eyetools: an R package for simplified analysis of eye data

Tom Beesley and Matthew Ivory

Lancaster University

Author Note

Tom Beesley ⓘD 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

Abstract goes here

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

eyetools: an R package for simplified analysis of eye data

Introduction

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 Reprodicibility Network: "We expect researchers to… make their research methods, software, outputs and data open, and available at the earliest possible point…The reproducibility

27 of both research methods and research results …is critical to research in certain contexts,

28 particularly in the experimental sciences with a quantitative focus…"

29   In the current article we introduce a new toolkit for eye-data processing and analysis called

30 "*eyetools*", which takes the form of an R package. R packages (like R itself) are free to use

31 without licence and are therefore available for any user across the world. The package provides a

32 (growing) number of functions that provide an efficient and effective means to conduct basic

33 eye-data analysis. *eyetools* is built with academic researchers in the psychological sciences in

34 mind, though there is no reason why the package would not be effective more generally. The

35 functions within the package reflect steps in a comprehensive analysis workflow, taking the user

36 from initial handling of raw eye data, to summarising data for each period of a procedure, to the

37 visualisation of the data in plots. We hope that the functions are simple enough to mean that the

38 package is easy to use for researchers who are unfamiliar with working with eye data. It should

39 also appeal to researchers accustomed to working with eye data in other environments who wish

40 to transfer to working in R.

41   *eyetools* is, of course, not the only package in R that allows users to work with eye data. A

42 recent assessment of available packages on CRAN identified six other packages that offer relevant

43 functions for the analysis of eye data. **eyeTrackr**, **eyelinker**, and **eyelinkReader**, all offer

44 functionality for data only from experiments that have used 'EyeLink' trackers (S-R Research). In

45 contrast, eyetools provides functions that are hardware-agnostic, relying on a format of data that

46 can be achieved from any data source. The **eyeRead** package is designed for the analysis of eye

47 data from reading exercises. The **emov** package offers a limited set of functions and is primarily

48 designed for fixation detection, using the same dispersion algorithm used in eyetools. Finally,

49 **eyetrackingR** is perhaps the most comprehensive alternative package available on CRAN. It has

50 functions for cleaning data and various plotting functions, including analysis over time. It does

51 not feature algorithms for detection of fixations or saccades, instead working with raw data [is this

52 true?].

53   [insert table of features here]

⁵⁴ In this tutorial we demonstrate the pipeline of analysis functions within eyetools. The

⁵⁵ package has been designed to be simple to use by someone with basic knowledge of data handling

⁵⁶ and analysis in R. It should appeal to researchers who are working with raw eye data for the first

⁵⁷ time, as well as those accustomed to working with eye data in other environments who wish to

⁵⁸ transfer to working in R.

⁵⁹ This tutorial is separated into five distinct sections. In the first section, we briefly describe

⁶⁰ the basic methodology of collecting eye data in general, and in regard to the specific dataset we

⁶¹ use to illustrate all the functionality of the eyetools package. The second section covers the

⁶² process for getting data from an eye tracker into an eyetools-friendly format. The third section

⁶³ introduces the foundational functions of the eyetools package, from repairing and smoothing eye

⁶⁴ data, to calculating fixations and saccades, and detecting time spent in Areas of Interest (AOIs).

⁶⁵ The fourth section takes the processed data, and applies basic analysis techniques commonplace in

⁶⁶ eye data research. In the fifth and final section, we reflect on the benefits of the eyetools package,

⁶⁷ including contributions to open science practices, reproducibility, and providing clarity to eye

⁶⁸ data analysis.

Data Collection

⁷⁰ First describe basic paradigms for collecting eye data. Also purpose etc. @tom

⁷¹ Then describe the specifics of the dataset we are using - I presume this is the HCL dataset

⁷² in full? Should make mention of the fact that the workflow can be done either with the full

⁷³ dataset, or the two participant dataset provided in package.

Converting Raw Data

⁷⁵ @matthew, @tom

⁷⁶ Owing to the vast range of eye tracking hardware available, eyetools does not offer much

⁷⁷ in the way of converting raw data into the eyetools format of participant ID, trial number,

⁷⁸ timestamp, x, and y coordinates. In this section, we cover the stages of transforming the data from

⁷⁹ a TOBII eye tracker.

