Towards a tool for visualizing pupil dilation linked with source code artifacts

Constantina Ioannou, Per Bækgaard, Ekkart Kindler
DTU Compute
Technical University of Denmark, Lyngby, Denmark
coio@dtu.dk, pbga@dtu.dk, ekki@dtu.dk

Barbara Weber
Institute of Computer Science
University of St. Gallen, St. Gallen, Switzerland
barbara.weber@unisg.ch

***Abstract*—Recent eye tracking research in the field of software engineering has proposed novel visualizations linking developer's gazes with the source code artifacts to better understand how developers comprehend source code artifacts potentially consisting of several different files. In addition, it is well established that cognitive processes can be monitored by recording the change in pupil dilation. Recent pupillometry studies in the software engineering field have shown that pupil dilation can be used either as an indicator of cognitive load or task difficulty. We envision to create a tool for visualizing pupil dilation linked to source code artifacts that can help to better understand the cognitive processes of a developer during code comprehension tasks in terms of cognitive load. In this paper, we describe a feasibility study we conducted to enable a more fine-grained analysis of pupil dilation and we demonstrate some preliminary results.**

***Index Terms*—pupil dilation, code comprehension, cognitive load**

I. Introduction

The process of software development is a complex activity, highly iterative, interleaved and loosely ordered and it requires both technical knowledge and extensive abstraction capabilities [1]. Moreover, it requires developers to understand the requirements presented, to comprehend a large and complex system including other related software artifacts, and to form an internal representation of the problem in their working memory before performing maintenance activities (e.g., modifying a code snippet's functionality) [2], [3].

Existing work has used eye trackers to investigate developers' behaviour while they read source code [4], [5] and comprehend diagrams [6]. An overview of related eye tracking studies in the software engineering field is provided in [7]. Moreover, tools like iTrace/iTraceVis [8] which provide informative visualizations of gaze-based features linked with the source code artifacts have been developed, in order to enable a better understanding on how developers behave to comprehend software artifacts (e.g. source code snippets) or what there reading patterns are.

Eye trackers can also capture the pupil diameter. Pupil dilations and fixations are complementary features; pupil dilations have long been recognised in literature [9]–[11] as related to cognitive load and are under control of the autonomous nervous system whereas fixations are more often associated with attentional processes [12] that can at least partly be controlled by the user.

In particular, it was observed that the more demanding a cognitive task is, the greater the pupil dilation is [13]–[15]. In addition, researchers have explored the application of pupil dilation in several human computer interaction scenarios [16] and also in some code comprehension studies [17].

However, there is no current visualization solution which allows cognitive processes (e.g., cognitive load) to be linked with the actual content of software artifacts, in particular source code snippets. Consequently, we envision to provide a visualization tool for graphically presenting pupil dilations as an indicator of cognitive processes linked with where a developer fixated on software artifacts. This, in turn, will provide a more fine grained analysis of cognitive processes and thereby help to better understand and identify which cognitive processes of the developer may lead to coding bugs or bugs escaping their attention. Fig. 1 shows an example of our envisioned visualization. In the figure, the fixations of a developer are indicated as circles and the duration of each fixation as the size of the circle. Additionally, the fixations are colored based on the pupil dilations which is an indicator of cognitive load.

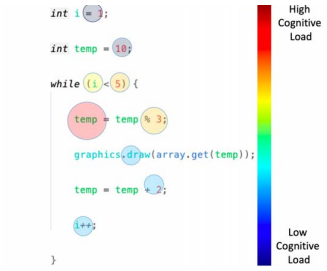

Fig. 1. Example of a developer reading source code. The location of fixations is indicated as circles. The size of the circle indicates the duration of a fixation. Additionally, the color of the circle shows the values of pupil dilation, an indicator of cognitive load

The remainder of this paper is structured as follows. In Section II, we discuss existing related work for analysing and visualizing eye tracking data (gazes and pupil). In Section III, we describe our method for extracting and visualizing pupil dilation. In Section IV, we present a feasibility study we conducted to validate our method to visualize the eye tracking data (gazes and pupil) along with some preliminary results. Finally, in Section V, we conclude with a summary and future work.

