The Perception Engineer's Toolkit for Eye-Tracking data analysis

Thomas C. Kübler
thomas.kuebler@uni-tuebingen.de
Wilhelm-Schickard Institut
Tübingen, Germany

Figure 1: Examplary sample-based gaze density map on Yarbus' experiment while viewing "the unexpected visitor" by Repin.

ABSTRACT

Tools for eye-tracking data analysis are as of now either provided as proprietary software by the eye-tracker manufacturer or published by researchers under licenses that are problematic for some use-cases (e.g., GPL3). This lead to repeated re-implementation of the most basic building blocks, such as event filters, often resulting in incomplete, incomparable and even erroneous implementations.

The Perception Engineer's Toolkit is a collection of basic functionality for eye-tracking data analysis under a friendly CC0 license that allows for easy integration, modification and extension of the codebase. Methods for data import from different formats, signal pre-processing and quality checking as well as several event detection algorithms are included. The processed data can be visualized as gaze density map or reduced to key metrics of the detected eye movement events. It is programmed entirely in python utilizing high performance matrix libraries and allows for easy scripting access to batch-process large amounts of data.

The code is available at https://bitbucket.org/inserted_after_blind_review

Etra '20, June 03–05, 2018, Stuttgart, GER

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

eye tracking, signal processing, event detection, data analysis

ACM Reference Format:
Thomas C. Kübler. 2020. The Perception Engineer's Toolkit for Eye-Tracking data analysis. In *Etra '20: ACM Symposium on Eye-Tracking Research and Applications, June 03–05, 2020, Stuttgart, GER.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Analysis of eye-tracking data is not trivial and the selection of suitable algorithms and parameters often depends on the recording device and experimental settings. As an example, a plethora of event detection algorithms exist (methods to separate individual eye-tracker samples into fixations and saccadic eye movements) and their applicability depends on the sampling speed of the eye-tracker, the expected precision of the gaze signal, and other factors. Therefore, most researchers choose to leave the task of algorithm and parameter selection to the manufacturers of their recording devices - and license expensive closed-source software together with the purchase of their devices. There are solid arguments to do so (reproducibility, reasonable default settings). On the other hand, it is essential that the community of eye-tracking researchers retains the ability to apply customized and novel methods as well as

Table 1: Overview on eye-tracking data analysis software, its capabilities (Event detection, importing custom data formats, recording data, visualization, scripting or command file interface and licensing). Various other tools that cover individual aspects of these capabilities do exist.

Name	Events	Import	Record	Visual	Script	Interf.	License
Tobii Studio	IVT	-	√	√	√	GUI / SDK	Proprietary
D-Lab	unspecified	-	√	√	-	GUI	Proprietary
imotions	unspecified	√	√	√	-	GUI	Proprietary
SMI BeGaze2	IVT, IDT	-	√	√	-	GUI	Discontinued
EyeLink DataViewer	IVT variant	-	√	√	-	GUI	Proprietary
NYAN	-	-	√	√	-	GUI / SDK	Proprietary
OGAMA [15]	IDT	√	√	√	√	GUI / C#	GPL3
Eyetrace [12]	IVT, IDT, GMM	√	-	√	-	GUI / C++	unspecified
EyeMMV [10]	2-step Dispersion	√	-	√	√	MATLAB	GPL3
GazeAlyze [2]	IDT	√	-	√	√	MATLAB / GUI	unspecified
ILAB [5]	IDT	√	-	√	√	MATLAB / GUI	unavailable
gazetools [7]	IVT	√	-	-	-	GNU R	unspecified
Eye Movement Classification [9]	IVT, IDT, IHMM, IMST, IKF	√	-	-	-	MATLAB	BSD
PyGaze [3]	IDT variant		√	√	√	Python	GPL3
Perception Engineer's Toolkit	IVT, IDT, HMM, GMM	√	-	√	√	Python	CC0

the option to compare different approaches against each other and to quantify the effect of different algorithm and parameter choices. Therefore, it is essential to base the novel work on solid ground.

Curiously (and somewhat consequently) textbooks on eye-tracking even contain chapters on how to compute event detection in an Excel sheet [6]. Also, derived data (such as fixations and saccades) are common parts of eye-tracking data formats [17], even though they could in theory be recalculated on demand anytime (even though reaching the exact same results is surprisingly difficult in practice). However, some of these very basic methods contain quite complex implementation details (e.g., how does I-DT handle approaching a gap in the data signal? Should the duration of a fixation include half of a sample duration before and after the samples assigned to the fixation?).

