Bachelor Thesis Project

Comparison of Two Eye Trackers for the Visualization of Eye Tracking Data in Node-Link Diagrams

Author: Nazli Bilgic
Author: Sofia Kiriaki Vulgari
Supervisor: Prof. Dr. Andreas Kerren
Supervisor: Björn Zimmer
Semester: VT 2016
Subject: Computer Science

Abstract

The usage of eye trackers is becoming more and more popular in the field of information visualization. In this project two eye trackers, The Eye Tribe and Mirametrix S2, are used to obtain eye tracking data for visualizations. It is planned to use the eye trackers with OnGraX, a network visualization system, where they will provide data for the implementation of visualizations, specifically, heatmaps. OnGraX already uses heatmaps to show regions in a network that have been in the viewport of the user. One aim of this thesis will be the comparison between the two eye trackers, and if the use of eye tracking data gives better results than the already existing viewport-based approach. At the same time, we provide the foundation for adaptive visualizations with OnGraX. Our research problem is also of interest for visualization in general, because it will help to improve and develop eye tracking technology in this context. To support the outcome of our implementation, we carried out a user study. As a result, we concluded that one of the two eye trackers appears to have more capabilities than the other, and that using the eye tracking data is a more preferred way of depicting the heatmaps in OnGraX.

Keywords: Eye Tracking, Eye Trackers, Visualization, Heatmaps, OnGraX, The Eye Tribe, Mirametrix S2

Acknowledgments

This project would not have been accomplished without the help of certain people. First of all, we would like to thank our supervisors, Prof. Dr. Andreas Kerren, head of the ISOVIS Group, and Björn Zimmer, Ph.D. Student and staff member in the ISOVIS Group, for motivating us to work hard and guiding us without any discouragement throughout the process of completing our project. Also, for giving us the opportunity to work with them and the chance to learn so many things about our thesis topic, but also other subjects within computer science. We are also grateful to all our family and friends who supported us from the start of our journey until the end and helped with keeping our spirits high. Finally, we would like to extend our thanks to all the people who participated in our user study, that supported us and dedicated some of their time to help us.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

Eye tracking is the process of measuring the point of where someone is looking at. Eye trackers measure fixations of the eye by measuring the movement of an object—like a special contact lens attached to the eye—by optical tracking without direct contact to the eye, and by measuring electric potentials using electrodes placed around the eyes [28]. Eye trackers are applied in various fields like research on the visual system, psychology, psycholinguistics, marketing, or as an input device for human-computer interaction. For collecting data, eye trackers will be used in this project with a new web-based system called OnGraX. It is a system for the visual analysis of large networks, and this system provides an environment for drawing and visualizing graphs as "interactive node-link diagrams" [15]. Also, it provides us with user views/viewports that are areas the users are viewing for a number of seconds seconds [3]. After collecting eye tracking data and user view-based data, heatmaps are used as the visualization technique for visualizing data. Heatmaps are basically the graphical representation of numerical data (Fig. 1.1). They help users to gain a quicker understanding of information by giving them the opportunity to see a large amount of data in a single view.

Heatmaps, OnGraX and the eye trackers are the main components of this project. We implement a software prototype including a user interface in order to control the flow of the eye tracking data between the eye trackers and OnGraX. With the help of the eye trackers, we collect eye gaze data that can be used to better visualize regions or graph objects that a user is focused on. Finally, we compare the final heatmaps that are produced and the heatmap that OnGraX has implemented with the data collected from user views during a graph session.

1.1 Background

Information visualization is a way of reviving abstract data in the human brain, trying to make an internal mental representation of the information. This abstract information can be numerical or non-numerical data, geographic information, or a part of a text. A successful visualization can help to turn a complex story into an understandable view in the mind. On the other hand, a badly designed visualization can both confuse and create misinterpretations [22].

Lately, with the growing evolution of technology, the amount of data that we have to process has increased a lot. Because of this, we are in need of effective methods that can help us to handle this enormous amount of data that we face in our everyday lives. Making visual representations of data is one of the most effective methods that can solve this problem. We should not forget that the human brain is, in many ways, like a machine, continously trying to handle and make use of a gigantic amount of information at the same time. The visualization of a large amount of data makes it easier to understand complicated systems and gives us a great opportunity to make better and more reasonable decisions. The usage of color, shapes, movements, graphs, and so on, gives people the possibility to explain a lot more with less material [16].

Eye tracking is a way of investigating what people look at, how their eyes move, and how their visual activities behave. It is basically a way of observing what people do with their eyes. The devices that measure these eye activities are called *eye trackers*. With the results that the eye trackers provide, researchers can examine what our brains choose to focus on. One of the most common visualization technique that is used by researchers for eye tracking studies is a heatmap. It is a very common and popular method, because it shows the viewing behavior of individual users. By using color coding (see Fig. 1.1), it provides information about gaze points of users or duration of fixations [18]. Heatmaps are an arrangement of cells, and each of these cells is colored by a value of data. Heatmaps help us to get a quick idea about the data distribution over an object, observed during an experiment [17].

Due to the large number of scientific fields where eye tracking is being used, different approaches have been developed for analyzing eye tracking data. Where some of them mainly provide quantitative results, new techniques allow the usage of different levels and aspects of the recorded eye tracking data in visualization systems. The first eye trackers were created in the late 1800s, and in the mid 1970s several companies started producing and then distributing them to researchers and others that found interest in their use. There are applications that provide users with an opportunity to observe what other users are changing and editing, when working synchronously or asynchronously on the same document. The network visualization application that we use in our thesis project is OnGraX [15], which is a web-based visualization system [2, 3, 15] that supports synchronous, asynchronous and distributed analysis of graphs. The graphs display a network of nodes and edges, in which a node "is usually represented by a circle with a label" [34]. An edge "is represented by a line or arrow extending from one node to another" [34]. A common example of graph visualizations is the visualization of family trees. A network between family members where each node represents a member, and in the case of two nodes being related, an edge is added between them.

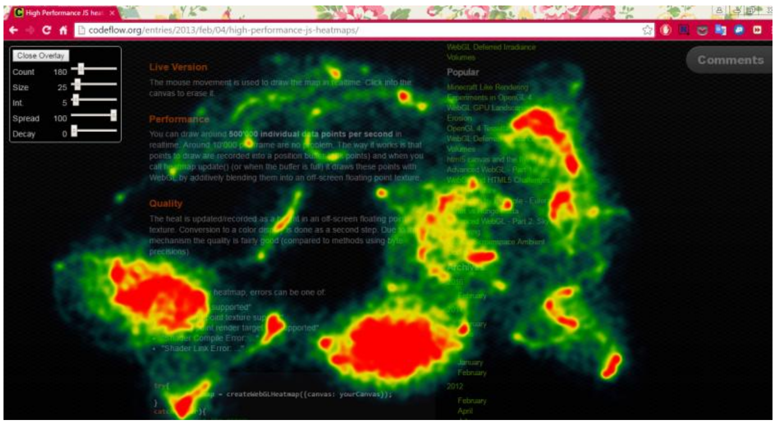


Fig. 1.1 Commonly used heatmap, marking red the regions of interest. Screenshot taken from [32].

1.2 Previous Research

Quite a lot of publications focus on visualization and eye tracking. Some of them are more specific and some look into other aspects of eye tracking.

Steichen et al. [4] provide adaptive information visualizations by the usage of eye tracking, but also with the help of machine learning to foresee and adjust to the user, task and visualization characteristics. Ahn and Brusilovsky [7] discuss how they want to develop systems that provide helpful personalization for search approaches in search systems. This work also focuses on making user interactions that raise the chance of more reliable user modeling. They also present an adaptive visualization system called Adaptive VIBE [7].

Steichen et al. [5] claim that making user-adaptive studies is better to improve the visualization performance. We can see, from several different kinds of research, that eye tracking measurements can be a very important source for studies of visualization systems [5, 6]. With the measurement of the results that they got from the tracker, they can see that eye gaze is a very valuable source to gain information about the characteristics of users and some visualization tasks [5]. In a similar research by Tomasz and Peter [6], eye tracking is used in an experiment to examine how students separate their attention and how eye movements are effected from different adaptations.

Ho et al. [8] investigate the likelihood of applying eye tracking analysis to assess flow visualization methods and examine how visualization methods impact on visual behaviors and how different kinds of flow visualization methods influence the user's performance in doing different tasks with the help of some experiments.

Research by Conatiet al. [9] is based on a user study where they try to gather information from different user characteristics, and design user-adaptive visualization tools which can adapt to real time needs. This characteristic data can affect a user’s visualization experience. Different types of methods are used for this study. One of them is eye tracking. They try to find out if characteristics of users can change the behavior of the gaze during the tasks and which features of the gaze mostly got affected by the user characteristics.

Toker et al. [10] present a statistical analysis with Mixed Models to study “how a user’s gaze behavior relates to user characteristics, task difficulty, and visualization type” [10]. At the same time, they introduce a new definition of task difficulty, taken from applying Principal Component Analysis to some objective and subjective performance measures. Also the results from all these analyses show that the user characteristics are affecting the user gaze behavior which is detectable from a lot of eye tracking metrics.

A publication of Kaufman et al. [11] talks about a user interface controlled by eye movement for 2D and 3D interaction, based on electro-oculography. “They built hardware and software to show the viability of electro-oculography for human-computer communication” [11]. In addition, they conducted experiments that show that it is very capable for virtual reality systems, games and also for handicapped people.

Steichen et al.’s research [12] is about using gaze data in real-time systems. They explain another architecture, which has more benefits than other methods used for eye gaze data analysis. This system is used for developing visualizations by using real-time gaze data. One of the advantages of their system is that it contains a web-service which takes a continuous stream of raw gaze data from an eye tracker, and another benefit is that it combines a wide range of data statistics.

The paper of Kelley is about THOR (“Tool for High-resolution Observation Review” [13]), and this tool gives an opportunity for the users to convert a desktop application to an application that can run in web browsers. With this process, users can visualize large data with a few mouse clicks. Being so simple, it attracts many researchers that want to adapt their visualization applications and deploy them online.

To avoid the problems that they had about eye tracking in other studies, Moro et al. [14] come up with a substructure for gaze tracking, focusing on dynamic web applications. With this research they introduced an infrastructure which can do an automatic assessment of the eye gaze data from multiple users and eye trackers, and it also provides visualization support for dynamic fields of interest.

All papers previously mentioned research either in visualization or eye tracking, but they are not exactly in the focus of our work. We provide the basics for adaptive visualizations (c.f. Section 1.3) by using eye tracking data and user views on OnGraX. Despite that, they are still a very good and helpful resource for our research. Also, since they are different, they motivate us to successfully complete our research. So, we can provide substantial results for other researchers and users in the future.

1.3 Problem Formulation

The use of eye tracking devices is becoming more and more popular in the research field of information visualization. However, there are differences in the way of how and why they are used. Mostly, they are used for user studies in order to see which elements of the visualization are important for the user in order to solve specific analysis tasks. Very rarely, they are used to influence the visualization (adaptive visualization), to change a visual representation (for instance, providing semantic zooming if a graph object, like a node or an edge, is in the user's focus) [2, 3], which is why we are offering the groundwork for it. Our main aim is to also discover if the heatmaps created with the help of the eye trackers and our software prototype are showing more accurately the user's focus in the graphs, than the view based heatmaps that are currently offered in OnGraX. Also, in our research, we include a comparison of the two eye trackers based on our experience with them, in order to see which is better and easier to work with.

1.4 Motivation

Our research problem is of interest to the field of adaptive visualizations, because it will help in the improvement and the development of the technology. Major problems here are the understanding of the data, and that typical visualization tools are not user adaptive. The overall aim is to develop better visualization tools, make them more efficient, effective, precise and able to provide better analysis results [2]. Also, as discussed before, to give the foundation for adaptive visualization, to be able to change the visualization of a graph object, for instance, to make the characteristics of a node appear around it, as the user is looking at it.

1.5 Research Questions

RQ1	Which of the two different eye trackers is more effective in our research context based on our experience with them and on which ways?
RQ2	Which are, if any, the technical and/or conceptual limitations on the use of the eye trackers that are faced during our research?

RQ3	How can the heatmap quality in OnGraX be improved by the use of the eye trackers?
RQ4	How can the results be integrated into the OnGraX system application scenarios?

RQ1: With this question, we want to compare the two trackers in terms of usability, quality and effectiveness of the APIs. By effectiveness, we mean how easy it is to complete the calibration, which API is more adaptable and can be incorporated in our code, and also if the eye tracking data from one device is more stable and accurate than the other.

RQ2: By including this research question we want to find out about any problems or limitations that we personally see with the use of the eye trackers (in relation to our research), to be able to document them and inform future researchers and users.

RQ3: With the third research question the goal is to see if the quality and accuracy of the heatmap by the viewports of the user, is improved or not with the use of the eye trackers.

