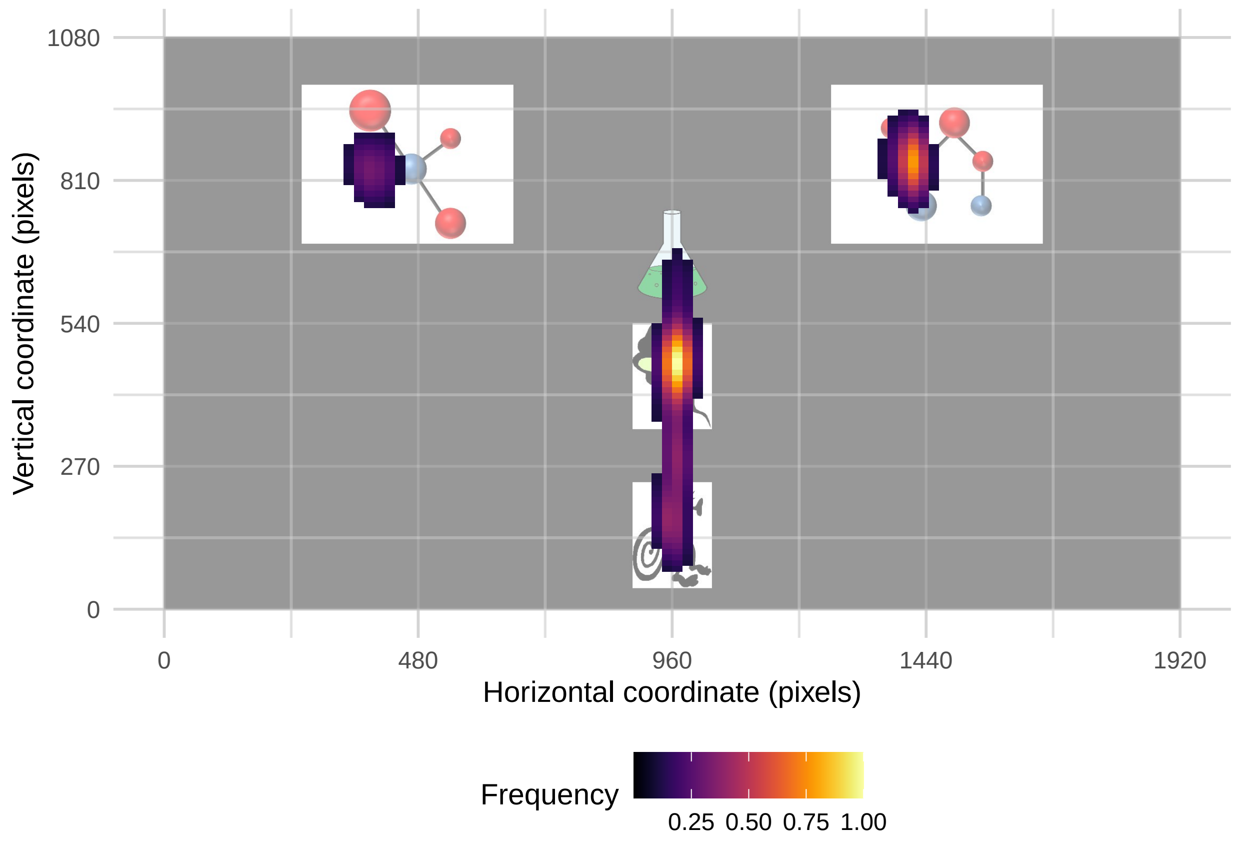
368 A heatmap of eye gaze positions can be generated using `plot_heatmap()` which takes
369 raw data as input. Like `plot_spatial()`, it is possible to select certain pID and trial_values,

therefore offering a complementary visualisation of raw data. As can be seen in Figure Figure 5, 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_range` parameter, which requires a pair of values to specify a range between 0 and 1. By restricting the range of values displayed

```r
plot_heatmap(data,
             pID_values = 118,
             trial_values = c(1,3),
             alpha_range = c(0.1,1),
             bg_image = "images/HCL_sample_image.png")
```