978-1-7281-9914-6/20/$31.00 ©2020 IEEE
DOI 10.1109/VISSOFT51673.2020.00016

II. Related work

There are several existing tools for analyzing and visualizing eye tracking data available online[1] and also summarized by a recent survey [18]. These tools can be categorized as:

1) Visualization for eye-movements
2) Visualization for pupil diameter

Many existing tools in the first category visualize the eye movements using heatmaps and gazeplots. For example, GazeVisToolBox[2] is a MATLAB package that provides visualization such as heatmaps and gazeplots. Moreover, SEQIT [19] is a visualization software designed for sequence analysis of eye tracking data. In addition, EyeCode[3] provides analysis of developers' gaze and visualizes the movement between lines of source code over time, shows the total number of fixations on a line, and also provides flow charts to show the developers' gaze movement.

In the second category, many of the tools were implemented as MATLAB and R packages and provide 2D graphs to visualize the processed pupil diameter in relation to time, e.g., PupilPreprocessing[4].

To combine eye movements and other physiological measurements (e.g., pupil diameter, electrodermal activity) for the needs of code comprehension research, CodersMuse was developed [20]. CodersMuse provides an environment for exploring synchronized, conjoint multi-modal data, specifically designed for program comprehension. The eye movements and other physiological measurements are integrated in the same environment, but still are visualized in different views, i.e., one of the views shows the eye movements overlaid on the software artifact (e.g., source code snippet) and other views show the physiological measurements as 2D graphs.

Additionally, physiological heatmaps is a visualization tool which links users' physiological measurements (e.g., pupil diameter, heart rate) with the eye movements on interfaces and HCI artifacts to help identifying regions where on the artifact users experienced an emotion [21].

While the aforementioned tools are sufficient to study how people read a fixed sized stimulus (e.g., a sentence or a few lines of source code), they are not suitable to study how developers comprehend an entire software system, because they are not able to visualize more than what fits on a screen at a time. Moreover, most of these tools besides physiological heatmaps, require the researcher to manually identify and define the areas of interest (AOI).

To address this, tools like iTrace/iTraceVis [22], [23] have been developed to provide an automatic detection of AOIs based on the gaze points and to visualize developers' gaze points on large source code files after a session where a developer is scrolling and switching between files. Using data from iTrace, iTraceVis provides a heatmap and static gaze map plot which are common visualization types for eye tracking data and also includes a line graph which visualizes how a developer reads lines of software artifacts. In addition, it includes a dynamic gaze map which enables the real-time playback of the eye tracking gazes of a developer. Similarly, another approach visualized eye tracking data from iTrace using process mining to investigate the reading patterns of developers [24].

As an extension to existing methods, we envision to provide a visualization tool that combines eye movements and other physiological measurements (e.g., pupil diameter) and graphically presents cognitive processes linked with the actual content of the source code artifacts (e.g., source code elements) where developers fixated on. This, in turn, will enable a more fine grained analysis and help better understand developers' cognitive processes during the software development process.

III. Proposed Visulization Method

To investigate the feasibility of visualizing pupil dilations linked to the actual content of where developers fixated on the source code artifacts, we designed and implemented the method shown in Fig. 2. This method consists of identifying fixations, extracting pupil dilations from pupil diameter data and graphically demonstrating the pupil dilation linked to where developers fixated on the software artifact (e.g. source code snippet).

The method takes as input the collected data from iTrace, a set of configured variables, a rendering of the software artifact used and then produces the visualization. An example of the produced visualization is shown in Fig. 3, and Fig. 4. In this section, we present details about each step of the method and the corresponding challenges are discussed.

A. Collect Data

The first activity is the collection of eye tracking data using iTrace [25], a plug-in for Eclipse. iTrace captures eye tracking data and links them to software artifacts while developer navigate and read source code. These data enable the link of a gaze point (x,y) to the respective source code element including the line and column number that developer looked at, they entail a timestamp (eye trackers' timestamp) and also the pupil diameter.

Pupil diameter is influenced by several factors, for example, by the change of light conditions, by the developers eyes' physiology and by the sleepiness of the developer [26]. To reduce likelihood of recording changes of the pupil diameter which are due to changes in the brightness of the room or screen, we suggest using a designated room where light conditions can be controlled as well as using experimental material that does not change the brightness of the screen.