RQ4: The fourth question is about how we can take the results of our work and apply them in the OnGraX system. After answering this question, we should have a visualization of the result.

1.6 Scope/Limitation

One limitation from the start of our thesis is that it is not possible to test and compare all eye tracking devices that exist. For that reason, we use two of them, one affordable and economical choice and one that is much more expensive, to discover the differences between them.

The next limitation is that the work is done for a specific system, OnGraX, so it cannot be tested directly on other tools without further development and adjustment.

For our user study, we brought in people from different kinds of study fields, not only computer science, but we did not include people from an older age range or scientists related to our subject. We were able to take the experiment data needed and compare them, but we were not able to test our work with senior scientists or researchers that might have used the project in their line of work.

1.7 Target Group

The target group interested in this thesis would be scientists, domain experts, academic staff, teachers and researchers who would like to use the tool for their own studies and developers who are interested in the same field of eye tracking and visualization who would like to take some ideas for their own work or research.

1.8 Outline

The report is structured in 7 chapters. Chapter 1 contains the background, the description of our subject, the problem, previous research and our motivation. The next chapter is about the methodology that we use in order to complete our work, while the 3rd chapter introduces the two eye trackers that are used in our study. Moving on to Chapter 4, we describe the implementation of our project while also including the limitations of the two eye trackers, based on our experience with them. Lastly, follows the description of the user study, the results (raw data in Appendix 1) and the analysis of the created heatmaps. Chapters 6 and 7 contain the discussion, the conclusion of our results and future research.

2 Method

In our research work, we are mostly going to use an incremental research strategy, since we will improve our implementation stepwise based on observations we have made. We are then going to collect the data which we will get from both eye trackers and use it to prove some questions like, which one of the eye tracker is better and what are their limitations. Also, to find out more about our research questions, we will design experiments/tests conducted in a controlled environment. In that user study, a number of users will be asked for their experience with the trackers and the developed visualizations.

2.1 Scientific Approach

Our research will be inductive since we will observe and collect knowledge from the implementation and tests that will be done [23]. We will use both quantitative and qualitative methods in our research.

- To answer the first research question, we will use a *qualitative* method as we will describe, according to our experience, in a free text which one of the two eye tracking devices is more effective for our purpose.

- The second research question, just like the first will be written in a free text according to our experience with the trackers, and we will refer and explain the limitations and problems we had with the use of the eye trackers. So, the research method for answering this question will also be *qualitative*.

- The third research question is about implementing and then testing the results. More specifically about comparing the experimental heatmaps from the eye trackers with the heatmaps from OnGraX, all produced from the experiments with the test subjects. After we obtain the results, we compare them to see which is better. After the users finish their task and complete the test, we will give them a questionnaire with six Likert scale questions and one question to answer with a free text. So finally for this question, the research methods will be both *qualitative* and *quantitative*.

- To answer the last research question we will use *qualitative* methods and discuss how visualizations can be adapted with the help of eye trackers.

2.2 Method Description

This section is on describing the methodology of the project.

2.2.1. Software Development Methodology

Building a software product is a hard and complicated process which consists of several different stages that are: research, planning, design, development, testing, setup and maintenance. Rules and guidelines that are used during these steps are called software development methodologies.

The main idea behind the method that we will use in our system is to develop the system in repeating cycles and small steps. Design, test and implementation of this project will be done incrementally. This means that progress will be gained by adding small things each time to the project. The project will be separated into different components, and the purpose of those will be to add more functionality to it. By adding all these separated components, the project will grow incrementally. One of the most important phases in our project method is that after the implementation of each different component, feedback will be taken from our supervisors. After taking the feedback and everything will be how they want it to be, we will continue to the next component. This process will be repeated in every single part of the project.

In the first step of the project, we will do the definition of the system so general knowledge about the topic is very important here. We will also put boundaries to the project and do the definition of the risks. Doing the definition of the risks will help us to take precautions previously. This will allow us to progress easily in the next steps [19]. Before we will start the project, we will read related papers to our research to improve our general knowledge in the field. Also, we will have meetings with our supervisors to have a general view about the project and discuss every possible fact that can happen during the implementation part. Generally, how eye trackers work, what is OnGraX and its working logic and communication basics between client, server, webpage, MySQL database are discussed during this step.

In the second step, we will form the skeleton of the project. We will try to understand the system better and in a more detailed way. Baseline architecture and elimination of critical risks and risk elements will be handled in this phase [20]. To avoid having problems during the implementation part, we will create a plan which shows the basic steps of the project. In this step we will have a more detailed information about how the eye trackers will be connected to the client, understanding the API’s of the trackers in detail, how OnGraX will work with the data that will be collected, how a live heatmap from tracking data will be created, storage steps for the data and how the visualizations will be used in the project.

Thirdly, we will do the implementation of iterations according to the requirements that are defined in previous steps. Also, we will do the integration of these iterations to the project. In this step, we will start to implement everything that we planned and discussed with our supervisors. It is the phase where all the connections from the eye tracker API's will be adapted to the system to be able to get the tracking data. For the transmission of data connections between eye tracking client, server, MySQL database, and the OnGraX main client will be created. On the OnGraX main client side of the project, we will create a live heatmap according to the tracker data and also change the color of the focused graph objects. For this project these graph objects will be nodes. The recording process will be implemented on the server side. Also, we will do the design and implementation of the eye tracking and login GUI windows in this step.

In the final step, transition of the finished implementation will be done to the end user. All the previous planned objectives will be met at the end of this section. For this project, the user study that we will do at the end is related to this phase. With the user study, we will have the opportunity to evaluate the finished project. From the results that we will gain, we will have a chance to see the differences between the two eye trackers and the limitations that we will face with them. In order to see if the heatmap quality in OnGraX can be improved by the use of the eye trackers, user heatmaps will be examined. In this step, in addition to the simple visualization adaptation that we will do with the focused graph objects, we will see how the results can be integrated into the OnGraX system application scenarios.

2.2.2 Brief Explanation of the User Study Structure

At the end of our project, we conduct a user study between two groups. Every group consists of four people. To be able to have better results in our study, we did not determine a specific subject of study for our participants. A task is determined for the study, and it is applied by every person that attends the study. Basically, in the task we ask people to go from Barcelona to Berlin in a Europe shaped graph which is visualized with OnGraX. This map occurs from nodes that are connected to each other. Every node represents a city, and the edges represent euroroads ("a numbering system for roads in Europe developed by the United Nations Economic Commission for Europe (UNECE). The network is numbered from E 1 up and its roads cross national borders. It also reaches Central Asian countries like Kyrgyzstan, since they are members of the UNECE [29]) between the cities. Every user logs into the system with their username and password and they start doing the task. The first group does the task with the Mirametrix S2 eye tracker and the second group with the Eye Tribe tracker. All the data is saved to the database according to the user id. After every participant finishes the task, they fill out eight questions. Seven of these questions are in Likert scale, and the last question is an open question where users write their own opinion about the experiment. While users do the experiment time is measured for everyone. Also, we measure the time while the users answer the questions after the task.

2.3 Reliability and Validity

On the subject of *reliability* and our method of data collection, we believe that it is quite adequate since we tried to consider everything. We recorded eight, four for each tracking device and for OnGraX, user tests on camera, took notes while the users were completing the task about the path they took, how long it took them to finish (the database also has a timestamp that measures time) and how long it took them to answer all the questions of the questionnaire. The participants did not have to stay for a long time doing the test so they were not affected by hunger or fatigue [26].

If other researchers redo our research, they might not get the same data since they will probably use different users, but the end result about the eye tracking heatmap being better or worse than OnGraX's heat map should be the same. For some of the implementation parts, developers can of course choose a couple of different ways to implement some parts of the coding, but the final result would be the same.

OnGraX has not been tested before with eye trackers for the subject of this project or similar.

Construct validity should not be a big issue, as all basic terms and processes are explained.

Internal validity was also avoided by taking into consideration all variables that might affect our research, like reading all the guidelines for the correct and uneventful use of the two eye trackers. Also, in our user study that we carried out after our implementation, we invited people from different nationalities and gender, different levels of education and different subject of studies. Some of them were acquaintances while others complete strangers, so, in the end, their answers were not biased.

The *external validity* is believed to be avoided too, as we are including not only students in our user study, but also job holders. Also, regarding the implementation, the project can run on any operating system that can support Eclipse IDE and a web browser [27].

2.4 Ethical Considerations

In our user study (Section 5), we used a questionnaire to collect information about the eye trackers. For that reason, we gathered some people/users to help us compare and rate the two eye trackers with each other and with the screen views of OnGraX. In the questionnaire given, their names, age, gender and the subject of their studies are asked to be given. We will mention the last three parts of that information, leaving out their names for anonymity reasons.

3 Eye Trackers

For our study, we used two different eye trackers. The first is from a company called The Eye Tribe, and the other one from the Mirametrix. The Eye Tribe is one of the most affordable eye trackers in the field (about 99 USD), while the Mirametrix S2 eye tracker is a more expensive choice (about 5.000 CAD ≈ 4.000 USD, with conversion rate 1 CAD = 0.788198 USD).

3.1 The Eye Tribe

The Eye Tribe, as a company, has the goal to provide inexpensive eye tracking for all. Very fast they got distinguished as one of the world leaders in economical eye tracking [33]. The Eye Tribe software allows eye control, permitting hands free usage of websites and applications.

The tracking system has to be positioned below the screen that will be used and the user within the tracker’s Trackbox (the volume in space where the user can be found by the tracker), so that it can track the eye movement and calculate the coordinates on the screen. The SDK (software development kit) of the tracker has a Trackbox sample that demonstrates how the users should be positioned [21].

3.1.1 Software

The software consists of two parts. First, the Eye Tribe UI, which needs the second one, the Eye Tribe server, to be running. If the server is not running first, then the UI (user interface) will try to start it automatically. In the Eye Tribe UI it is possible to see information about the calibration and the state of the eye tracking. It also provides the possibility to change the settings according to the user’s needs. In the UI, we can also see the Trackbox window, so it informs the user if the position is correct and if the tracking is working or not. In the image below, we can see the different tracking cases (see Fig. 3.1) [21].

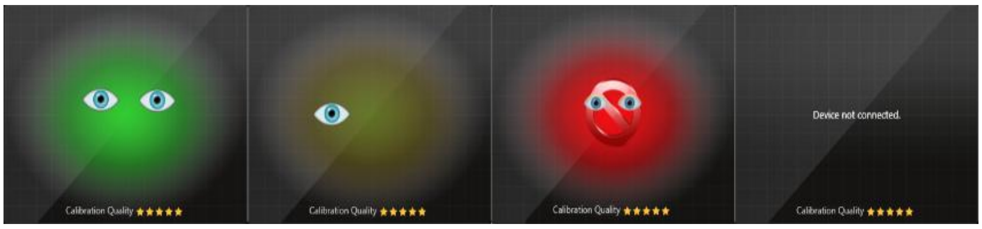

Calibration Quality ★★★★★
Calibration Quality ★★★★★
Calibration Quality ★★★★★
Device not connected.
Calibration Quality ★★★★★

Fig. 3.1 Cases of good, medium, bad or non-tracking as shown in the trackbox window.

3.1.2 Calibration

During the calibration of the system, the software calculates the gaze coordinates of the user. The average accuracy is around 0.5 to 1^o of the visual angle. If the user is around 60 cm in front of the tracker and the screen, the accuracy has an average error of 0.5 to 1 cm.

Before the use of the tracker, each user has to undergo a calibration process so that the software can model the different eye characteristics and finally be able to estimate the gaze accurately. The API (application program interface) of the calibration is displayed as a black window with circular targets that are illustrated in nine different places on the screen. Nine targets are recommended from Eye Tribe, but there is the option for twelve or sixteen target locations that will make the tracking results more accurate. The users have to look at all the targets one by one when they are being displayed so the calibration can be completed. If the tracker is moved after the calibration, then the user has to calibrate again: in this way the Eye Tribe system can update the new parameters [21]. The calibration screen with 9 target points can be seen in Figure 3.2 below.

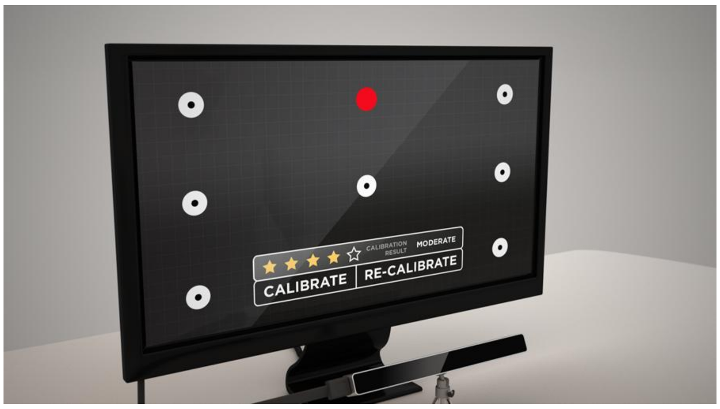


Fig. 3.2 Eye Tribe Calibration screen.