Figure 5

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

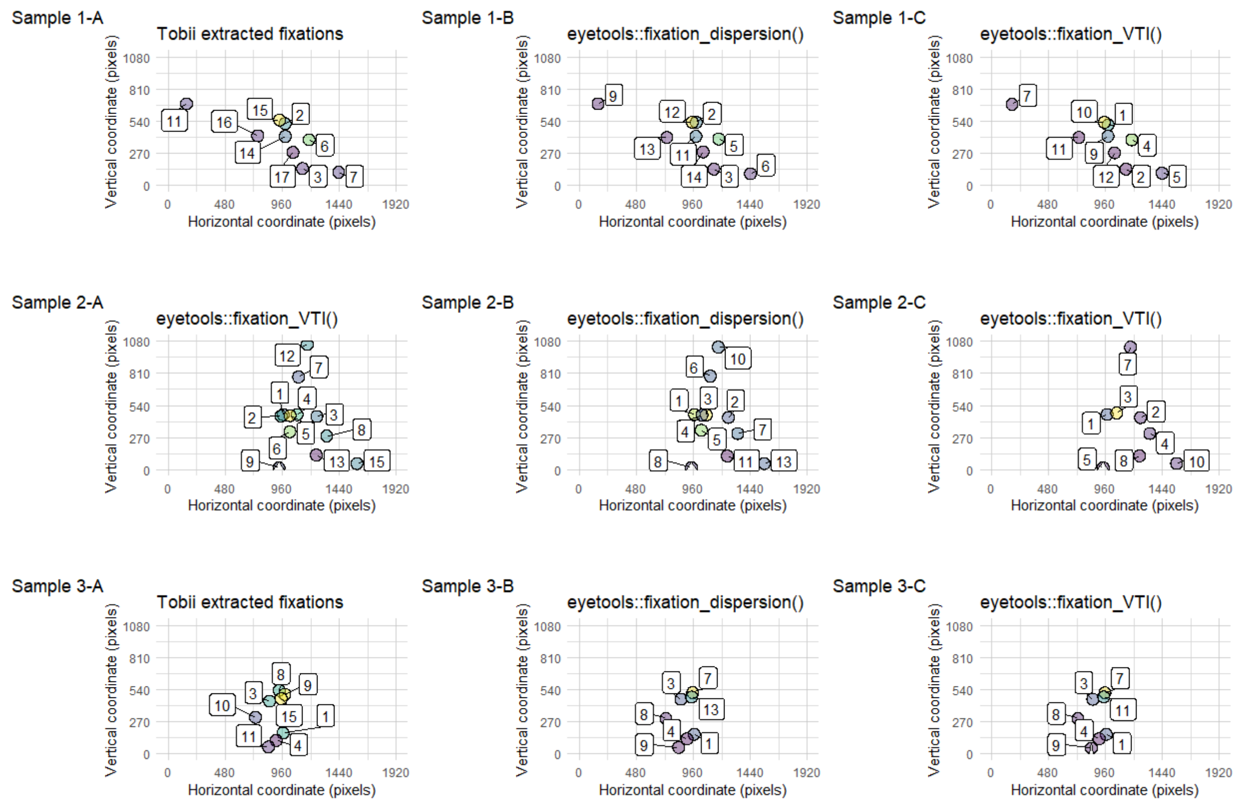


Validation of fixation metrics

eyetools uses implementations of common methods for extracting fixations from raw data (Salvucci & Goldberg, 2000). To provide a simple validation of our primary fixation algorithm, we took raw data collected independently from a researcher outside of our lab on an unknown task. The researcher provided the raw data and extracted fixations for a 6 minute period of data collection, from Tobii Pro Lab software. From the raw data we used the *eyetools* `fixation_dispersion()` and `fixation_VTI()` algorithms to compute fixations from randomly drawn periods of 10 seconds. Figure Figure 6 shows side-by-side comparisons of Tobii and *eyetools* extracted fixations from 3 such periods (the raw data and analysis script for this comparison is available in the manuscript repository for full exploration of other periods). Somewhat unsurprisingly, the algorithms show a very similar spread of fixations for these periods. Notably the number of overall fixations differs across the samples, which is a consequence of the particular parameters used to define fixations, such as the dispersion tolerance and the minimum duration.

