Eye Tracking Gaze Visualiser: Eye Tracker and Experimental Software Independent Visualisation of Gaze Data

Benedict C.O.F. Fehringer*
Department of Psychology of Education
University of Mannheim

Abstract

Eye tracking research in disciplines such as cognitive psychology requires specific software packages designed for experiments supporting reaction time measurement, blocking and mixing of conditions and item randomisation. Although recording raw eye movement data is possible, its visualisation is difficult regarding the experimental design. The currently used eye tracking software is often built as an all-in-one program that can only visualise the eye tracking data recorded by itself. Therefore, in this paper a software tool is presented that visualises nearly any recorded eye tracking gaze data on the corresponding video independent of the specific software that runs the experiment. Summarised visualisations over randomised item presentations according to experimental conditions can be created. In addition to basic visualisation functionalities, further features such as simple object detection, repetitive pattern exploration and subset selection of subjects are provided.

CR Categories: H.5.2 [User Interfaces]: Evaluation/methodology—Graphical user interfaces (GUI); H.1.2 [User/Machine Systems]: Software psychology—; H.3.2 [Information Storage]: File organization—;

Keywords: eye tracking, visualisation, gaze data

1 Introduction

During the last years an increasing amount of studies using eye tracking for a broad area of different disciplines can be observed. The higher the studies' demands are (e.g. randomised item presentation, playback videos, measurement of reaction times), the more requirements hard- and software have to fulfil. Yet, there is often only one program for creating and running the experiment, visualising the results and analysing the data. Whereas in most cases the data can be exported to be analysed in separate statistical software packages, only the program's own data can be visualised by itself. It becomes more and more difficult to find an all-in-one program that fulfils a research's special needs completely with respect to the experimental conception on the one hand and visualising the data in an appropriate way on the other hand. A solution would be to modularise the study process in single software components and optimise each of them individually (Figure 1).

The purpose of the present approach is the creation of a visualisation unit, that imports data independently of the experimental software and provides functionalities satisfying the special needs of highly standardised experiments (e.g. in cognitive psychology).

*e-mail: b.fehringer@uni-mannheim.de

ETRA 2014, March 26 – 28, 2014, Safety Harbor, Florida, USA.
2014 Copyright held by the Owner/Author.
ACM 978-1-4503-2751-0/14/03

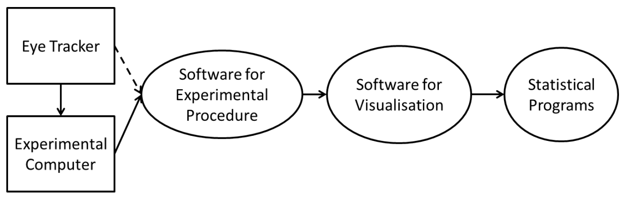

Eye Tracker
Experimental Computer
Software for Experimental Procedure
Software for Visualisation
Statistical Programs

Figure 1: *Modularised study process of an typical eye tracking experiment*

To point out the challenges for the visualisation of such experiments and to demonstrate our tool's usage, a one-person-case-study was conducted. The subject (female, 20 years, wearing no glasses or contact lenses) had to attend to a computer-based version of the chronometric test [Jansen-Osmann and Heil 2007]. It is a test to measure the mental rotation ability and consists of 60 items. An 'item' is one single task to be solved by the subject. In one task two two-dimensional figures (Figure 2) were presented. The right figure is either a rotated (*same*) or a rotated and mirrored (*different*) version of the left figure. The subject had to decide whether the figures are *same* or *different*. All items were presented in a random order and were based on 14 different figures. The test was realised with the experimental software *E-Prime*[1] connected to the Tobii Eye Tracker TX300 (300Hz). The **Item** number, the **Answers** (*correct*, *incorrect*), the **Kind** of Presentation (*same*, *different*) and the estimated coordinates of the point of gaze (POG) (**X**, **Y**) were recorded. Additionally, E-Prime documents if an AOI (area of interest) at a certain rectangular AOI-position is shown. In the case study, the AOI-positions were the two places where the figures were presented (**AOI1**, **AOI2**).

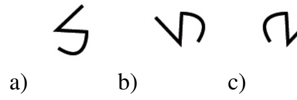


Figure 2: *Examples of two-dimensional figures for the chronometric test: b) is a rotated version of a) through* 90° *and c) is a rotated version of b)*

2 Related Work

Currently, there are different kinds of programs that support gaze visualisation. On the one hand, there is software belonging to a specific eye tracker and on the other hand, there are 'independent' tools to which in principle any kind of eye tracker can be connected.