⁸⁰ `hdf5_to_csv()`

81   When you have data that is in a binocular format, that is you have a set of coordinates for

82 each eye, it needs to be converted into single x and y coordinates for both eyes combined. Using

83 eyetools, this can be done in one of two ways, either taking an average of the coordinattes from the

84 two eyes, or by choosing the eye with the fewest missing samples is used to represent the data. An

85 averaging of the two coordinates sets is the typical method of combining binocular data, and can

86 be done using the `combine_eyes()` function in eyetools. This returns a flattened list of

87 participant data that has x and y variables in place of the left_* and right_* variables.

<div align="center">

Working with eyetools

</div>

89   In the last section, we finished with the data in a format that holds participant ID, trial

90 number, a timestamp, along with x and y coordinates. This is the format expected by eyetools

91 when working with multi-participant data, however if you some reason you are working with a

92 single participant then the participant ID column is superfluous and can be dropped. This basic

93 data format of ID, trial, time, x, and y ensures that eyetools is applicable to a variety of eye data

94 sources and does not depend on specific eye trackers being used.

95 *Counterbalanced designs*

96   Many psychology experiments will position stimuli on the screen in a counterbalanced

97 fashion. For example, in the example data we are using, there are two stimuli, with one of these

98 appearing on the left and one on the right. In our design, one of the cue stimuli is a "target" and

99 one is a "distractor", and the experiment counterbalances whether these are positioned on the left

100 or right across trials.

101   Eyetools has a built in function which allows us to transform the x (or y) values of the

102 stimuli to take into account a counterbalancing variable: `conditional_transform()`. This

103 function currently allows for a single-dimensional flip across either the horizontal or vertical

104 midline. It can be used on raw data or fixation data. It requires the spatial coordinates (x, y) and a

105 specification of the counterbalancing variable. The result is a normalised set of data, in which the

106 x (and/or y) position is consistent across counterbalanced conditions (e.g., in our example, we can

107 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. To transform the data, we require knowledge of where predictive cues were presented. Using this, `conditional_transform()` can align data across the x or y midline, depending on how stimuli were presented. In the experimental design used in our study, cues were presented either on the left or the right, so by applying `conditional_transform()`, all the predictive cues in the dataset are recorded on the same side.

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 no data is present for where the eyes would be looking. To mitigate this issue, the `interpolate()` function estimates the path taken by the eyes 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.

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 accessed easily using the following code:

```
  pNum missing_perc_before missing_perc_after
1  118           0.02314313          0.015838577
2  119           0.01214128          0.005402579
```

As shown, not all missing data has been replaced, this is because when gaps are larger than a given size they are kept as missing data due to it being unreasonable to try to estimate the path

134 taken by the eye. The amount of missing data that will be estimated can be changed through the

135 maxgap parameter.

136   Once missing data has been fixed, a common step is to smooth the eye data to remove

137 particularly jerky eye movements. To do this, `smoother()` reduces the noise in the data by

138 applying a moving averaging function. The degree of smoothing can be specified, as well as

139 having a plot generated for random trials to observe how well the smoothed data fits the raw data.