3.2 Mirametrix S2

The Mirametrix S2 eye tracker adapts to the Open Eye-Gaze Interface [24], which supports a standardized design and procedure for calibrating and obtaining the eye-gaze information, allowing developers to use eye gaze in their applications. In comparison to the previously discussed eye tracker, the Mirametrix is almost 40 times more expensive.

3.2.1. Calibration

The Mirametrix S2 tracker functions as a server. During calibration (see Fig. 3.3), the server will send the results to the client for each point. The communication is achieved through a TCP/IP connection and provides XML data output [24]. The tracker should be located beneath the screen of the computer that is being used, or in front if it is a laptop. For better results and performance, the tracker should be placed around 65 cm from the user, and it should be used far away from windows and infrared light [24].

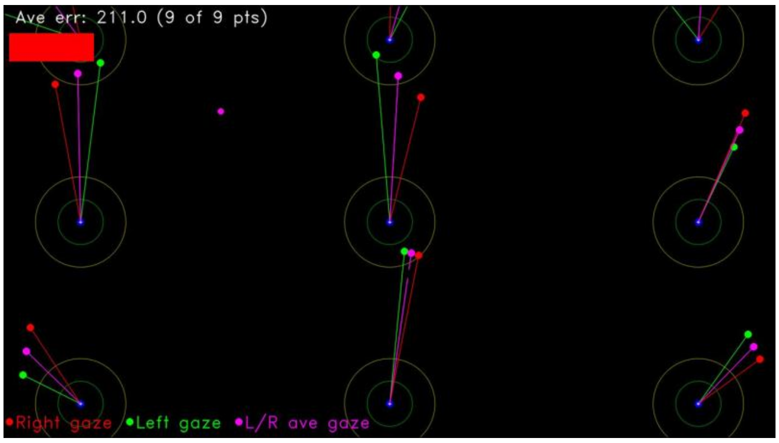

Ave err: 211.0 (9 of 9 pts)
•Right gaze •Left gaze •L/R ave gaze

Fig. 3.3 Results from a calibration with the Mirametrix eye tracker.

3.2.2. Software

The software consists of two parts. The Tracker window and the Viewer window. The Tracker window shows the real-time image of what the device is recording from its camera, while also identifying the left and right eyes of the user (see Fig. 3.4). If the tracker operates properly, the left eye is bordered with a green rectangle, the right with red and both of them have a little cross in the center of the pupil [25].

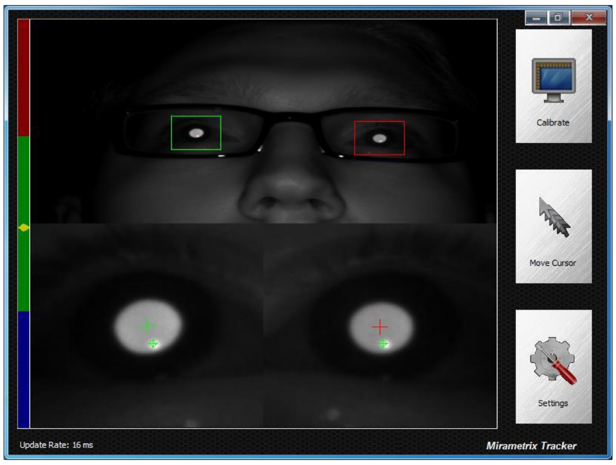
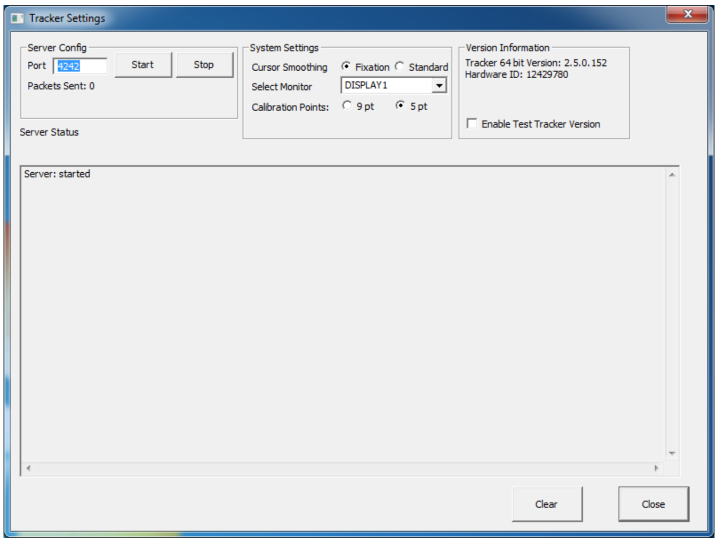


Fig. 3.4 Mirametrix S2 User Interface.

The bar on the left is estimating the depth that indicates when the head position of the user is in the perfect place, which is at the green space in the center of the bar.

Also, there are three buttons on the right of the window. The *Calibrate* button on the top begins the calibration process which is necessary once for every user or if the tracker is moved from its position.

The *Move Cursor* button can be used when the user wants to connect the mouse cursor with the user's gaze on the screen so the user can have a visual feedback [25].


Tracker Settings

Server Config
Port 4242 Start Stop
Packets Sent: 0

System Settings
Cursor Smoothing [✓ Fixation] Standard
Select Monitor DISPLAY1
Calibration Points: 9 pt [✓ 5 pt]

Version Information
Tracker 64 bit Version: 2.5.0.152
Hardware ID: 12429780
Enable Test Tracker Version

Server Status

```
Server: started
```

Clear Close

Fig. 3.5 Tracker Settings window for the Mirametrix S2 eye tracker.

The *Settings* button opens up the Tracker Settings window (see Fig. 3.5). Here, there is the Server Configuration section, where we can see the Port box with the default server port that can also be changed by the user. Next to the Port box there are the Start and Stop buttons that turn on and off the TCP/IP server.

Next, are the Gaze Settings that contain the Cursor Smoothing, which can be set to Fixation (detects fixations when moving the mouse, and results in a faster response) or Standard (smoother response but slower) [25]. Also, there are the *Select Monitor* and *Calibration Ports* where the user can choose the calibration points (the standard is nine, but there is also the option for five, although it results in loss of accuracy).

Lastly, the *Version Information* section shows the tracker's *Hardware ID* and *Version.*

4 Implementation

This part of our thesis explains the implementation process.

4.1 General Explanation of the System Architecture

The main components of our system are the two eye trackers, the eye tracking client, the Tomcat server, the OnGraX main client, and the MySQL database (see Fig. 4.1). To be able to obtain data from the trackers, their APIs were adapted to the system (see Section 4.2). This adapting process is done on the eye tracking client side of the project, while Java is used for the eye tracking client and the tomcat server. The OnGraX main client uses WebGL, JavaScript and HTML5 [15]. Additionally, the visualization of graphs is done in WebGL which "is a JavaScript API for rendering interactive 3D computer graphics and 2D graphics within any compatible web browser without the use of plug-ins" [36]. Our eye tracking client is used to control the eye trackers and to send the eye tracking data to the Tomcat server.

The server side of the project is implemented in Java and runs on an Apache Tomcat server. The connections between eye tracking client, server and OnGraX' main client are provided through Web-Socket ("a protocol providing full-duplex communication channels over a single TCP connection") [35] endpoints. Through the endpoints, messages are sent as JSON-strings, and each message contains an integer value to identify the type of action that was performed. The messages give information about the operation that the users select to do, and specific data can be associated with the action which is used for specific events. For example, in order to send the tracker data, the "tracking data" action is used with an account name, a session id, and x and y coordinates that represent one tracked point of the user's eyes. For the login functionality, the login action is used with the username and password attributes. The login functionality and sending the session id with every single message are important, as OnGraX can be freely accessed via the Internet. Thus, we had to implement a login procedure for our eye tracking client, to be able to connect to OnGraX and send the tracking data to the correct user and graph session. User id, x/y tracking coordinates and time data are stored in tables in a MySQL database, once the user starts recording the data. When the users want to record the data during the session, a table is created with the graph name that is open during the session, if it does not yet exist, and the data is saved there.

User id, x/y tracking coordinates and time data are stored in tables in a MySQL database, once the user starts recording the data. When the users want to record the data during the session, a table is created with the graph name that is open during the session; if it does not yet exist, and the data is saved there.

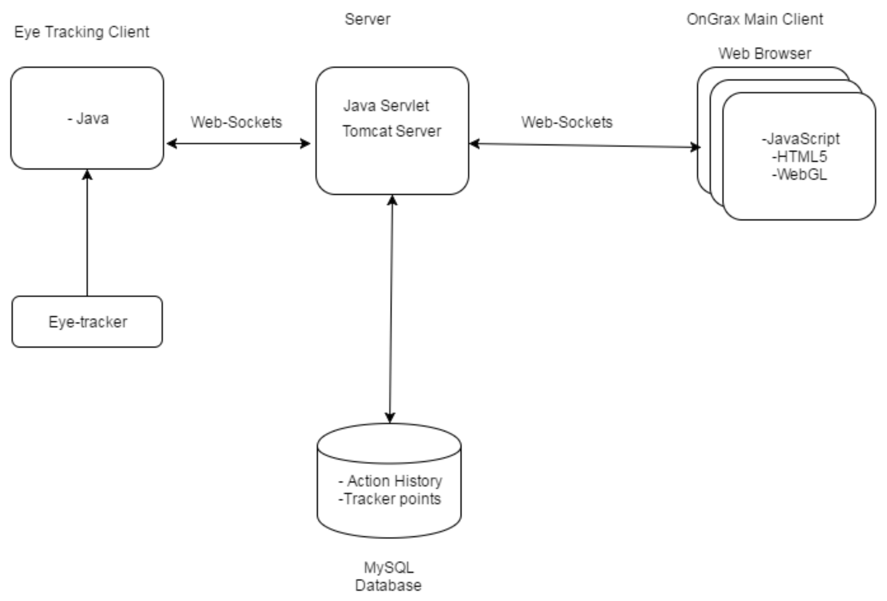


Fig. 4.1 General architecture of the system.

4.2 Adapting the APIs of the Eye Trackers

To be able to obtain data from the eye trackers, we had to access their APIs in our Java implementation of the eye tracking client. In the current implementation, the Java class OnGraxEyeTrackingClient.java has been implemented and includes both communication protocols of The Eye Tribe tracker and the Mirametrix S2 eye tracker.

4.2.1 The Eye Tribe Tracker

The Eye Tribe tracker provides an easy API for programmers, so they are able to get the data from the device. In order to use this API, the tracker device has to be connected with the computer. It has to be correctly positioned where it can observe the user’s eyes in a good perspective and obtain a good quality calibration. Of course, the tracker should be running too. A TCP connection is required between the OnGraxEyeTrackingClient.java and the localhost, on port 6555. The message system for the API of this tracker is in JSON format, and it provides the gaze data in pixels. One very important issue here is that gaze data is only available in a calibrated and ready system.

One of the steps to start building the communication with The Eye Tribe tracker is to download TET Java SDK [21] into the Java project. To be able to make the connection with The Eye Tribe tracker, the "GazeManager" needs to be activated in the main class. During this process The Eye Tribe server has to be running, otherwise, the system will not continue the process. Then, the next step is to receive gaze data from the device. To do that, "IGazeListener" and "GazeManager" have to be implemented [21]. Gaze data contain the raw coordinates and smoothed coordinates. Raw coordinates are the coordinates that the tracker sends directly without any process, while the user is looking somewhere on the computer screen. Most of the time, this data can be wrong. This can happen because of some calibration errors, blinking, or the tracker misses a frame during a session, etc. Smoothed coordinates are the average of the eye gaze data. These values will be filled once the calibration process is over. The 2D coordinates are taken from the gaze manager as raw coordinates, and then they are ready to be called by the "send tracker data" method and sent to the OnGraX server.

4.2.2 Mirametrix S2 Eye Tracker

The Mirametrix S2 eye tracker provides developers the Open Eye-Gaze Interface with a simple method for taking the data and starting the calibration process. Communication between this tracker and the computer is provided by a TCP/IP connection via passing XML strings. It uses 4242 as the port for supplying the connection. For this device, the eye tracker runs as the server. This server replies to all the statements that are coming from the client, which is connected to it. The tracker provides all points in percentages of the tracking window, which means that the screen size is needed in order to convert them into pixels. The OnGraxEyeTrackingClient.java class includes a method called "getMiraMetrixData" where implementations for the Mirametrix S2 tracker are done. This method also contains the socket connection, readers for the XML string data and the queries for providing the tracking data. To send the best point of the tracker's gaze data, the "enable_send_pog_best" variable is used [24]. This variable gives the average of the left and right point of the gaze data estimates. For this estimation, both left and right eye data have to be valid. If the data from one of the eyes is not received, then only the point of the other one is used. The gaze point is converted from string to double after the message from the tracker is received and the points are converted to screen size points. The percentage data obtained for the x and y coordinates are multiplied by the width and height of the screen. At the end of these changes, the tracking data is ready to be used in our system.