Moreover, before recording data we suggest that the developers should be selected based on their vision capabilities and if they wear glasses they should be checked whether they are suitable to use (e.g., unscratched, without bi-focal lenses) [27]. In addition, a proper calibration needs to be performed

[1] *https://github.com/davebraze/FDBeye/wiki/Researcher-Contributed-Eye-Tracking-Tools*

[2] *https://www.mathworks.com/matlabcentral/fileexchange/56236-djangraw-gazevistoolbox*

[3] *http://emipws.org/sample-page/2013-analyzing-experts-gaze/visualizations/\#eyecode*

[4] *https://github.com/anne-urai/pupil_preprocessing_tutorial*

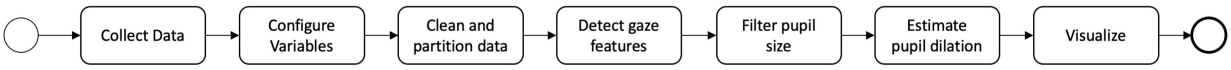

Fig. 2. The method of visualizing pupil dilations linked to the actual content of source code artifacts that developers fixated at.

and check needs to be made that eye tracking data can be recorded from the person.

B. Configuration

Once the data are collected, some variables need to be configured before the next steps of the method can be applied. Based on the equipment used during the experiment these variables can be determined: the operational frequency of the eye tracker, the name of the task under investigation, the velocity threshold for detecting fixations and saccades, the screen distance, the screen resolution, the minimum saccadic duration, and also the region for interpolating pupil diameter measurements before and after the blink.

C. Clean and partition data

Once the data are collected from all participants and the configuration is set, it is necessary to clean and partition the data before applying any analysis. First, we check the sampling rate of the collected data and ensure that the data loss is not greater than 2-3% of the sample. Next, we detect artifacts such as blinks and gazes which are out of the area of interest. Then, we partition the data sets based on source code files.

In the context of this paper, the area of interest (AOI) is defined as the gazes on the source code snippets. Because iTrace also captures gazes on other Eclipse views which are outside of the editor window we filtered those using linear interpolation.

D. Detect gaze-based features

After the data is cleaned and partitioned, we continue with detecting gaze-based features (i.e., the fixations and saccades). The process of detecting these features can be done in different ways. An overview of different algorithms identifying these gaze-based features can be found in [27]. In our case, we applied a velocity-based threshold algorithm similar to the ones described in [28] and obtained a time series of labelled gaze data.

E. Filter pupil diameter

Once the gaze-based features are extracted, we proceed with filtering the pupil diameter data. These data often contain outliers and missing data points that usually are generated due to blinks, look away moments, or glitches from the eye tracking device itself, and thus filtering is required before analysis. Filtering is not a trivial task and, in our method, we followed the guidelines provided in [29] and used linear interpolation.

F. Estimate pupil dilation

Once we filtered the pupil diameter measurements, we continued to estimate the pupil dilations. First, the average pupil diameter for each fixation was calculated and then we subtracted the grand mean of all fixations average pupil diameters, resulting in the pupil dilation for each fixation.

G. Visualizing

The visualization was achieved in two steps. First, we marked each fixation on top of the line/column of the linked source code element using a circle identifier (see Fig. 3 and Fig. 4). Next, each fixation was color-mapped based on the pupil dilation value estimated in the previous step. The pupil dilation values typically range between 0 – 0.3 mm. The colors are selected to form a gradient which ranges between 0 – 0.3 mm and changes every 0.05 mm to denote more visibly the change of pupil diameter.

Our visualization method does not preserve any temporal relationship between fixations, i.e., the scan path is not visible. To make it easier to see multiple overlaid fixations and to emulate smaller eye movements that occur during fixations (e.g., tremors, drifts, microsaccades), the exact location is jittered and moved underneath the relevant source code line to slightly offset the location of where the fixations appear on the source code. Moreover, our method does not preserve the physical properties of the eyes, where eye movements are faster than pupil dilations. Further analysis and alignment is required to reach that goal.

IV. Feasibility study

This section briefly describes the study we conducted to validate the process to visualize eye tracking data linked to the software artifact and demonstrates some preliminary results.

A. Study Design and Execution

The main aim of this study is to investigate the feasibility of visualizing pupil dilations, an indicator of cognitive load, linked with the content of the software artifact that a developer fixated on.