Asking for further extensibility in proprietary software is likely not the way to go (as companies are not eager to maintain compatibility with third party code and doing so might actually endanger reproducible results generated with the own software).

Researchers on the other hand have created a variety of different toolsets over the last years. Contrary to other disciplines, no gold standard (such as OpenCV or tensorflow) has emerged. This is in part also due to the licensing regulations chosen by researchers: Licenses such as GPL3 are difficult to handle for commercial use-cases and therefore deemed to remain of academic use only. Probably even worse to handle is software without proper licensing, often resulting in no legal use being possible at all due to personal or institutional copyright.

So, while companies offer well maintained and tested software, researchers need to re-implement the same basic methods again and again to reach a state where novel implementations can be added.

This toolkit offers the ability to combine, modify and extend basic building blocks in eye-tracking data analysis. Based upon these ready-made components, one can rapidly implement advanced analysis methods.

Through a strictly modular design, future components that might depend on more restrictive licenses (e.g., LGPL) could easily be enabled or dropped depending on the current use-case.

Naturally, eye-tracking data analysis software is based on a graphical user interface (GUI) so that usage is simple and accessible to most people. However, scripting interfaces such as command line programs often provide better means of batch processing and piping data. Imagine you just created a chain of processing steps that works reasonably well on your data. Some months later, you want to reapply the same processing to additional data and run the exact same analysis again. While most software will enable you to reconstruct the processing steps and parameters from a project history or log files, it still involves pressing a lot of buttons and manually adjusting parameters. With a scripting-enabled application, one can simply re-run the pipeline with additional input data without further adjustment. Similarly, imagine you want to apply an event detection method to your data and to brute-force different settings for, e.g., the I-DT dispersion threshold parameter. For GUI applications that would usually mean a manual adjustment of the parameter within the GUI, re-running event detection and manual export of the results to post-process e.g. within MATLAB or python. Within a scripting-enabled environment this boils down to a single line of code that re-runs the pipeline with an increment parameter.

Over the last years scripting languages such as MATLAB and Python have proven their power in the processing of large scale experimental data as well as their relative ease of use also for non computer scientists. With the Perception Engineer's Toolkit we present a computer scientists approach to eye-tracking data analysis software that aims at alleviating the problems with existing software mentioned above by being scripting enabled by default, licensed under CC0 as well as being modular with a clean API interface.

1.1 State of the art

A probably non-complete overview of existing toolsets is provided in Table 1. Note that the extend to which the tools cover individual aspects might be very different, e.g., Eyetrace offers a rich variety of visualization methods while Pygaze focuses more on the recording aspect with rather basic visualization capabilities.

Probably closest to the presented approach is the software GazeAlyze [2], which also offers a pipeline from data import over event detection up to the calculation of eye movement features. The tool is based on MATLAB and can be configured via a command file. However, no explicit license is provided and MATLAB is required in order to run the tool.

2 METHODS

The toolbox is implemented entirely in python. High computational performance is achieved by massive use of numpy[16] and Tabel[1] libraries. Contrary to e.g. Pandas, Tabel allows to create views (i.e., slices) of data matrices that guarantee no in-memory copy of the data. As such an operation is very frequent for many signal processing steps, this pays off in a significant run-time performance improvement (as no time-intense copying of data in memory is required). Besides this important implementation detail, it offers easy data access similar to Excel or MATLAB matrices.

Each processing step is encapsulated in a *Command* structure. A command is able to access one or more *Datamodel* objects that are provided to it, each containing one eye-tracking trial. Commands are able to read and modify these trials, e.g., to perform steps such as filtering or event detection. They are grouped into several categories that determine their responsibility, such as loading and writing data from/to file, preprocessing, event detection, or visualization. Each such responsibility is encapsulated as a plugin interface. Individual plugins can be loaded through the lightweight Yapsy plugin system[2].

Commands can either be executed from a python shell (and as such scripted) or pushed to a command stack via a command file parser. The command file is an human-readable YAML file that can easily be modified in a text editor. This file specifies the order in which commands are stacked as well as their parametrization. An example is shown in listing 1.

The whole toolbox is thoroughly documented and all parameters, their units, defaults and reasoning behind them is contained in the docstrings alongside the code.