4.3 Transmission and Storage of Eye Tracking Data

In this section, the transmission and storage of the data will be explained in detail. To be able to apply the data to the visualization, the eye tracking data, which was obtained from the eye trackers, should be sent to the server and from there it should be sent to the "OnGraX.Web.Client".

4.3.1 Login Process

The login process is a very important part of this project. To be able to connect the eye tracking client with the OnGraX web page, the user has to login from both sides with the same username and password, and to make sure that he/she entered the correct username and password to the system. If they do not enter the same login information, errors will occur during the transmission and storage of the tracking data. For this login operation in OnGraxEyeTrackingClient.java class, a "login" method is created. This method contains the appropriate action with the related data that are the name and password of the users. These data are sent to the server through Web-Sockets with JSON strings and there, these strings are received in the related methods. The name and password that are sent to the server are the information that users enter in the fields of the login window of the eye tracking client. After the server receives the data of the login, the system will check if it is a match with the data from the database and on the OnGraX web page. If it is not, the user will not be allowed to use the other functionalities of the system. If all the data is correct, the user will be able to open the client window which contains the start/stop sending data buttons, the start/stop recording data buttons and the slider. After the connection between this window and the web page is approved, the users can enter into a graph session, record their behavior during the session, send data, stop recording etc.

4.3.2 Storing the Eye Tracker Data

The communication between trackers and the eye tracking client is provided with the help of the tracker API, and the eye tracking points can be displayed.

Start/stop sending data, start/stop recording and login processes are implemented on the eye tracking client side of the project. Methods that are responsible for these transactions send the proper data to the server through Web-Sockets with JSON strings. These strings contain actions and connect the data to that operation. For the start sending data method, start sending the tracking data action is used with the account name and session id. To be able to transmit a lot of tracking points, a Java thread is created. With the use of this thread and the tracking data action, the data are sent to the server. A tracking data action contains the account name, the user session id, x, y, and the slider information data. A slider is used to show the last x seconds of the tracker data and to calculate which node is in the focus of the user (see Section 4.4 for details). On the server side of the project, in

EyeTrackerActionHandler.java class, cases are created for all actions. Inside these cases, appropriate processes are done to be able to complete the communication process. Data from the client come to the “receive tracking data” method on the server.

In order to store the eye tracking information that arrives at the server, the MyBuffer.java class is implemented in the server. The main idea of creating this class is to be able to store the tracking data that is received from the eye tracking client (see Fig. 4.2). From this class, “MyBuffer” object is used in the methods that are connected with the eye tracking client. This class has connections with the “MyPoint” and “EyeTrackingActionHandler” classes in the project. Inside the MyBuffer.java class, we insert all points that are coming from the eye trackers. The points that are received from the trackers are generated by the current user’s camera position in the graph view. The storage of the points should be in world coordinates, so whenever a point is received from the eye trackers, it has to be mapped from screen coordinates to world coordinates, and then saved temporarily in the buffer array list (see Fig. 4.3). Figure 4.2 shows from where the screen coordinates and the world coordinates are starting.

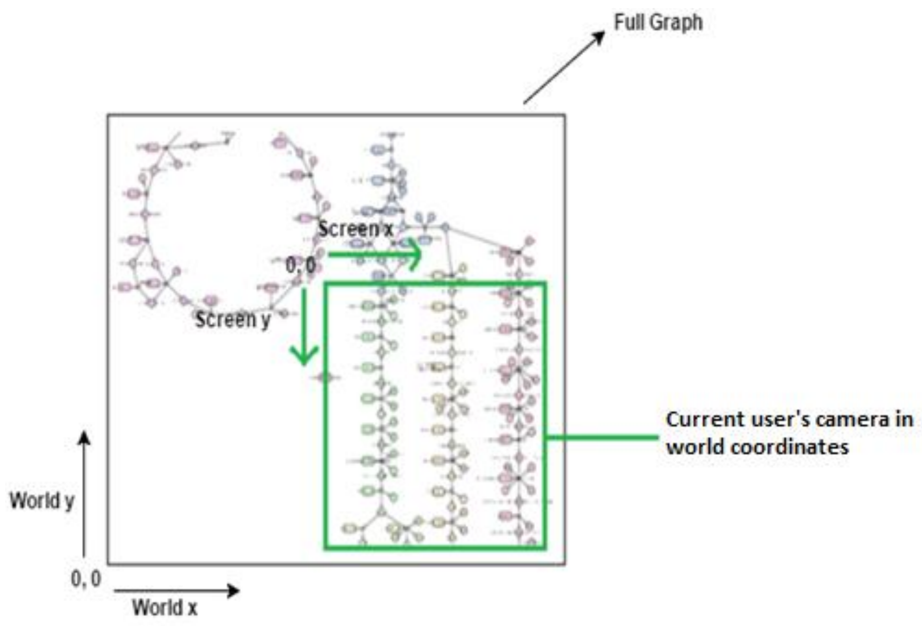


Fig. 4.2 Screen points and World points.

Another step for the data transmission is to transfer the data from the server to the web content of the project. The communication between server and OnGraX web client is also provided by Web-Socket endpoints. In the web content of the project, in EyeTrackingHandler.js, we use the tracking data, start receiving data and stop receiving data functions.

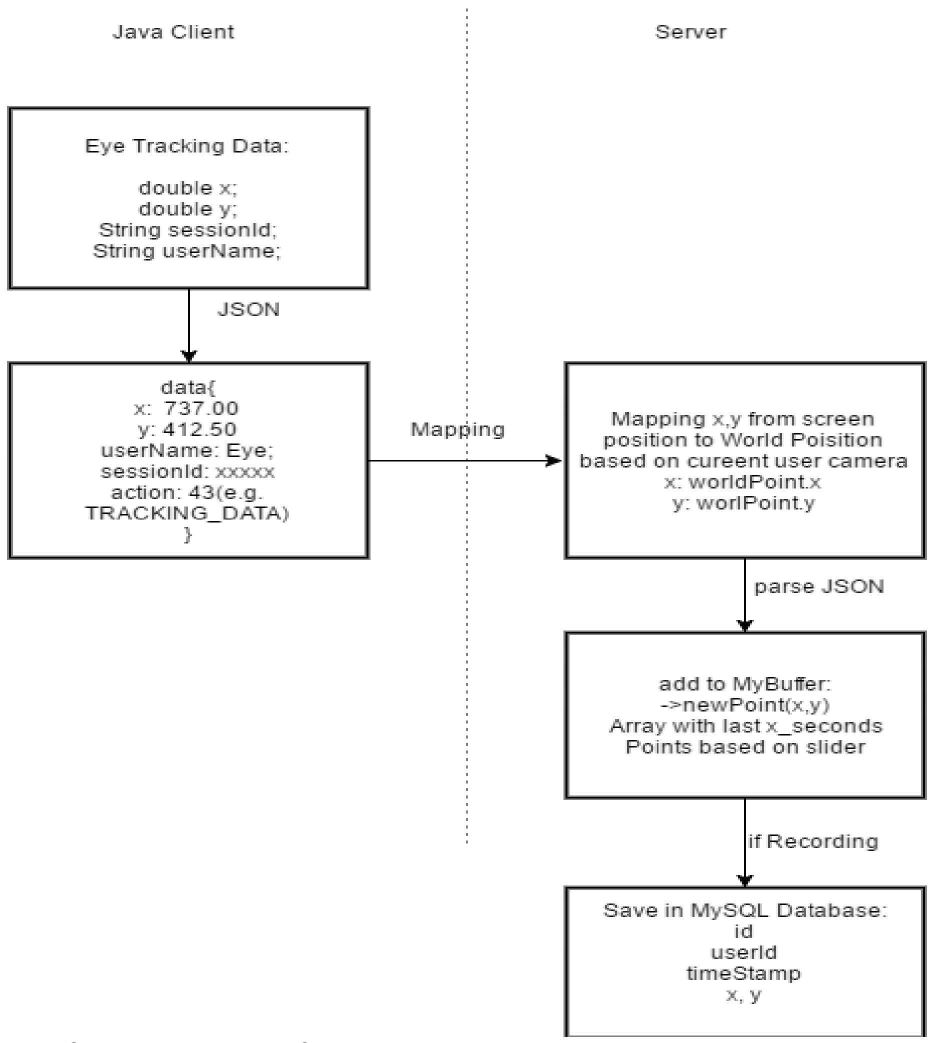


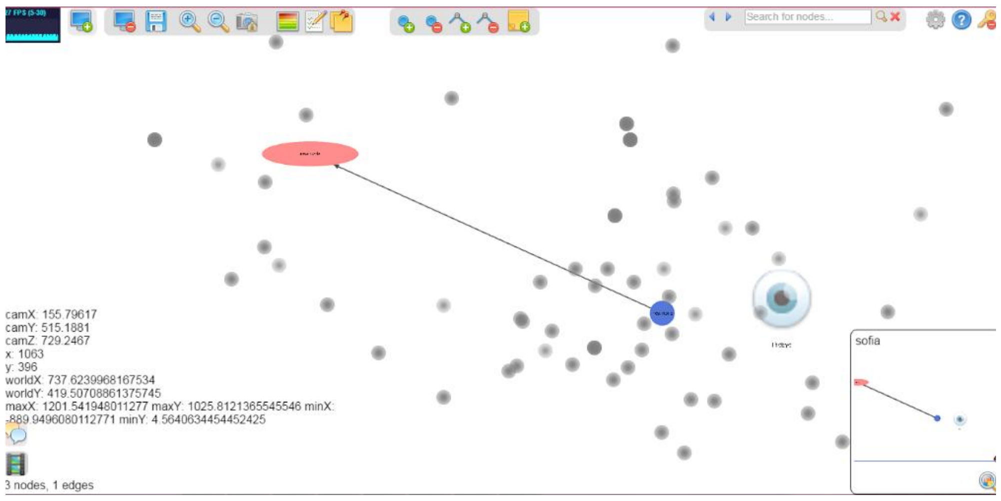
Fig. 4.3 Steps for storage processes of the system.

4.3.3 Identifying Focused Graph Objects Based on Tracker Data

Also, inside the myBuffer.java class, another storage operation is made for the identification of graph objects. For our prototype implementation, we focus on identifying the nodes in a graph [34]. It would also be possible to extend this functionality to the edges. To be able to identify graph objects that are in the user's focus, the tracker points from the buffer are used. In this process, those points that are inside the nodes are stored in an array list, and the node with the most points changes its color to blue. To be able to calculate which points are inside the node, width and height of the nodes are taken from the system. To do a correct calculation, width or height, whichever is higher, is taken as the diameter of the node and is used in the formula. For this calculation, we use the “inside a circle” mathematical formulation (see Equation 4.1 and [37]). The basic idea of this formula is that a point will be inside a circle if its distance from the center is at most the value of the radius. After the storage of the data, the system will check and find the node that has the most points inside and then change its color (see Fig. 4.4). This identifying technique is just the basic foundation for adaptive visualizations and could, for instance, be extended to show detailed attributes of a focused node or zoom in on a focused object. It also can be extended to other graph objects in the graphs, like edges.

$$\sqrt{|x_p - x_c|^2 + |y_p - y_c|^2} < r.$$

Equation. 4.1 Mathematical formula. It is used to check if the tracking points are inside the nodes. In this formula x_c and y_c represents the center of the circle, x_p, y_p are the points and r is the radius of the circle.0



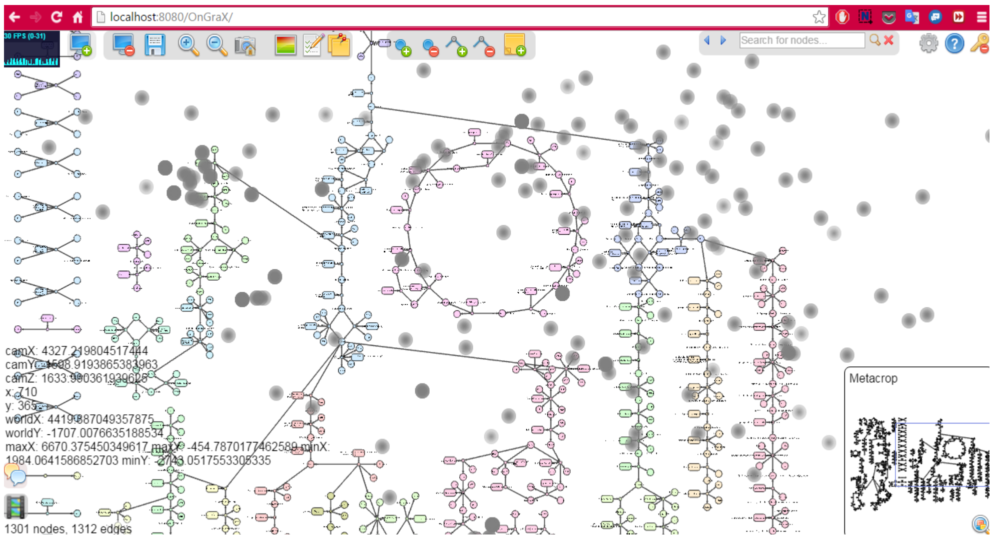
```
camX: 155.79617
camY: 515.1881
camZ: 729.2467
x: 1063
y: 396
worldX: 737.6239968167534
worldY: 419.50708861375745
maxX: 1201.541948011277 maxY: 1025.8121365545546 minX:
-889.9496080112771 minY: 4.5640634454452425
3 nodes, 1 edges
```

Fig. 4.4 Graph object identification. Identification of a graph object under user focus. The node color turns to blue while it is in the user’s focus.