We conducted a lab experiment with 8 subjects that had an academic background in Computer Science or related engineering field. Each subject performed 5 source code comprehension tasks as we recorded eye tracking measurements.

1) Subjects: Eight volunteering subjects were included in the study. The subjects were selected based on their vision condition (i.e., they had normal or corrected to normal vision) and also their knowledge in object-oriented programming languages.

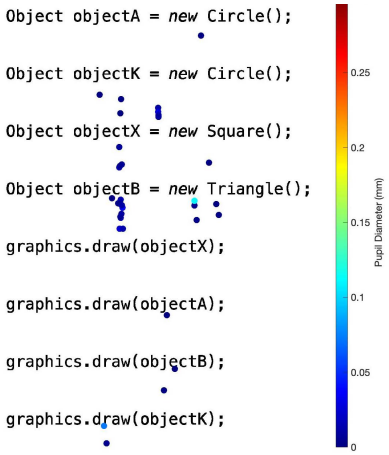

Fig. 3. Graphical representation of the fixations and pupil dilation of one developer (Participant 4) performing an easy task.

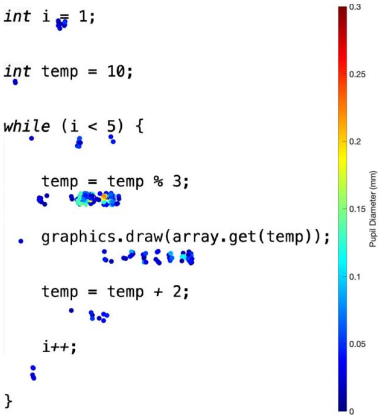

Fig. 4. Graphical representation of the fixations and pupil dilation of one developer (Participant 4) performing a difficult task.

2) Task Description: Subjects were asked to perform code comprehension tasks. These comprehension tasks were taken from [17] and translated into Java. In this study, we selected one type of comprehension task, i.e., code snippets for creating four shape objects (Circles, Squares, Rectangles, and Triangles) and drawing them in some order on the screen. We included five code snippets with varying levels of difficulty (i.e., three easy and two difficult). In particular the easy tasks consisted of a few variable names (mnemonic or generic naming) to impact subjects' working memory by interfering with their ability to remember the mapping between variable name and its shape. One of difficult task consisted of a loop that adds variables of shapes in an array by using a complex mathematical function to impact subjects working memory (for remembering the order of shapes) and their mathematical skills. The other difficult task consisted of double-nested question-mark-colon operator requiring a subject's mathematical and working memory abilities.

During the experiment subjects were instructed to read through the code and identify the last three shapes drawn on the screen (choosing among five possible answers). Syntax highlighting was enabled and there are no comments explaining the functionality of the code. Moreover, subjects never had to write or execute the code and were only allowed to read, scroll and navigate between the tasks. Each of the tasks was designed to take between 2-5 minutes for completion, however, there was no time limit for completing them.

3) Instrumentation: All the tasks were provided to the subjects in the Eclipse IDE and each task was defined as a different project. This was done in order to prevent subjects from taking their eyes away from the monitor. Moreover, for collecting the eye movement data, we used a Tobii 300X eye tracker with sampling rate of 300Hz. Tobii 300X is stationary and attached to the bottom of the screen. The monitor used in the study was 23-inch and had a resolution of 96 dpi (1920x1080). Moreover, the monitor and the eye tracker were placed in a distance of 60cm from the subject. The text font was increased to 14-point size and also we added 2 extra lines between each source code line to reduce the estimation error which occurs with the translation of the detected fixations to line and column.

4) Study Procedure: The study was conducted in a designated room for eye tracking to minimize interruptions and to control the light conditions. When the subject arrived at the designated room, we first asked the subject to fill-in a screen questionnaire to ensure that the subject was eligible for the experiment. The two main criteria for eligibility were that the subject was knowledgable in object-oriented programming languages and that the subject had normal or corrected to normal vision. In the case the subject wore glasses, we checked the glasses if they were in a suitable condition (e.g., unscratched, without bi-focal lenses). If the subject was eligible then the consent form was signed. Afterwards, we provided each subject with an introduction of the study procedure and how to interact with iTrace and how to navigate between the different tasks. Each subject was asked to work on five tasks: one warm-up task and four tasks which were measured (i.e., two easy and two difficult ones). After the completion of all the tasks, we asked the subject to fill-in a post-questionnaire to rank their perceived difficulty for each task. Subjects were allowed to go back and re-familiarize themselves with the tasks.