3 CONTENT

The workflow usually starts with a data import from text-formatted files. Presets for Tobii and SMI exports are provided. Other (custom) formats are covered by a customizable importer that allows to adjust the data separating character, the column names containing relevant data, a comment symbol, number of header lines, and many more parameters. Import and export of hdf5 data is also implemented, but requires a certain internal file layout.

Signal processing workflows include quality checking the tracking ratio as well as a visualization of tracking quality per trial (see Figure 2). Such a visualization can be extremely helpful as one can

[1] https://github.com/BastiaanBergman/tabel
[2] http://yapsy.sourceforge.net/

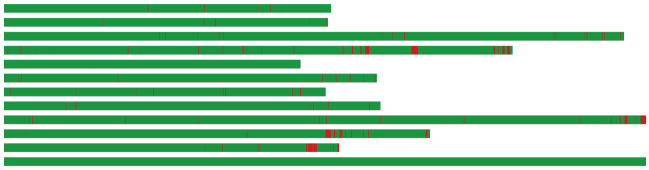

Figure 2: Data quality visualization of twelve trials (one per row) of different length. Green parts indicate valid data while red parts visualize data segments with invalid samples. In these trials tracking loss is interspersed and overall quality is high, as opposed to trials where tracking might get completely lost at some point during the trial and not recover.

easily check whether trial lengths are similar. While for trials of few seconds duration the total tracking rate (i.e., the ratio of valid samples in the data) is a good measure of data quality and a trial exclusion criteria, this does not hold for very long recordings (e.g., an hour of driving), where overall bad tracking performance has to be distinguished from temporary tracking losses (e.g., because of driving against the sun). Moving average and Median filter of the gaze X- and Y-coordinates as well as filling short tracking losses via linear interpolation are also implemented.

Listing 1: Command file in YAML format that defines the workflow and parameters of used algorithms. This file loads Tobii's exported data and performs event detection. Non-specified parameters (such as filter lengths) are reasonably defaulted.

Commandlist.YAML

```
- {plugin: "PersistenceCSV", action: "read",
   parameters: { filename: "data.txt",
    separator: "\t",
    timestamp_to_ms_factor: 0.001,
    aliases: {
      "Time": "Time",
      "X": "L_POR_X_[px]",
      "Y": "L_POR_Y_[px]"}
    }
  }
- {plugin: "PreprocessGapFill",
   parameters: {max_gap_length: 70} }
- {plugin: "PreprocessMedianFilter"}
- {plugin: "EventdetectionIHMM"}
```

Event detection can be performed either via I-DT, I-VT [13], using HMMs with Gaussian emission [13] or Gaussian Mixture Models [14]. See [1, 8] for a review on which method to use when.

Based on the detected eye movements, characteristic measures can be computed: for fixations the count, rate, average duration, dispersion standard deviation and ranges are reported. For saccades the count, rate, average duration, HV-ratio, average amplitude, velocity, peak velocity and time to peak.

There is also an experimental (not thoroughly tested) implementation for microsaccade detection [4] and scanpath comparison

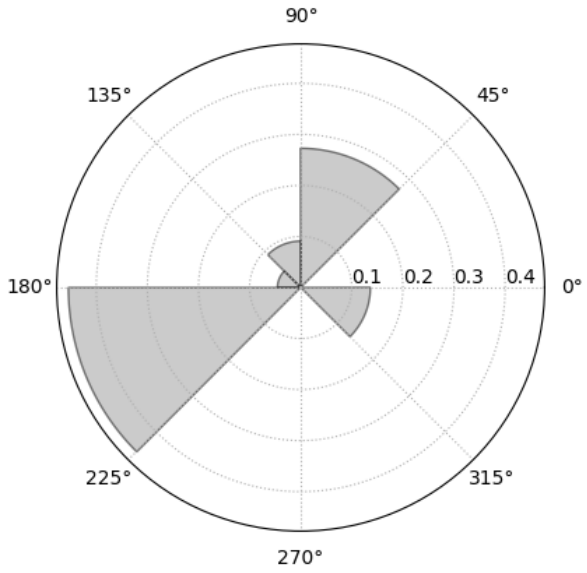

Figure 3: Visualization of saccade direction frequencies in eight angular bins.

using subsequence frequencies [11] as well as a novel approach to mine for relevant patterns via a genetic evolutionary algorithm.