140
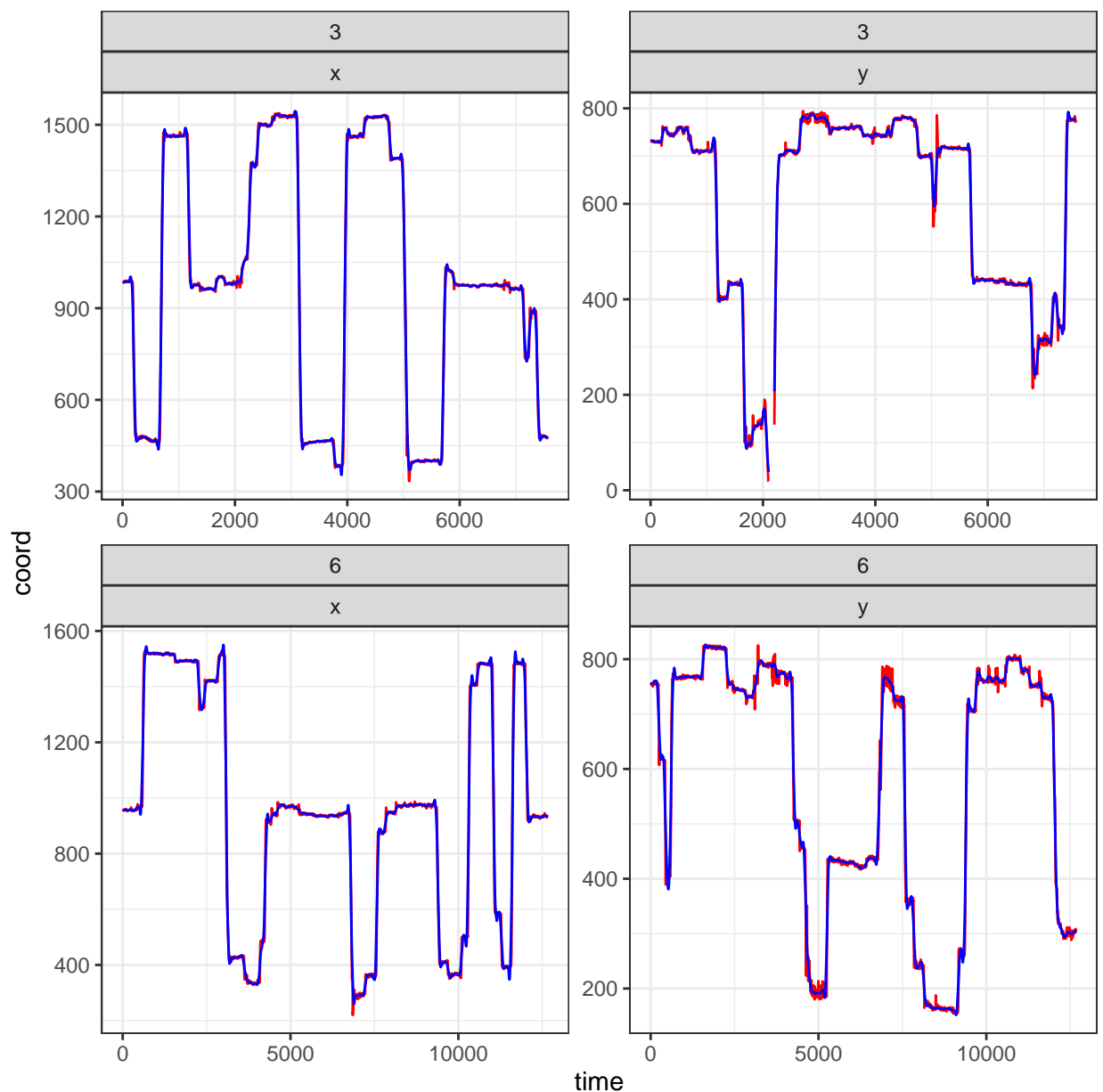
Fixations

Once the data has been repaired and smoothed, a core step in eye data analysis is to identify fixations (Salvucci & Goldberg, 2000), defined as when the gaze stops in a specific location for a given amount of time. When the eyes are moving between these fixations, they are considered to be saccades. Subsequently, data can be split into these two groups, fixations and saccades. 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?].

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).

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.

Time spent in AOIs can also be calculated from fixations or raw data using the

168 `AOI_time()` function available. This calculates the time spent in each AOI in each trial, based on

169 the data type given, in our case fixation data.

170   If choosing to work with the raw data, there is also the option of using

171 `AOI_time_binned()` which allows for the trials to be split into bins of a given length, and the

172 time spent in AOIs calculated as a result.

173   **Analysing eye data**

174   @tom

175   **Discussion**

176   In the present tutorial, we began by identifying the current gap in available tools for

177 working with eye data in open-science pipelines. We then provided an overview of the general

178 data collection process required for eye tracking research, before detailing the conversion of raw

179 eye data into a useable eyetools format. We then covered the entire processing pipeline using

180 functions available in the eyetools package that included the repairing and normalising the data,

181 and the detection of events such as fixations, saccades, and AOI entries.

182 @SOMETHING_ON_THE_ANALYSIS_GOES_HERE.

183   From a practical perspective, this tutorial offers a step-by-step walkthrough for handling

184 eye data using R for open-science, reproducible purposes. It provides a pipeline that can be relied

185 upon by novices looking to work with eye data, as well as offering new functions and tools for

186 experienced researchers. By enabling the processing and analysis of data in a single R

187 environment it also helps to speed up data analysis.

188 **Advantages of Open-Source Tools**

189   eyetools offers an open-source toolset that holds no hidden nor proprietary functionality.

190   **Data Availability**

191   The data required for reproducing this tutorial is available at: @URL. A condensed

192 version of the dataset (starting with the `combine_eyes()` function) is a dataset in the eyetools

193 package called HCL.

Code Availability

The code used in this tutorial is available in the reproducible manuscript file available at:

References

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

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.