B. Preliminary Results

Next we applied our method, and Fig. 3 and Fig. 4 show a representative example of the generated visualization for an easy and a difficult task of the same subject.

From Fig.3 we can infer a low number of fixations, and this is expected since this was a fairly simple task. On the other hand, Fig. 4 illustrates the difficult task which is more demanding, which in line with our expectations, shows a high number of fixations.

In addition, in Fig. 3 the pupil dilation values range between 0 to 0.113mm whereas in Fig. 4 the pupil dilation values

range between 0 to 0.259 mm. The task depicted in Fig. 4 was designed to be more challenging and impact the subjects' memory and mathematical skills, which may explain the higher maximum value of pupil dilations.

Similar results were observed in the other subjects. These preliminary results seem promising and imply that the visualization of pupil dilation as an indicator of cognitive load linked on subject's fixations on the software artifact is technically possible and it may help to provide insight into the underlying cognitive processes of a developer.

V. Conclusions and Future work

In this paper, we presented a method we designed and implemented using MATLAB that allows us to investigate the feasibility of visualizing pupil dilations, an indicator of cognitive load, linked with where a developer fixated on a software artifact. Moreover, we presented the results of a preliminary study where subjects read code snippets in Eclipse IDE. The tasks were designed with varying levels of difficulty to impact subjects' working memory and mathematical skills.

We observed that the tasks designed to be difficult can be visually differentiated from the easy tasks. The difficult tasks presented a higher number of fixations and broader range of pupil dilations whereas the easy tasks presented fewer fixations and smaller range of pupil dilations. Our results demonstrate that it may be possible to visualize cognitive processes linked to the content of the software artifact.

In our future work, we plan to create a more automated visualization software and also to extend and improve on the method and visualization to overcome the current shortcomings. One improvement will be to consider the physical property of the eyes, where eye movements are faster than pupil dilations and consider this latency when aligning the pupil dilations and the eye movements. Another improvement, will be to incorporate the temporal aspect to enable visualizing the order that fixations occurred and the fixation durations.

References

[1] R. Guindon and B. Curtis, *Control of cognitive processes during software design: what tools are needed?*, J. O'Hare, Ed. Association for Computing Machinery, 1988, vol. Part F130202.

[2] S. C. Müller and T. Fritz, "Stakeholders' information needs for artifacts and their dependencies in a real world context," *Ieee International Conference on Software Maintenance, Icsm*, pp. 6 676 900, 290–299, 2013.

[3] I. Schröter, J. Krüger, J. Siegmund, and T. Leich, "Comprehending studies on program comprehension," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, May 2017, pp. 308–311.

[4] B. Sharif and J. Maletic, "An eye tracking study on camelcase and under_score identifier styles," 08 2010, pp. 196 – 205.

[5] L. Yenigalla, V. Sinha, B. Sharif, and M. Crosby, "How novices read source code in introductory courses on programming: An eye-tracking experiment," *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9744, pp. 120–131, 2016.

[6] B. Sharif and J. I. Maletic, "An eye tracking study on the effects of layout in understanding the role of design patterns," *Ieee International Conference on Software Maintenance, Icsm*, p. 5609582, 2010.

[7] Z. Sharafi, Y.-G. Guéhéneuc, and Z. Soh, "A systematic literature review on the usage of eye-tracking in software engineering," *Elsevier Journal of Software and Information Technology (IST)*, 07 2015.

[8] B. Sharif, B. Clark, and J. I. Maletic, "Studying developer gaze to empower software engineering research and practice," *Proceedings of the Acm Sigsoft Symposium on the Foundations of Software Engineering*, vol. 13-18-, pp. 940–943, 2016.

[9] J. Beatty and D. Kahneman, "Pupillary changes in two memory tasks," *Psychonomic Science*, vol. 5, no. 10, pp. 371–372, 1966.

[10] D. Kahneman and J. Beatty, *The pupillary system*, J. T. Cacioppo, Ed. Cambridge University Press, 2000.

[11] J. Klingner, B. Tversky, and P. Hanrahan, "Effects of visual and verbal presentation on cognitive load in vigilance, memory, and arithmetic tasks," *Psychophysiology*, vol. 48, no. 3, pp. 323–332, 2011.