The first group consists mostly of commercial products from eye tracker companies like Tobii[2] or SMI (SensoMotoric Instruments)[3]. To visualise recorded data by *Tobii Studio*^{tm} in an optimal way,

[1]http://www.pstnet.com/
[2]http://www.tobii.com/
[3]http://www.smivision.com/

the user has to use one of Tobii's own eye trackers and to carry out the experiment by *Tobii Studio*^{tm} itself. Although the usage of other experimental software (e.g. E-Prime) is possible, establishing the whole set-up (connection with the eye tracker and visualise the data) can be cumbersome and time-consuming. Furthermore, if in an experiment different conditions are tested by many items in random order, there is no (simple) method to generate an *average* visualisation of only one condition (or a combination of conditions) with all other items ignored. Similar drawbacks exist for *SMI BeGaze*^{tm} when using an alternative eye tracker and experimental software. However, the exclusive visualisation of items belonging to one condition can be better (but not perfectly) realised by using annotation and event filters.

The programs of the second group are mostly free and/or open source software like *iComponent* [Špakov 2008] (University of Tampere) or *OGAMA (open gaze and mouse analyzer)* [Voßkühler et al. 2008] (Freie Universität Berlin). The first one is an all-in-one tool for which the user can choose an arbitrary eye tracker (that has to be supported by the software) and one of the experimental plug-ins (e.g. Video Viewer or Web Viewer) for designing the experiment. After recording, the gathered data can be visualised in two modes with various records of different subjects. For data-visualising, the *iComponent*-tool itself is needed. *OGAMA* also provides an interface for an arbitrary eye tracker to build a slide-show as an experiment. In contrast to many other programs (e.g. *iComponent*), it is also possible to import already recorded data that do not have to come from *OGAMA* itself. But to visualise the results of such data, the user can only import a set of images, i.e. no videos or compositions of several images are possible. A good overview about these and further eye tracking software is given by the COGAIN (Communication by Gaze Interaction) Network's wiki[4].

Summing up, the problems of all software packages are to visualise data that are not generated by the program itself and to provide an adequate *average* visualisation with respect to different conditions. What is missing is a gaze visualisation program that offers importing the presentation of a conducted experiment, merging it with the corresponding raw data (containing gaze coordinates, conditions, etc.) and synchronising them - irrespective of the used program and eye tracker. Moreover, with respect to a high standardised experiment, the visualisation features have to fulfil special needs regarding the item's assignment to specific conditions, the selection of certain conditions and the search for repetitive patterns over many items of the same condition.

3 Eye Tracking Gaze Visualiser

In order to overcome the problems explained in Section 2, we implemented the Eye Tracking Gaze Visualiser (ETGV) as a gaze visualisation software that is nearly independent of the used eye tracker and the experimental software. Basically, the ETGV needs a screen recording of the conducted eye tracking experiment and the corresponding data file containing the necessary eye tracker information. After merging both, a visual inspection can be done. The ETGV is not created to record gaze data by itself. Also, it is not intended to do data analysis which goes beyond that what is needed for the visualisation.

The program is written in Python [van Rossum 1995]. For the GUI the package wxPython [Talbot 2000] was used, the video representation and manipulation was done with openCV [Bradski 2000] and for data computations numpy, scipy [Oliphant 2007] and scikit-learn [Pedregosa et al. 2011] were used.

[4]http://wiki.cogain.org/index.php/Eye_Trackers

The tool supports various POG-visualisations and a heat-map presentation. It can show AOIs, visualise Hidden Markov Models (HMMs) and provide some multi-subject representations. For both, heat-maps and HMMs, it is possible to filter the data with respect to certain conditions for an *average* visualisation. The theoretical concept of the tool is sketched in Figure 3, whereas the GUI of the realisation is shown in Figure 4.