4.3.4 Basic Heatmap Creation from the Live Tracking Data During a Session

To show users a heatmap based on the live eye tracking data, the heatmap visualization is implemented in the HeatmapRenderer.js file. Inside this file, we create a transparent canvas on top of the one in OnGraX. To be able to draw points on this canvas, x and y coordinates are taken from the “MyBuffer” object, where we store the data for visualization. For convenience, we decided on the proper inner and outer radius of the points. The gradient of the points is set and the coloring is done in a grayscale encoding.

When the users start a session with the eye trackers and activate the real time heatmap option from the website, the system starts to draw a heatmap with their eyes according to the tracking data. This heatmap is created from the live data which comes from the eye trackers. Users have the opportunity to see the points that they are looking at on the screen, during a graph session. Figure 4.5 shows a real time tracking data visualization during a graph session.



```
camX: 4327.219804517444
camY: -1598.9193865383963
camZ: 1633.980361939625
x: 710
y: 365
worldX: 4419.887049357875
worldY: -1707.0076635188534
maxX: 6670.375450349617 maxY: -454.787017746258 minX:
1984.0641586852703 minY: -2743.0517553305335
1301 nodes, 1312 edges
```

Fig. 4.5 Heatmap by tracking data during a graph session.

4.3.5 Recording Process of the Eye Tracking Data

In the recording process of the tracking points, tables are created in a MySQL database according to the graph sessions that users are joining. Inside these tables, user id, x, y, and time are stored. While a user is in a graph session, the system will check the graph name that the user is in, and if a table with the graph name already exists in the database, the data will be stored in that table. If not, a table with that graph name will be created and the user id, time, and x/y tracking points will be saved into that table. The current graph session and graph name are taken from the user's session object. The variables that are stored in the database are identified for the insert operation. The tracking data is acquired through messages that are coming from the eye tracking client, the user id is taken from the user session and time is provided from the MySQL timestamp.

4.4 Eye Tracking Client User Interface Design and General Functionalities

In this section, the design and functionality of the eye tracking client will be explained in detail. The primary goal of the eye tracking client is to give the users a quick and easy way to control the eye trackers and use them with OnGraX. In order to enable the users to use the system's functionalities, we implemented two different user interface windows.

The first user frame is a login window. A basic design is done by setting the bounds of each element on the GUI. Basic GUI elements that are used: button, combo box, text field, password field, and labels. The implementation of this frame is done in the Firstframe.java class in the OnGraX eye tracking client. The combo box in this window contains two options to choose from, which are the Mirametrix S2 and TheEyeTribe trackers. Username and password are expected to be entered by the users (see Fig 4.6). The username and password that the user enters have to be the same as the username and password that is logged in to the OnGraX' web page. If they are not the same, the system will not allow the users access to the program and will not do any actions. After the user presses the login button with the correct information, according to the device that he/she will use, the related calibration window opens for that tracker and the second user window opens.

The second frame, the OnGraX eye tracking client window, opens after the user logs in successfully from the first one (see Fig 4.7). This window includes a slider, start/stop sending and star/stop recording buttons with a white drawing panel which has a cursor on it. In OnGraxEyeTrackingClient.java class, the functionalities of these GUI elements are implemented. The white drawing panel's size is given relative to the aspect ratio of the screen and a blue rectangle represents the web site from OnGraX. The red cursor which is on this panel is in the shape of a cross. It moves on the white drawing panel according to the eye tracking and can be used to see if the calibration process was successful and that the eye tracker is working properly. The start sending and stop sending buttons are responsible for sending the eye tracking data from the client to the server, and from there that data goes to the web content of the project. The functionality of this button is given by startSendingData() and stopSendingData() methods. To be able to start and stop recording the data into the table, appropriate buttons are provided. Recording and stop recording methods that are implemented on the client side of the project are responsible for the communication to the server side for the recording process. Inside these methods, proper attributes are sent to the server with the actions start recording tracking data and stop recording tracking data. On the server, the class EyeTrackerActionHandler.java is responsible for handling all these actions and messages.

An additional slider is used to set how long stored points are used to identify graph objects (Sect. 4.3.3). For example, if the user sets the slider to four seconds, the system will only use the stored points that are sent in the last four seconds from the eye trackers during a graph session on the OnGraX web page.

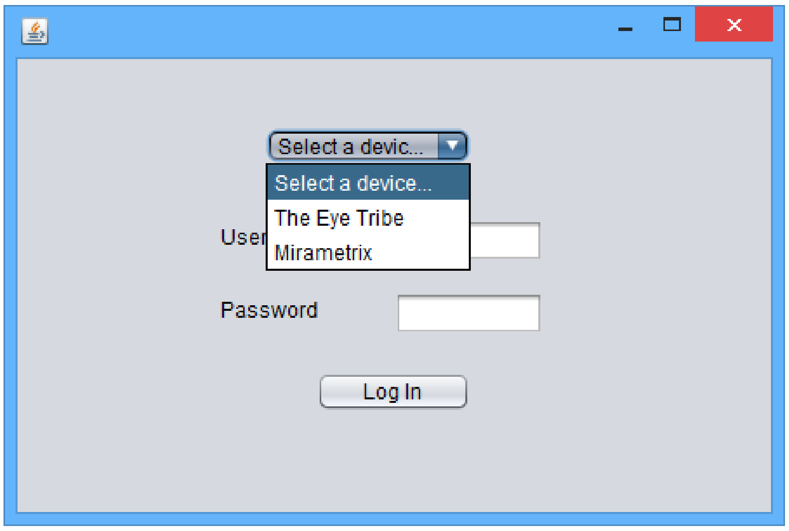


Fig. 4.6 Login window

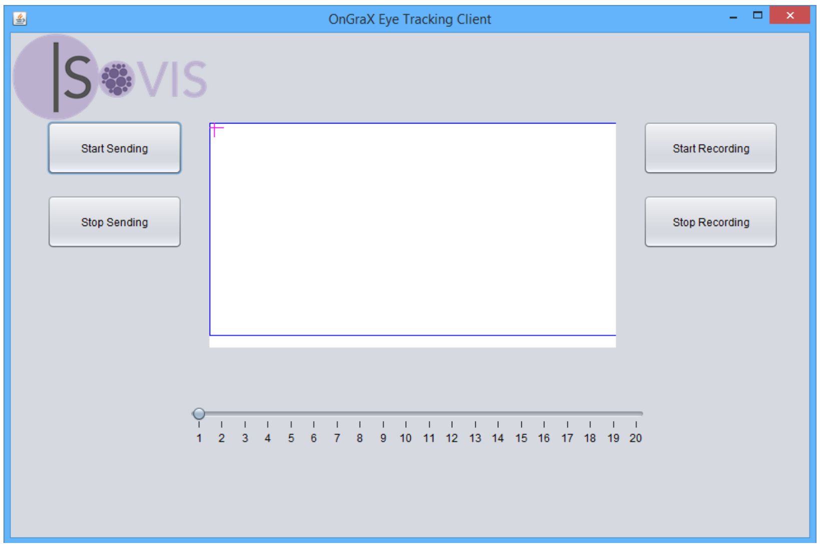


Fig. 4.7 Eye tracking client window

4.5 Comparison and Limitations of the Eye Trackers Based on our Experience

This section will explain the advantages, disadvantages and limitations that we discovered on both Mirametrix S2 and The Eye Tribe tracker during our project.

The APIs of the trackers are an important part of the implementation of the project after the building of the communication process between the trackers and our system. We realized that The Eye Tribe tracker's API is more understandable and easier to implement with compared to the Mirametrix S2 API. The way that the trackers provide this data is different from each other. The Eye Tribe tracker provides the tracking data as pixels. These pixels are oriented from the top left of the screen. Eye gaze data from this tracker is also available in both smoothed and raw coordinates. For the Mirametrix S2 eye tracker, it is a little bit different. It provides the data in percentages of the screen size, it passes data in XML type strings, and it can also give coordinates for both eyes separately. In our project, this data is first read and converted to double type, and then (to be able to convert this data to pixels) additional operations are done. Width and height values of the screen are taken from the system and used for the converting process of the data. Another compelling thing for The Eye Tribe tracker is that it works only with USB 3.0, which with the new generation computers this does not create a big problem for developers and users.

We also faced some limitations of the devices. The Eye Tribe tracker had problems during the calibration process while being near a window. Because of this, we had a lot of "calibration failed" warnings. Also, during our user study, we realized that The Eye Tribe tracker is getting overheated quickly and this affected the calibration process of the device. Because of this reason, it had problems detecting the participants' eyes during the calibration and task solving process. The Mirametrix S2 eye tracker completed the calibration successfully under the same conditions. This eye tracker also offers a more detailed calibration window. The calibration interface includes a scale with red, green and blue colors, and a bigger screen where users can see their eyes clearly. With this scale on the interface, users can arrange their sitting distance to the device in order to gain a better quality calibration. After the calibration of both eye trackers, a window opens to show the calibration results and give the users a chance to see if the tracker is really following their eyes. The window that opens for the Mirametrix S2 after calibration shows the calibration results in a more detailed way than The Eye Tribe tracker. In this window, users can follow the point to test if the calibration is of good quality. For the Mirametrix S2, this window also shows the deviation of the right, left and the average of the right/left gaze. Both trackers had problems tracking the eyes if the users moved their heads. Therefore, users had to sit in front of the trackers and be careful not to move their heads too much. Also during the calibration process for the Mirametrix S2 eye tracker, some of the users in our study (Sect. 5) mentioned that they had to open their eyes more in order to make the tracker see their eyes properly. These two situations caused an environment that was not very natural and comfortable for the users.

5 User Study

In this chapter, we discuss and describe the user study. We also explain the process that we followed in order to finish it.

5.1 The Task

For the task that the users had to solve, we imported a graph in OnGraX that depicts the map of Europe and Asia with the name eRoads, as shown in Figure 5.1. Major cities are represented by circle nodes and their names and lines between them show their road connections. The task we gave to the users was to identify the cities of Barcelona and Berlin and try to find the links and cities between them in order to travel from the first to the second city.

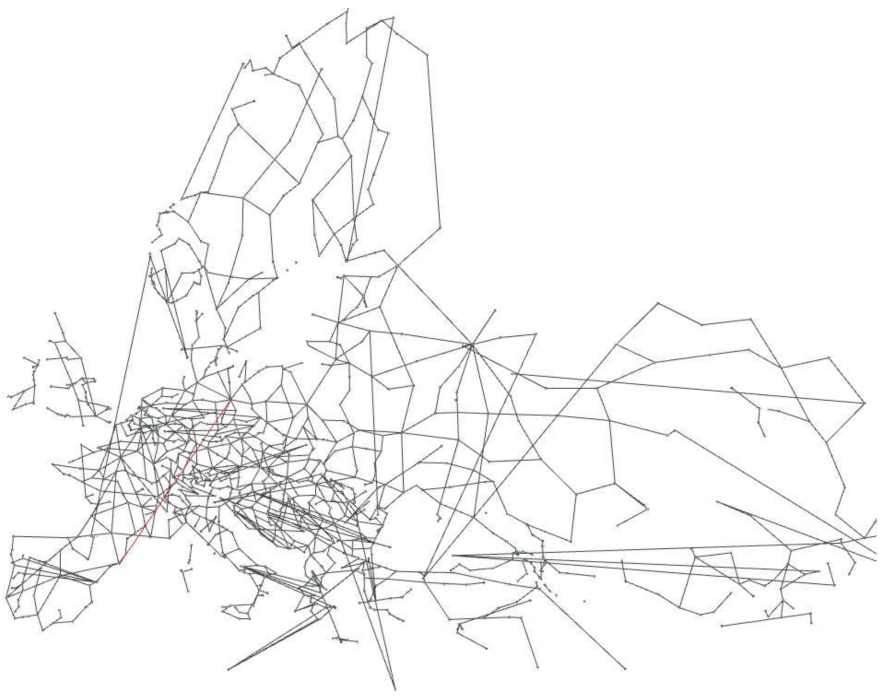


Fig. 5.1 eRoads map based on [29].