[12] M. K. Eckstein, B. Guerra-Carrillo, A. T. Miller Singley, and S. A. Bunge, "Beyond eye gaze: What else can eyetracking reveal about cognition and cognitive development?" *Developmental Cognitive Neuroscience*, vol. 25, pp. 69–91, 2016.

[13] D. Alnæs, M. H. Sneve, T. Espeseth, T. Endestad, S. H. P. van de Pavert, and B. Laeng, "Pupil size signals mental effort deployed during multiple object tracking and predicts brain activity in the dorsal attention network and the locus coeruleus," *Journal of Vision*, vol. 14, no. 4, p. 1, 2014.

[14] D. Kahneman and J. Beatty, "Pupil diameter and load on memory," *Science*, vol. 154, no. 3756, p. 1583, 1966.

[15] B. Wahn, D. P. Ferris, W. D. Hairston, and P. König, "Pupil sizes scale with attentional load and task experience in a multiple object tracking task," *Plos One*, vol. 11, no. 12, p. e0168087, 2016.

[16] M. Köles, "A review of pupillometry for human-computer interaction studies," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 61, no. 4, pp. 320–326, 2017.

[17] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, and M. Züger, "Using psycho-physiological measures to assess task difficulty in software development," *Proceedings - International Conference on Software Engineering*, no. 1, pp. 402–413, 2014.

[18] T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, and T. Ertl, "Visualization of eye tracking data: A taxonomy and survey: Visualization of eye tracking data," *Computer Graphics Forum*, 02 2017.

[19] M. M. Wu and T. Munzner, "SEQIT: Visualizing Sequences of Interest in Eye Tracking Data," *Proc. IEEE Conference on Information Visualization (InfoVis)*, 2015.

[20] N. Peitek, S. Apel, A. Brechmann, C. Parnin, and J. Siegmund, "Codersmuse: Multi-modal data exploration of program-comprehension experiments," *Ieee Int. Conf. Program Comprehension*, vol. 2019-May, pp. 8 813 268, 126–129, 2019.

[21] F. Courtemanche, P. M. Leger, A. Dufresne, M. Fredette, E. Labonte-LeMoyne, and S. Senecal, "Physiological heatmaps: a tool for visualizing users' emotional reactions," *Multimedia Tools and Applications*, vol. 77, no. 9, pp. 1–28, 2017.

[22] T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif, "Itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks," *2015 10th Joint Meeting of the European Software Engineering Conference and the Acm Sigsoft Symposium on the Foundations of Software Engineering, Esec/fse 2015 - Proceedings*, pp. 954–957, 2015.

[23] B. Clark and B. Sharif, "Itracevis: Visualizing eye movement data within eclipse," *Proceedings - 2017 Ieee Working Conference on Software Visualization, Vissoft 2017*, vol. 2017-, pp. 22–32, 2017.

[24] C. Ioannou, I. Nurdiani, A. Burattin, and B. Weber, "Mining reading patterns from eye-tracking data: method and demonstration," *Software and Systems Modeling*, vol. 19, no. 2, pp. 345–369, 2020.

[25] T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif, "Itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks," *2015 10th Joint Meeting of the European Software Engineering Conference and the Acm Sigsoft Symposium on the Foundations of Software Engineering, Esec/fse 2015 - Proceedings*, pp. 954–957, 2015.

[26] M. F. Bear, B. W. Connors, and M. A. Paradiso, *Neuroscience: Exploring the brain: Fourth edition*. Wolters Kluwer Health Adis (ESP), 2015.

[27] R. Dewhurst, J. Weijer, M. Nyström, K. Holmqvist, R. Andersson, and H. Jarodzka, *Eye tracking : a comprehensive guide to methods and measures*. Oxford University Press, 2011.

[28] D. D. Salvucci and J. H. Goldberg, "Identifying fixations and saccades in eye-tracking protocols," *Proceedings of the Eye Tracking Research and Applications Symposium 2000*, pp. 71–78, 2000.

[29] M. E. Kret and E. E. Sjak-Shie, "Preprocessing pupil size data: Guidelines and code," *Behavior Research Methods*, vol. 51, no. 3, pp. 1336–1342, 2019.