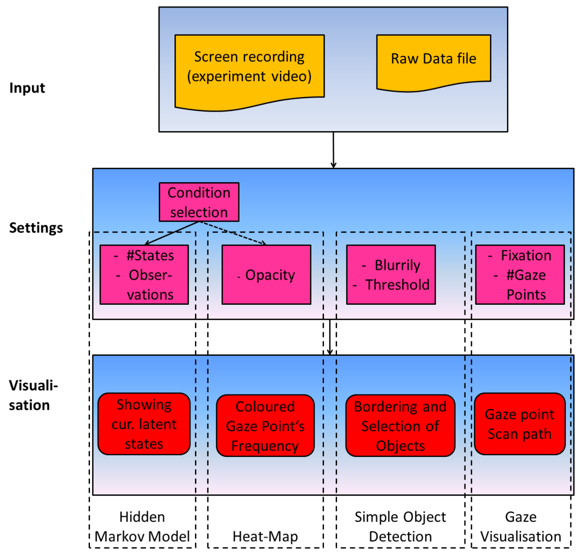


Figure 3: *Sketch of the Eye Tracking Gaze Visualiser*

3.1 Input and Synchronisation

Firstly, the screen recorded video of the experiment and the data file has to be imported (Figure 4, black framed GUI-elements), where each line of the data file corresponds to one measurement point of the eye tracker (*In the case study: 300 lines per second.*). For the data file, the meaning of the single columns is important to do the correct computation (i.e. the program needs to know which column contains the POG's x- and y-coordinate). So, a 'meta-file' has to be read that holds the names of all needed columns (tabulator-separated) in a predefined order. This file has to be created by the user and replaces an importing procedure for the data file. Internally, the data are represented as a data frame from the pandas library[5]. So, the columns can be easily addressed if the names are known by the meta-file. A dictionary is used as data structure, that matches the meanings as keys with the column names (from the meta-file) as (lists of) values. Besides the subject number and the x- and y-coordinates, it also needs - if existent - the set of columns that define the conditions (Subsection 3.4), predefined AOIs (Subsection 3.2) and other optional columns to define the manifest observations for the HMM (Subsection 3.6):

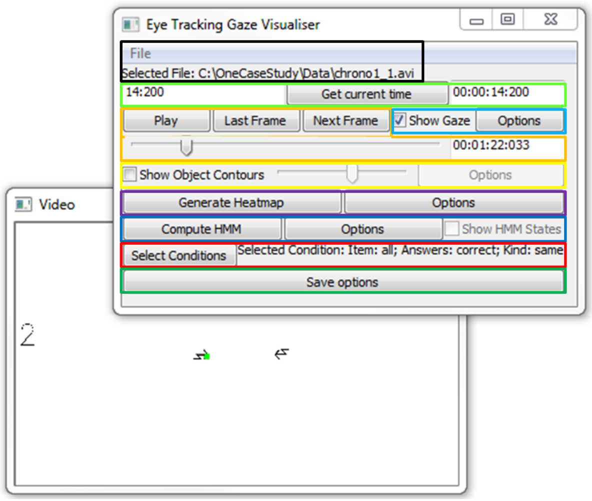
[Subject] [X] [Y] ([Cond])* ([AOI])* ([HMM-observations])*

where * means that 0 or a positive number of columns can be comma-separated listed. For the case study, it looks like:

Subject X Y Item,Answers,Kind AOI1,AOI2

Table 1 shows the corresponding part of the case study's data file. The concrete contours of the rectangular AOI-positions have to be inserted before by entering the four corners' coordinates. Arbitrarily shaped AOIs can be imported by an extra file that contains a list

[5]http://pandas.pydata.org/



Selected File: C:\OneCaseStudy\Data\chrono1_1.avi

14:200

00:00:14:200

00:01:22:033

Selected Condition: Item: all; Answers: correct; Kind: same

Figure 4: *GUI of the ETGV: Control unit with video window*

of the points that border the corresponding AOI. Not a priori known AOIs can be defined via Simple Object Detection (Section 3.3).

Subj.	X	Y	Item	Answ.	Kind	AOI1	AOI2
1	0.6	0.5	2	incor.	diff.	Fig7	Fig7_m
…	…	…	…	…	…	…	…
1	0.5	0.5	4	incor.	same	Fig4	Fig4_s
…	…	…	…	…	…	…	…
1	0.4	0.5	4	incor.	same	Fig4	Fig4_s

Table 1: *A data point selection from the data file of the conducted case study.*

After importing, the data and video file have to be synchronised. To this end, a video frame of the screen recorded experiment is needed that can be assigned to a specific data file time stamp. Usually, this can be done in two ways: a) The time that is written in the data file can be explicitly printed out on the screen at the experiment's beginning. b) The user has to look for a switch between two frames which is also documented in the data file (e.g. change from one item to the next item). If this time correspondence is found, the data file's delay with respect to the video can be manually entered to bring both in phase (Figure 4, light green). Multi-subject visualisations are supported, too.