5.2 Users testing with The Eye Tribe and the Mirametrix S2 Eye Trackers

As a first step for the user study and the meeting with the participants, we prepared a questionnaire with seven Likert Scale questions, where they are asked to select one of the numbers between one and six, and one open question, so they can write about their experience. These numbers represent things like; poor or excellent quality degree during the task, ease or difficulty of the task or the usage of the trackers, etc. The last question is an open question where we asked the participants to write down their general comments about the experience with the eye trackers and the task.

When we invited the study participants, we asked them if they would have any issues with us recording their session. They agreed as we explained to them that they would not have to face the camera and that the videos would be used for research purposes and some pieces of them for the presentation of the test.

Before the arrival of the participants, we put the camera in the right position and prepared the graph that would be used for the testing. When they arrived, we asked them to sit in front of the computer and introduced them to OnGraX and showed them another graph, so they could get accustomed to the tool. After that, we explained them the task that we would like them to do and showed them a map of Europe, just in case they were not familiar with the geographical positions of the two cities.

Additionally, we let them try out the eye tracker before the actual test, so they could get comfortable with it too. After that, we gave them their usernames and passwords and explained the order of the steps they needed to make in order to complete the test correctly: First, they logged in to the OnGraX website with the username and password given and entered the graph called “eRoads”. Next, we ran our program and opened the eye tracker’s user interface. After that, the participants started the calibration process with the eye tracker, then logged into the eye tracking client with the same username and password they used in OnGraX. After our eye tracking client’s GUI opened, they pressed the “Start Sending” button, the “Start Recording” button and finally returned to the browser window with the OnGraX web page to find their path from Barcelona to Berlin. While they searched their path to the destination city, we asked them to tell us every node/city that they passed. After they finished, they pressed on the GUI’s “Stop Recording” and “Stop Sending” button. Finally, they logged out from OnGraX, and we continued to give them the questionnaire with the eight questions.

5.3 Results from Questionnaire

At the end of the task, the Likert scale questionnaire has been offered to the participants. With this questionnaire, they had the chance to say their opinion about the task and answered questions about their experience with the trackers.

The tables below show the collected data for each question in the Likert scale questionnaire (see Fig. A.1 and Fig. A.2 in the appendix). The first

Table 5.1 is the data gained from the group which used Mirametrix S2 eye tracker and the second Table 5.2 is the group which completed the task with The Eye Tribe tracker.

The first column of the tables shows the users in the groups and the first line of the tables contains the asked questions, i.e., Q1, Q2, Q3, Q4, Q5, Q6, and Q7 which are:

Q1: How good would you rate the capability of the eye tracker following your eye movement?

Q2: How much trouble did you have with the calibration process?

Q3: How would you rate the interface of the tracker?

Q4: How would you rate the difficulty of the task provided?

Q5: How much would you rate your experience in using computers?

Q6: Are you familiar with network drawing?

Q7: Have you used eye trackers before?

User ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7
1	6	5	5	3	5	1	1
2	6	1	5	5	5	5	1
3	5	4	5	3	5	1	1
4	6	1	5	2	6	5	3

Table 5.1 Answers given by each user from Group 1.

User ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7
5	5	3	5	4	6	3	2
6	5	5	5	3	5	2	1
7	5	4	5	2	4	2	1
8	5	5	2	4	5	4	1

Table 5.2 Answers given by each user from Group 2.

With the data from the first table, we can see that for Q1, most of the users chose six. In the second group, they all selected five, so we can see that with a little difference, the Mirametrix S2 eye trackers' capability of following the eye movements is better than The Eye Tribe tracker. For the second question Q2, we can clearly see from the given answers that the second group had more trouble with the calibration process than the first group users. The answers to the third question Q3 show that the interface of the Mirametrix S2 eye tracker is more detailed and understandable for users. The averages of Q4 show that the users found the difficulty of the task normal in both of the groups, and the Q5 indicates that both groups have an adequate knowledge of computers. The numbers that the users chose as an answer for the Q6 are very close. This points out that the users from both groups do not have a lot of knowledge in network drawings. Finally, from the answers that are given to the last question Q7, it is shown that both of the groups did not have previous experience with eye trackers.

To conclude, in the first, second and third question, we have the opportunity to see some of the technical and conceptual limitations on the use of the eye trackers (RQ2) and also see some of the differences between the two eye trackers according to user experience (RQ1).

From the data that we show in the tables, we create two bar charts. They show the average of the values that are given by the users to each Likert question. Fig. 5.2 is based on the data from the first group and Fig. 5.3 shows the results from the second group. The first group answered the questions according to the experience that they had by completing the task with the Mirametrix S2 while the second group did it with The Eye Tribe tracker. The vertical axis of the bar chart represents the numbers that are used to answer the Likert questions, and the horizontal axis is showing the seven questions. Blue colored bars display the average value that is chosen by the users for the questions, and the black thin lines represent the standard deviation. "In statistics, the standard deviation (SD, also represented by the Greek letter sigma σ or s) is a measure that is used to quantify the amount of variation or dispersion of a set of data values." [31].

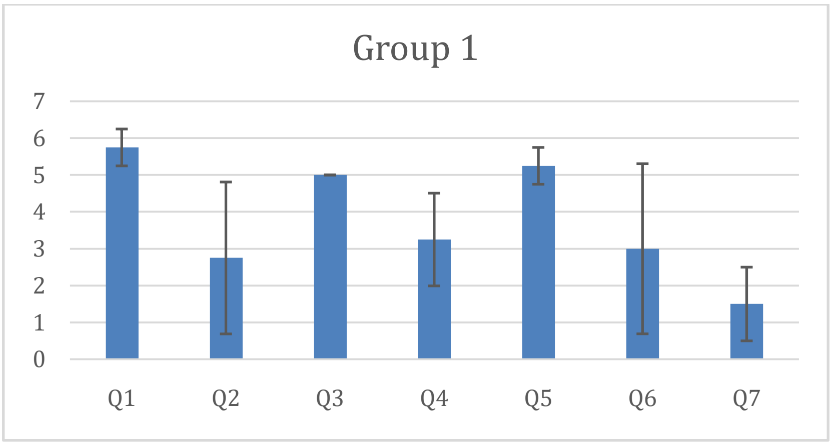


Fig. 5.2 Bar chart of Group 1. Chosen values for each of the Likert type questions of the first group (Mirametrix S2) and the standard deviation.

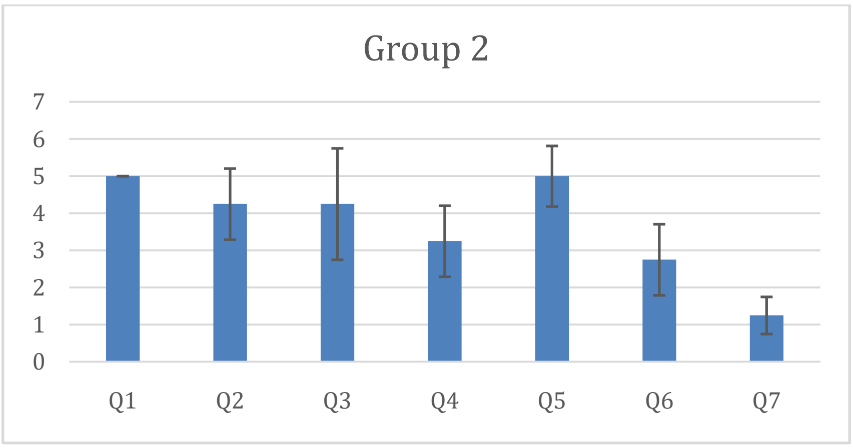


Fig. 5.3 Bar chart of Group 2. Values that participants of the second group (The Eye Tribe) chose for answering the Likert type questions and the standard deviation.

Quantitative results that are gained from the user study: Time was measured (in minutes and seconds) during the tasks for every user session. Time measurement was stopped when the participants finished the given task. For the first group with the Mirametrix S2 eye tracker, the average completion time was 4 minutes, and for the second group with The Eye Tribe tracker, 6 minutes. In the same way, time was measured for the completion of the questionnaire. This questionnaire was done by each user after they completed the task. The average time for the first group members to answer the questions was 4 minutes and 10 seconds, when for the second group it was 3 minutes.

Qualitative results that are gained from the user study: We asked all participants to comment on their experience, and most of them answered that it was interesting and fun. All of the participants that used The Eye Tribe tracker, complained about the calibration quality, and they said that it took a long time for the tracker to find their eyes.

5.4 Analysis of Created Heatmaps

This part contains the analysis of the heatmaps created based on the tracking data or viewport, respectively.

5.4.1 According to Data Collected from the Eye Trackers

After the user study, in order to see the results in a clear way, heatmaps are created from the data in the database. Figure 5.4 shows a heatmap from the data that is collected from one of the users who used the Mirametrix S2 Eye tracker. Figure 5.5 is another heatmap which is created from the data of The Eye Tribe tracker. In both heatmaps, in some nodes/cities, colors are going from purple to red and then yellow, meaning that the users paid more attention to them. In the user study, we asked our participants to go from Barcelona to Berlin. First, the participants tried to locate both cities on the map, and then they followed the linked nodes to go from one city to another. From both figures, we can see the path that these two participants followed to complete the task.

From the heatmaps, we can see the differences between the two eye trackers. In Fig. 5.4 we can see that the gaze points are created more continuously, and in Fig. 5.5, in some places of the heatmap, there are gaps. One of the biggest reasons for such a phenomenon is that during the calibration The Eye Tribe tracker had difficulties finding the user's eyes. This problem continued during the whole task. On the other hand, the Mirametrix S2 eye tracker did not have any problems with calibration, so during the task, it caught all the points that the participants looked at. From this, we can say that the Mirametrix S2 eye tracker gives more accurate results than The Eye Tribe tracker. Also, another finding of our user study is that when a participant focuses on exactly one node/city in the map, The Eye Tribe tracker generates points on the focused node/city, but it also creates extra dots (points) around it. In Figure 5.5 (A), we can see these extra points that The Eye Tribe tracker created while a node is in user’s focus. For example, when we check Spain, we can see that there are a lot of extra created points around Barcelona (the focused node), see Fig. 5.5. The Mirametrix S2 eye tracker didn’t create a lot of extra points. We can see from Figure 5.4, the focused cities are more evident, and they don’t have extra scattered dots around them. With this result, we can say that it is easier to understand the focusing point of the user with the Mirametrix S2 eye tracker.

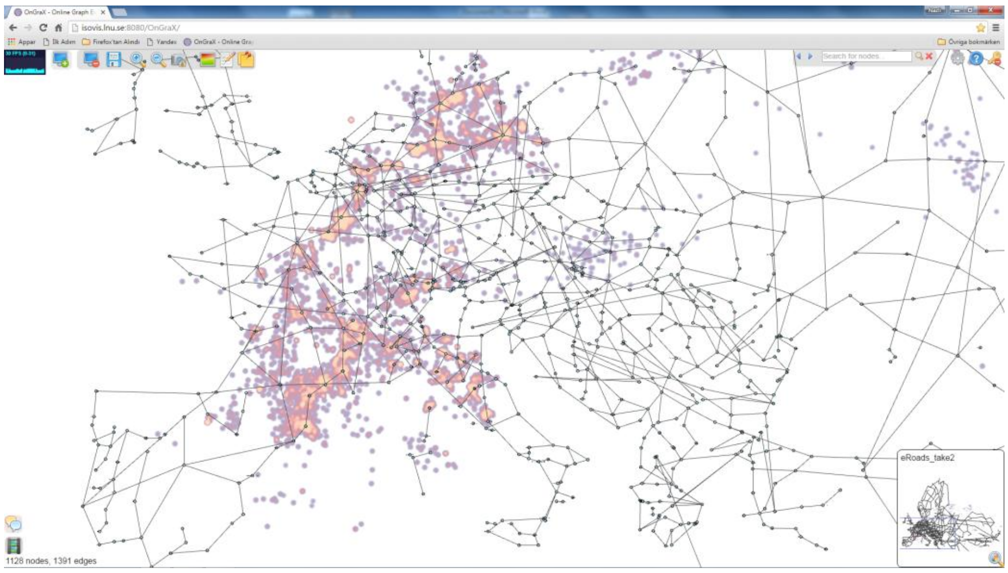


Fig. 5.4 Created heatmap according to the Mirametrix S2 Eye tracker data.