Figure 6

A comparison of extracted fixations from Tobii Pro Lab (A:left), eyetools::fixation_dispersion() (B:centre) and eyetools::fixation_VTI() (C:right), across 3 samples.



Summary and future directions

This paper has given an introduction and basic tutorial on working with an open source R package for an analysis pipeline for eye data. We began by identifying the current gaps in available tools for working with eye data in reproducible pipelines. We then provided an overview of the initial steps in working with raw eye data and the conversion of raw eye data into a usable format for working with the functions in *eyetools*. We then covered many useful steps in the analysis pipeline using functions available in the *eyetools* package that included the repairing and normalising of the data, the detection of events such as fixations and saccades, and the trial level analysis of data patterns, such as time on areas of interest, and the sequencing of entries to areas of interest.

This tutorial offers a step-by-step walk-through for handling eye data using R, demonstrating that the *eyetools* package provides a set of tools that will lead to reproducible analysis steps for many experimental psychologists. It is hopefully clear that the functions in *eyetools* are flexible and powerful, yet ultimately simple to implement. While these functions represent some initial steps in eye data analysis, since the objects that result from these functions are standard formats in R (i.e., dataframes and plots), they will provide the user a means to enable more complex or nuanced analyses.

eyetools offers an open-source toolset that holds no hidden nor proprietary functionality. The major benefits of open-source tools are extensive: not only do they allow for full inspection and reproducibility of analyses, but they also support and enable the development and sharing of new analysis functions. There are a number of obvious features that would be hugely beneficial in future versions of *eyetools*. For example, while the package provides means to determine the order of eye-movements (`AOI_seq()`) there is no means to compare these patterns across different trials or periods. Such "scan-path analyses" have been used effectively in cognitive studies and so algorithms to conduct these analyses would be an obvious next step. There are also no functions within the package to handle pupilometry data, despite the obvious benefits of analysing these data. From a development perspective, while these features would be hugely beneficial to the package, they will only be implemented as and when there is a need in our research. Thus our hope for *eyetools* is that future versions will benefit from user engagement and an expansion of the toolset to enable an ever more powerful set of features. We believe eyetools provides a solid foundation for this collaborative venture.

Data and code availability

This manuscript was written in Quarto and can be reproduced from the manuscript source files which are available at https://github.com/tombeesley/BRM_eyetools . The manuscript details functions from the latest CRAN version of *eyetools* which is 0.9.2. We welcome contributions to the development of *eyetools* by posting bug reports and suggested improvements at https://github.com/tombeesley/eyetools/issues .

References

Beesley, T., Nguyen, K. P., Pearson, D., & Le Pelley, M. E. (2015). Uncertainty and predictiveness determine attention to cues during human associative learning. *Quarterly Journal of Experimental Psychology*, *68*(11), 2175–2199. https://doi.org/10.1080/17470218.2015.1009919

Beesley, T., Pearson, D., & Le Pelley, M. (2019). *Chapter 1 - eye tracking as a tool for examining cognitive processes* (G. Foster, Ed.; pp. 1–30). Academic Press. https://doi.org/10.1016/B978-0-12-813092-6.00002-2

Duren, L. L. van, & Sanders, A. F. (1995). Signal processing during and across saccades. *Acta Psychologica*, *89*(2), 121–147. https://doi.org/10.1016/0001-6918(94)00029-G

Irwin, D. E., Carlson-Radvansky, L. A., & Andrews, R. V. (1995). Information processing during saccadic eye movements. *Acta Psychologica*, *90*(1), 261–273. https://doi.org/10.1016/0001-6918(95)00024-O

Salvucci, D. D., & Goldberg, J. H. (2000). *the symposium*. 71–78. https://doi.org/10.1145/355017.355028

Sanders, A. F., & Houtmans, M. J. M. (1985). There is no central stimulus encoding during saccadic eye shifts: A case against general parallel processing notions. *Acta Psychologica*, *60*(2), 323–338. https://doi.org/10.1016/0001-6918(85)90060-5