3.2 Gaze Visualisation

For the gaze visualisation (Figure 4, light blue) it can be chosen between single dots or a scan path over a certain time interval. In the former case, the POG with the closest time stamp is drawn on each video frame. If there are fewer frames than POGs per second, the user can decide to draw more than only one POG per frame as long as the drawn POG has no time stamp closer to another frame. That means for a video with 30 fps and an eye tracker with 300 Hz, maximal 10 POGs can be drawn on each video frame. The expected x- and y-coordinate values in the data file has to be between 0 and 1 as a relative representation of the position (Table 1). Together with the frame width and height from the video, the absolute points can be computed and drawn on the video using openCV.

The scan path connects all gaze points in the correct time order with an edge. However, in opposite to the dot representation, there is no choice between one or more POGs per frame; all existing POGs will be drawn. Manually, the user can set the width of a shifting time window, that means, the amount of POGs that are allowed to be shown at the same time. If the maximal number of POGs is reached, for each new POG, the oldest one will be removed.

For both representations, indicating fixation by growing dots is selectable. To identify fixations and saccades, the user can choose between one of the algorithms presented by [Salvucci and Goldberg 2000]: *Velocity-Threshold Identification*, *HMM I.*, *Dispersion-Threshold I.*, *Minimum-Spanning-Tree I.* and *Area-of-Interest Identification*. For future data analysis, the computed fixations and saccades can be stored - together with all other data - in a new data file (Figure 4, dark green). Furthermore, it is possible to visualise all subjects at the same time using different colours. Also, the predefined AOI-position can be shown by coloured AOI frames, where the colour switches, when the POG is on this AOI.

3.3 Simple Object Detection

The simple object detection is then useful, if an AOI could not be defined in advance. For instance, if an item contains a video with a jumping ball and the experimenter is interested, whether the subject is looking at the ball. With this feature the ball can be marked as AOI in the first frame and for the following frames, it can automatically be checked, whether the subject is looking at it. The simple object detection (Figure 4, yellow) consists of two parts: In the first one, the focus lies on the detection of possible objects in the current frame. In the second part, the prior detected and selected objects in former frames should be recognized in the following frames. For all image manipulations, the openCV library was used.

A simple version of the *Canny Edge Detector* is used as object detection: At first, the frame is transformed in a grey-scaled image. Then, optionally, the image can be blurred with a Gaussian filter for better detection of unconnected object borders. Finally to the image, the Binary (inverted) threshold function is applied:

$$int\_new(x,y) = \begin{cases} 0 & \text{if } int\_old(x,y) > thresh \\ maxVal & \text{otherwise} \end{cases}$$

where (x, y) are the image's coordinates, int_new and int_old the new and old intensity values, $maxVal$ the new maximum intensity value and $thresh$ the entered threshold.

This function looks for jumps in the image's intensity to identify potential object borders. The so found contours are sketched in the frame. All single parameters can be manually changed, until the desired contours that frame the interested objects, are found. If the result is satisfying, the contours can be selected via mouse click and recognised by the tool in the following frames.

For each further frame, an object detection is carried out as described above, using the same parameter configuration. All found contours of the new frame are compared to the selected ones from the reference frame with respect to overlapping shapes and the position. A tolerance value for deviations can be adjusted. Also, the user can store whether the estimated gaze point is on that object in a new data file (Figure 4, dark green).

3.4 Condition selection

Especially in the context of standardised experiments with different conditions of the stimulus material, it can be interesting to select items from the same conditions (Figure 4, red) to do an *average* visualisation. Therefore certain conditions and combinations of conditions can be filtered via check boxes, so that all data points that do not belong to the selected conditions will be ignored. It can be used for the heat-maps (Subsection 3.5) and the HMMs (Subsection 3.6)

3.5 Heat-maps

In the standard case, a heat-map (Figure 4, purple) is generated over the item that the current video frame belongs to. However, the user is free to change the beginning and the end of the covered time interval to arbitrary positions. The heat-map can also be computed over selected conditions (Subsection 3.4) and more than one person.

The heat-map is computed using the *heatmap* library[6] to generate and store a heat-map image with a transparency α-channel. In a next step the generated image is loaded and lain over the current frame with a certain opacity, which is manually adjustable.