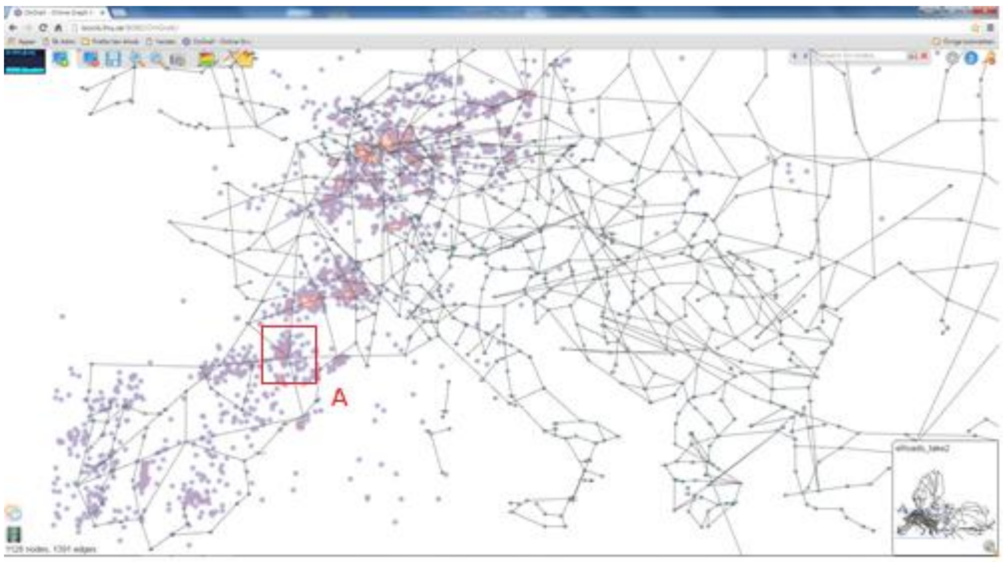


Fig. 5.5 Created heatmap according to The Eye Tribe tracker data. (A) marks a region of a focused node with a lot of scattered points due to bad tracking quality.

5.4.2 According to Data Collected from Viewports

In Figure 5.6, a heatmap is shown that was created from the viewports of one of the participants. When the graph view opens, the default screen view is the area near Moscow. For that reason, we can see that there is a heatmap drawn in that area. While the participant started panning to the left part of the graph, the viewport was altered and showed areas near where the participant was looking at and not only where he/she did actually look. When the user caught sight of Barcelona he/she immediately started going his/her way to Berlin. In contrast, Figure 5.7 shows a heatmap generated from eye tracking data. We can see that the path the participant followed in Fig. 5.6 is not clearly visible as compared to Figure 5.7, as the participant did not focus enough time on some nodes in order for the heat map to be shaped accordingly (more red and yellow). For instance, according to our tracking data in Fig. 5.7, the participant went through Paris, but this fact cannot be seen in Figure 5.6. The path the participant followed was Barcelona, Toulouse, Orleans, Paris, Mons, Brussels, Leuven, Hasselt, Heerlen, Cologne, Dortmund, Bad Oeynhausen, Hanover, Braunschweig, Magdeburg, and finally Berlin.

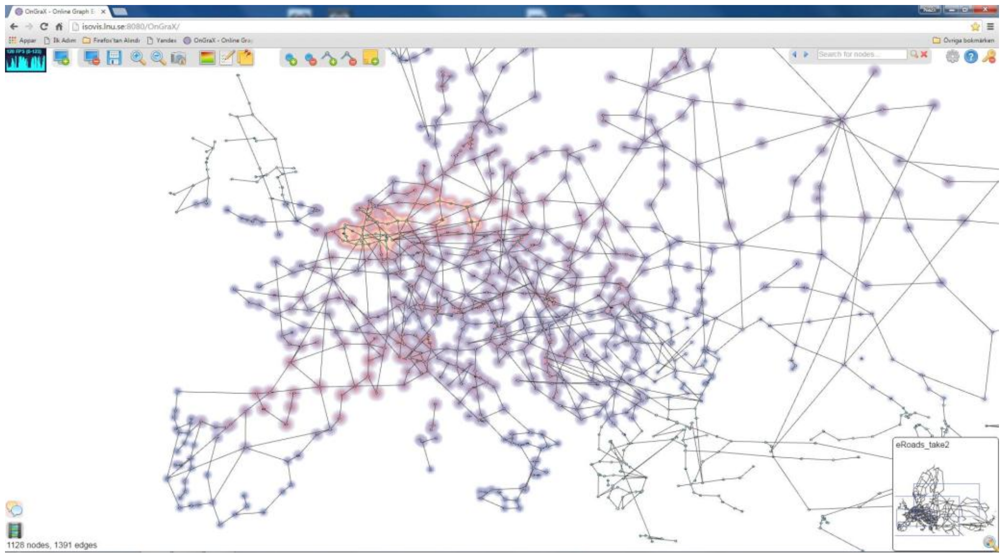


Fig. 5.6 Created heatmap according to OnGraX' viewports.

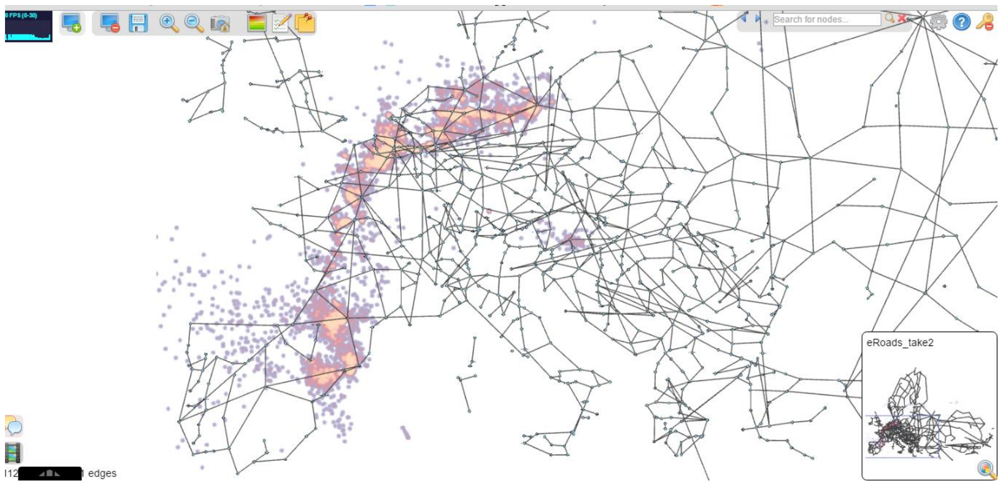


Fig. 5.7 Created heatmap according to the Mirametrix S2 tracker data.

6 Discussion

We use two eye trackers in our project. According to our findings, we realize that the Mirametrix S2 is more effective than The Eye Tribe tracker in the context of our research. The Mirametrix S2 is easier to work with, as its calibration does not get so easily affected by light, glasses and other factors. The Eye Tribe tracker showed evidence that it cannot be used for a long period of time, as it gets overheated very fast. Also, The Eye Tribe's way of eye tracking is not very accurate and results in a lot of jitter. As a result, we see that the Mirametrix S2 is living up to the expectations of being a high quality eye tracker and it is worth its price as it gives better results. In addition, The Eye Tribe tracker being a lot more economical does not affect its effectiveness too much. It is not as good as the Mirametrix S2, but it can still give adequate results. With these conclusions, we answer our first and second research questions (RQ1 and RQ2).

For the third research question (RQ3), and with the help of the user study, the answer would be that the heatmap quality of the viewports in OnGraX can be improved by the use of the eye trackers since they are more accurate with showing the user's focus points. They show exactly where the users were looking at through the whole duration of the task. That way, in the case of a collaborative graph session (with other users), the rest of the users would see where each one is looking at exactly. So, finally, the heatmap is more defined and detailed.

With the implementation of our project, we can answer the last research question (RQ4) as we identify graph nodes that are in the focus of the user, so they can be adapted visually according to the future developers' preference. In our implementation, we resulted in making a node that is in the user's focus change color (turn to blue).

7 Conclusion

From our results, by answering research question 1 (RQ1), we conclude that for our research, the Mirametrix S2 is a better choice for eye tracking. As an answer to our third research question (RQ3), we can come to the conclusion that eye tracking is a more accurate way of identifying the user's focus than the viewports in OnGraX as the viewports show areas that the user might have not even looked at, at all, making the other users not quite sure of where their colleagues paid more attention in the graph. To answer this question more accurately, in the future there could be a measurement of the data collected and statistical analysis of it, and not only a comparison based on our experience and knowledge on the two trackers. This fact can also be applied to the second research question, measuring exactly how much are the limitations of the eye trackers, give also quantitative results. Finally, answering our fourth research question (RQ4), we can say that adaptive visualizations are possible with the use of eye trackers and they can be designed in any way the developer wants. If we would have had more time we could have also applied this to other graph elements and not only to a node, and also we would have made different kind of adaptive visualizations. Our results are relevant and can be helpful for scientific and societal research.

7.1 Future Research

As it was natural, due to time and resources, we were not able to extend our project too much. For further and future research, there could be more implementation on adapting visualizations with eye tracking data since we gave the foundation for our fourth research question. For example, there could be a depiction of the characteristics of the nodes that are in the focus of the eye or change the node's heatmap colors and intensity when it is in the focus of the eyes. In Section 4.3.3, it is explained that the identification of the nodes is done by changing their colors while they are in the user focus. Also for future research, identification can be applied to other graph objects, like edges. The project could also be tested with other brands of eye trackers. In addition, future researchers can try to redo the same project but with more different types of eye trackers and also with a bigger amount of people for the user study.

References

[1] Kenneth Holmqvist and Marcus Nyström. "*Eye tracking – A comprehensive guide to methods and measures"*. Oxford University Press, 2011.

[2] Björn Zimmer and Andreas Kerren. "Applying Heat Maps in a Web-Based Collaborative Graph Visualization". Conference: *Poster Abstract, IEEE Information Visualization* (InfoVis '14), Paris, France, 2014.

[3] Björn Zimmer and Andreas Kerren. "Displaying User Behavior in the Collaborative Graph Visualization System OnGraX". Proceedings of the *23rd International Symposium on Graph Drawing and Network Visualization* (GD '15), pages 247-259, Volume 9411 of LNCS, Springer, Los Angeles, CA, USA, 2015.

[4] Ben Steichen, Giuseppe Carenini, and Cristina Conati, "Adaptive Information Visualization - Predicting user characteristics and task context from eye gaze". Poster Proceedings of the *20th Conference on User Modeling, Adaptation, and Personalization*, Montreal, Canada, 2012.

[5] Ben Steichen, Giuseppe Carenini, and Cristina Conati, "User-Adaptive Information Visualization - Using Eye Gaze Data to Infer Visualization Tasks and User Cognitive Abilities". Proceedings of the *2013 international conference on Intelligent user interfaces*, Vancouver, Canada, 2013.

[6] Tomasz D. Loboda and Peter Brusilovsky. "User-adaptive explanatory program visualization: evaluation and insights from eye movements". *User Modeling and User-Adapted Interaction*, pages 191-226, Volume 20 Issue 3, Springer Science+Business Media B.V., 2010.

[7] Jae-wook Ahn and Peter Brusilovsky, "Adaptive visualization for exploratory information retrieval", *Information Processing & Management*, pages 1139-1164, Volume 49 Issue 5, 2013.

[8] Hsin-Yang Ho, I-Cheng Yeh, Yu-Chi Lai, Wen-Chieh Lin, and Fu-Yin Chern. "Evaluating 2D Flow Visualization Using Eye Tracking". *Eurographics Conference on Visualization* (EuroVis), Volume 34 Issue 3, 2015.

[9] Cristina Conati, Giuseppe Carenini, Derevk Toker, and Sebastien Lalle. "Towards User-Adaptive Information Visualization". Proceedings of the

Twenty-Ninth AAA1 Conference on Artificial Inteligence, pages 4100-4106, Vancouver, Canada, 2015.

[10] Dereck Toker, Cristina Conati, Ben Steichen, and Giuseppe Carenini. “Individual User Characteristics and Information Visualization: Connecting the Dots through Eye Tracking”. Proceedings of the *SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, Canada, 2013.

[11] Arie E. Kaufman, Amit Bandopadhay, and Bernard D. Shaviv. ”An Eye Tracking Computer User Interface”. Proceedings of the *IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, San Jose, CA, USA, 1993.

[12] Ben Steichen, Oliver Schmid, Cristina Conati, and Giuseppe Carenini. “Seeing how you’re looking -Using Real-Time Eye Gaze Data for User-Adaptive Visualization”. Proceedings of the *User Modeling, Adaptation and Personalization (UMAP)*, pages 1-4, Rome, Italy, 2013.

[13] Owen A. Kelley. ”Adapting an existing visualization application for browser-based deployment: A case study from the Tropical Rainfall Measuring Mission”. *Computers and Geosciences*, pages 228-237, Volume 51, 2013.