Further, gaze density maps based on samples, fixation count and fixation duration can be created via superposition of Gaussian probability density functions and visualized as either heatmaps or shadowmaps (shown in Figure 1). The major saccade transition directions can be visualized in a polar plot via aggregation over angular bins (Figure 3).

Overall, the Perception Engineer's Toolkit is a versatile set of basic signal processing building blocks specific for the post-experimental analysis of eye-tracking recordings. Typical analyses that commercial software provides can be reproduced. Furthermore, it offers scripting access, e.g., to enable studying of the influence of parameter choices.

ACKNOWLEDGMENTS

REFERENCES

[1] Richard Andersson, Linnea Larsson, Kenneth Holmqvist, Martin Stridh, and Marcus Nyström. 2017. One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. *Behavior research methods* 49, 2 (2017), 616–637.

[2] Christoph Berger, Martin Winkels, Alexander Lischke, and Jacqueline Höppner. 2012. GazeAlyze: a MATLAB toolbox for the analysis of eye movement data. *Behavior Research Methods* 44, 2 (01 Jun 2012), 404–419. https://doi.org/10.3758/s13428-011-0149-x

[3] Edwin S Dalmaijer, Sebastiaan Mathôt, and Stefan Van der Stigchel. 2014. PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eye-tracking experiments. *Behavior research methods* 46, 4 (2014), 913–921.

[4] Ralf Engbert and Reinhold Kliegl. 2003. Microsaccades uncover the orientation of covert attention. *Vision research* 43, 9 (2003), 1035–1045.

[5] Darren R Gitelman. 2002. ILAB: a program for postexperimental eye movement analysis. *Behavior Research Methods, Instruments, & Computers* 34, 4 (2002), 605–612.

[6] Prof Kenneth Holmqvist and Dr Richard Andersson. 2017. *Eye Tracking - A Comprehensive Guide to Methods, Paradigms, and Measures* (2. aufl. ed.). CreateSpace Independent Publishing Platform, Ort.

[7] RM Hope. 2014. Gazetools: a collection of functions for processing and classifying eye gaze data.

[8] Oleg V Komogortsev, Denise V Gobert, Sampath Jayarathna, Sandeep M Gowda, et al. 2010. Standardization of automated analyses of oculomotor fixation and saccadic behaviors. *IEEE Transactions on Biomedical Engineering* 57, 11 (2010), 2635–2645.

[9] Oleg V Komogortsev and Alex Karpov. 2013. Automated classification and scoring of smooth pursuit eye movements in the presence of fixations and saccades. *Behavior research methods* 45, 1 (2013), 203–215.

[10] Vassilios Krassanakis, Vassiliki Filippakopoulou, and Byron Nakos. 2014. EyeMMV toolbox: An eye movement post-analysis tool based on a two-step spatial dispersion threshold for fixation identification. *Journal of Eye Movement Research* 7, 1 (Mar. 2014). https://doi.org/10.16910/jemr.7.1.1

[11] Thomas C Kübler, Colleen Rothe, Ulrich Schiefer, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2017. SubsMatch 2.0: Scanpath comparison and classification based on subsequence frequencies. *Behavior research methods* 49, 3 (2017), 1048–1064.

[12] Thomas C Kübler, Katrin Sippel, Wolfgang Fuhl, Guilherme Schievelbein, Johanna Aufreiter, Raphael Rosenberg, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2015. Analysis of eye movements with Eyetrace. In *International Joint Conference on Biomedical Engineering Systems and Technologies*. Springer, 458–471.

[13] Dario D Salvucci and Joseph H Goldberg. 2000. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*. 71–78.

[14] T. Santini, W. Fuhl, T. C. Kübler, and E. Kasneci. 2016. Bayesian Identification of Fixations, Saccades, and Smooth Pursuits. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications (ETRA)*. 163–170.

[15] Adrian Voßkühler, Volkhard Nordmeier, Lars Kuchinke, and Arthur M Jacobs. 2008. OGAMA (Open Gaze and Mouse Analyzer): open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior research methods* 40, 4 (2008), 1150–1162.

[16] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30.

[17] Stefan Winkler, Florian M Savoy, and Ramanathan Subramanian. 2014. X-Eye: A reference format for eye tracking data to facilitate analyses across databases. In *Human Vision and Electronic Imaging XIX*, Vol. 9014. International Society for Optics and Photonics, 90140L.

.