3.6 Hidden Markov Models

The idea to compute a HMM (Figure 4, dark blue) is based on the assumption that the eye movements across items from the same condition of a standardised experiment are comparable. So, if all of these items would be concatenated, a repetitive pattern should be observed. To reveal such a pattern a sequence algorithm, like the HMM, is needed. To compute the HMM's latent state model, the interested conditions has to be selected (Subsection 3.4). So, all items from the selected conditions are used, whereas all other items are ignored. Without visualisation the interpretation of the resulting model, which assigns to each POG a state, can be difficult. If one is interested in the underlying cognitive processes, it is necessary to see how eye movement within a certain state looks like.

The HMM is computed by the scipy-package using the Baum-Welch-algorithm as described in [Rabiner 1989] with a Gaussian distribution. Parameters like the number of states can be selected by the user. Also the user can decide which data are interpreted as manifest observations for the HMM by choosing a column of the data file, specified in the meta-file. After the HMM-model is computed, the states can be stored in a new data file (Figure 4, dark green) and can be indicated in the gaze visualisation mode by a label on each frame (Figure 4, shows the state 2).

4 Application of the ETGV in the case study

The focus of the visual analysis was the comparison between the correctly answered *same*-items and the incorrectly answered *same*-items. The aim was to detect differences that indicate good and bad performance. At first the corresponding conditions were selected [(*correct*, *same*) or (*incorrect*, *same*)]. Then the average heat-map for each selection was created. It shows that in case of the correct answers, some of the subject's POGs were beside the presented figures. This could be confirmed by the computed HMMs, each with three states. Regarding the relative frequencies of the single states, state 1 was more often in the correct answered items (relative frequency: 0.101) than in the incorrect ones (0.026). The difference was marginally significant [t(22)=1.781, p=0.089]. Through the visual inspection, it could be detected that state 1 corresponds to fixations that are on neither of both figures. If and how this result can be interpreted has to be further examined.

5 Conclusion

The ETGV is a tool for exclusively visualising gaze data on a screen recorded video of an experiment. The program is independent of the used eye tracker and the experimental software as long as the needed data are contained in the recorded file. Beside of standard visualisation features, it provides condition selection, a simple object detection for AOIs and HMMs. Also, the results can be stored in a new data file for further data analysis.

Nevertheless, there are some limitations. The current meta-file solution is cumbersome and restricts the data files that have to be imported. So, an extra importing procedure is intended, which allows for different data structures. Secondly, the screen recording quality is restricted by hardware performance. However, regarding the constant increase in CPU- and graphic card-performance, the restrictions concerning the quality are becoming smaller. The third limitation is the needed manual synchronisation, which is not solvable without an explicit exporting feature of the program that collects the data.

[6]http://jjguy.com/heatmap/

6 Future Work

The current program is part of an on-going work that towards building to a more powerful visualisation tool with a broader set of functionalities. The main focus will lie on features that make use of the standardisation and the fact that there are many items of the same kind. In this way, the extraction of repetitive (parts of) gaze pattern (over many items) and their interpretation via visualisation should be achieved.

References

BRADSKI, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

JANSEN-OSMANN, P., AND HEIL, M. 2007. Suitable stimuli to obtain (no) gender differences in the speed of cognitive processes involved in mental rotation. *Brain and Cognition 64*, 217–227.

OLIPHANT, T. E. 2007. Python for Scientific Computing. *Computing in Science & Engineering 9*, 3, 10–20.

PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

RABINER, L. R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE 77*, 2, 257–286.

SALVUCCI, D. D., AND GOLDBERG, J. H. 2000. Identifying Fixations and Saccades in Eye-Tracking Protocols. In *Proceedings of the Eye Tracking Research & Applications Symposium 2000*, ACM, 71–78.

ŠPAKOV, O. 2008. *iComponent Device-Independent Platform for Analyzing Eye Movement Data and Developing Eye-Based Applications*. University of Tampere, TamPub.

TALBOT, H. 2000. wxPython, a GUI Toolkit. *Linux Journal 2000*, 74es (June).

VAN ROSSUM, G. 1995. Python Library Reference, CS-R9524. Tech. rep., Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

VOSSKÜHLER, A., NORDMEIER, V., KUCHINKE, L., AND JACOBS, A. M. 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, 1150–1162.