[14] Robert Moro, Jakub Daraz, and Maris Bielikova. “Visualization of Gaze Tracking Data for UX Testing on the Web”. Proceedings of the *25th ACM Hypertext and Social Media Conference*, Santiago, Chile, 2014

[15] Björn Zimmer and Andreas Kerren. “Harnessing WebGL and WebSockets for a Web-Based Collaborative Graph Exploration Tool”. *Engineering the Web in the Big Data Era*, pages 583-598, Volume 9114, 2015.

[16] Riccardo Mazza. *Introduction to Information Visualization*. Springer, 2009.

[17] O.Špakov and D.Miniotas. “Visualization of Eye Gaze Data using Heat Maps”. Electronics and Electrical Engineering, Volume 115 Issue 2, 2007.

[18] Jakob Nielsen and Kara Pernice. *Eyetracking Web Usability*. New Riders, 2010.

[19] Jesper Andersson. Class Lecture, Topic: “Software Development Inception-Phase”. Linnaeus University, Computer Science Department, Växjö, Sweden, 2015.

[20] Gornik Davor. “IBM Rational Unified Process: Best Practices for Software Development Teams”. *International Business Machines Corporation*, 2003.

[21] The Eye Tribe, (2014). [Online, Last access: 2016-04-23]. Available: http://dev.theeyetribe.com/java/.

[22] Shawn Graham, Ian Milligan, and Scott Weingart. “Principles of Information Visualization” The Historian's Macroscope: Big Digital History. Under contract with Imperial College Press. Open Draft Version, (2013). [Online, Last access: 2016-05-16]. Available: http://www.themacroscope.org/?page_id=469.

[23] Johan Hagelbäck. Class Lecture, Topic: “Scientific Methods in Computer Science”. Linnaeus University, Computer Science Department, Växjö, Sweden, 2015.

[24] Mirametrix. *Mirametrix API*, 2015.

[25] Mirametrix. *Mirametrix S2 User Guide*, 2015.

[26] Linnaeus University. (2015), Course Room for Degree Projects, *Reliability*. [Online, Last Access: 2016-05-18]. Available: https://coursepress.lnu.se/subject/thesis-projects/reliability/

[27] Linnaeus University. (2015), Course Room for Degree Projects, Validity. [Online, Last Access: 2016-05-18]. Available: https://coursepress.lnu.se/subject/thesis-projects/validity/

[28] Daniel Cernea and Andreas Kerren, “A survey of technologies on the rise for emotion-enhanced interaction”. *Visual Languages and Computing*, pages 70-86, Volume 31, 2015.

[29] Wikipedia. *International E-road network*, (2016), [Online, Last access: 2016-05-18]. Available: https://en.wikipedia.org/wiki/International_E-road_network

[30] The Cyclosys. *Methodology,* (2015), [Online, Last access: 2016-05-16]. Available: http://www.cyclosys.com/Practices/MethodologiesFramework

[31] Wikipedia. *Standard Deviation*, (2016), [Online, Last access: 2016-05-19]. Available: https://en.wikipedia.org/wiki/Standard_deviation

[32] Codeflow. *High Performance JS heatmaps,* (2013), [Online, Last access: 2016-05-19]. Availvable: http://codeflow.org/entries/2013/feb/04/high-performance-js-heatmaps/

[33] The Eye Tribe. *Our Big Mission*, (2014). [Online, Last access: 2016-04-23]. Available: http://dev.theeyetribe.com/about/

[34] Wikipedia. *Vertex (graph theory)*, (2015), [Online, Last access: 2016-05-19]. Available: https://en.wikipedia.org/wiki/Vertex_(graph_theory)

[35] Wikipedia. *WebSocket*, (2016), [Online, Last access: 2016-05-19]. Available: https://en.wikipedia.org/wiki/WebSocket

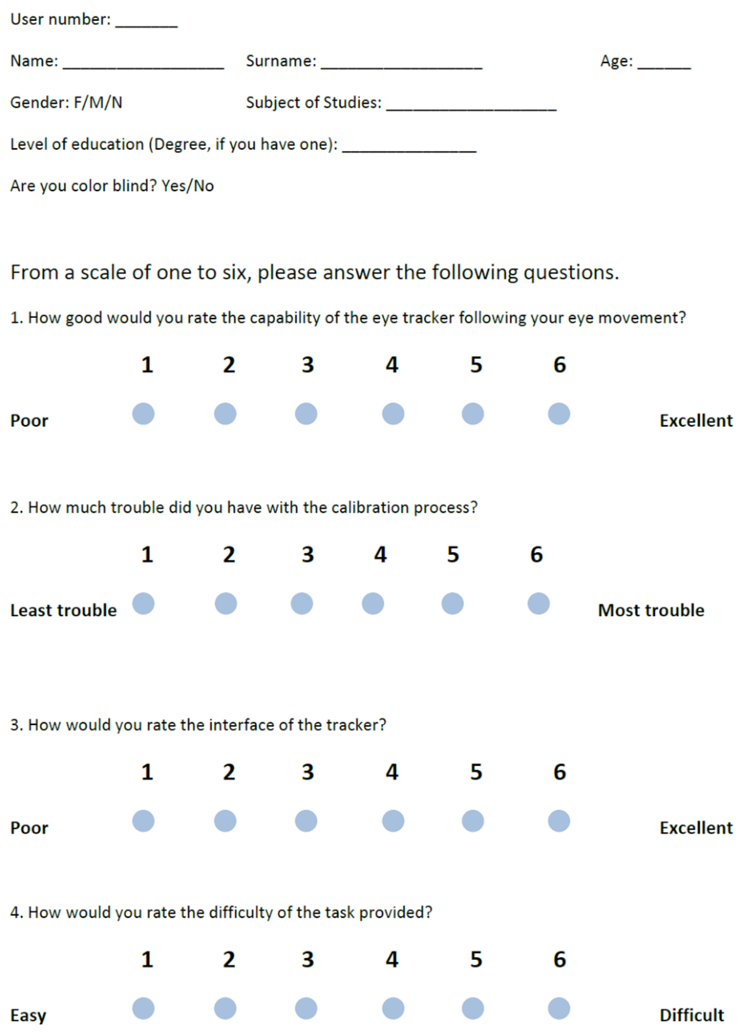
[36] Wikipedia. *WebGL*, (2016), [Online, Last access: 2016-05-18]. Available: https://en.wikipedia.org/wiki/WebGL

[37] Andreas Kerren, Introduction to Computer Graphics, (2015). [Online, Last access: 2016-07-12]. Available:
http://cs.lnu.se/isovis/courses/fall15/1dv800.html

Appendix 1

In this part of the appendix, we provide all the raw data from the user study.

Likert Questionnaire



User number: _______

Name: __________________ Surname: __________________ Age: ______

Gender: F/M/N Subject of Studies: ___________________

Level of education (Degree, if you have one): _______________

Are you color blind? Yes/No

From a scale of one to six, please answer the following questions.

1. How good would you rate the capability of the eye tracker following your eye movement?

	1	2	3	4	5	6	
Poor	●	●	●	●	●	●	**Excellent**

2. How much trouble did you have with the calibration process?

	1	2	3	4	5	6	
Least trouble	●	●	●	●	●	●	**Most trouble**

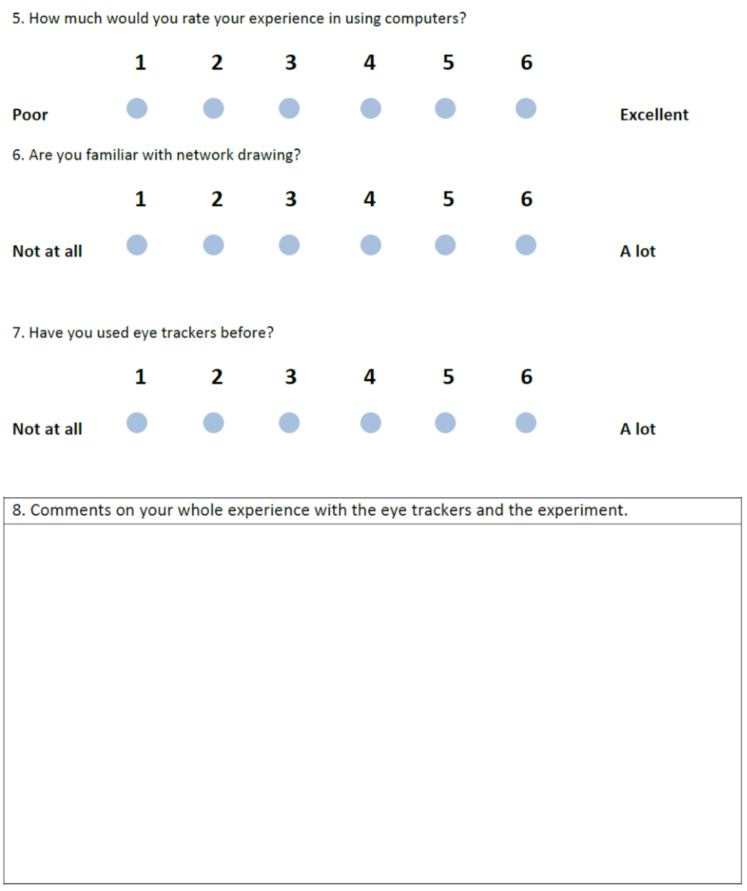
3. How would you rate the interface of the tracker?

	1	2	3	4	5	6	
Poor	●	●	●	●	●	●	**Excellent**

4. How would you rate the difficulty of the task provided?

	1	2	3	4	5	6	
Easy	●	●	●	●	●	●	**Difficult**

Fig. A.1 First page of the Likert Scale questionnaire that was given to the users after they completed the task.



5. How much would you rate your experience in using computers?

	1	**2**	**3**	**4**	**5**	**6**	
Poor	●	●	●	●	●	●	**Excellent**

6. Are you familiar with network drawing?

	1	**2**	**3**	**4**	**5**	**6**	
Not at all	●	●	●	●	●	●	**A lot**

7. Have you used eye trackers before?

	1	**2**	**3**	**4**	**5**	**6**	
Not at all	●	●	●	●	●	●	**A lot**

8. Comments on your whole experience with the eye trackers and the experiment.

Fig. A.2 Second page of the Likert Scale questionnaire and an open question.

Appendix 2

Collected Data from Questionnaire

Collected data from users who used the Mirametrix S2 tracker:

	1	2	3	4	5	6
Q1	0	0	0	0	1	3
Q2	2	0	0	1	1	0
Q3	0	0	0	0	4	0
Q4	0	1	2	0	1	0
Q5	0	0	0	0	3	1
Q6	2	0	0	0	2	0
Q7	3	0	1	0	0	0

Table A.2.1 Frequency Distribution

	Average	Std. Deviation	Mode	Low limit	High Limit
Q1	5,75	0,500	5	5,250	6,250
Q2	2,75	2,062	5	0,688	4,812
Q3	5	0,000	5	5,000	5,000
Q4	3,25	1,258	3	1,992	4,508
Q5	5,25	0,500	5	4,750	5,750
Q6	3	2,309	2	0,691	5,309
Q7	1,5	1,000	1	0,500	2,500

Table A.2.2 Other Statistics

Collected data from users who used The Eye Tribe tracker:

	1	2	3	4	5	6
Q1	0	0	0	0	4	0
Q2	0	0	1	1	2	0
Q3	0	1	0	0	3	0
Q4	0	1	1	2	0	0
Q5	0	0	0	1	2	1
Q6	0	2	1	1	0	0
Q7	3	1	0	0	0	0

Table A.2.3 Frequency Distribution

	Average	Std. Deviation	Mode	Low limit	High Limit
Q1	5	0,000	5	5,000	5,000
Q2	4,25	0,957	5	3,293	5,207
Q3	4,25	1,500	5	2,750	5,750
Q4	3,25	0,957	3	2,293	4,207
Q5	5	0,816	5	4,184	5,816
Q6	2,75	0,957	2	1,793	3,707
Q7	1,25	0,500	1	0,750	1,750

Table A.2.4 Other Statistics

The answers given in the open question:

Mirametrix

User1: ”My problem was more geography than the eye trackers. It was good to use the eye trackers, but maybe a bit funny when trying not to move your head.”
User2: “Fun.”
User3: “Eye tracker calibration was smooth, fast and fun, I still remember some geography.”
User4: ”It is the first time for me to experience the eye-tracker, fun experience, but had a little bit of hassle while searching the cities, overall, a great experience.”

The Eye Tribe

User5: “The task was very well formulated, the calibration software was easy to use and intuitive, eye movement recognition was quite precise, the dots on the graph were not equally distributed.”
User6: “It was fun.”
User7: “It was fun and interesting. The device is a bit difficult to calibrate at first, but then it works pretty well.”
User8: “Everything was OK, but the eye tracker couldn’t find my eyes with the glasses. It was a pleasure attending to this test maybe the tracker could be a little bit more accurate.”